VIRTUAL PIANO

CZ4079: Final Year Project

Submitted in partial fulfilment of the requirements
for the degree of Bachelor of Computer Engineering
of Nanyang Technological University

Name:	**Tan Lai Chian Alan**
Supervisor:	**Mr Ravi Suppiah**
Examiner:	**Dr Vivek Chaturvedi**
Project Code:	**SCE15-0421**
School:	**Computer Engineering**
Matric No.:	**U1321110K**

School of Computer Engineering

AY 2015/2016

ABSTRACT

Smartphones are small but powerful devices, providing humans with countless uses, from making phone calls and taking photos, to accessing the Internet from almost anywhere. The late twentieth-century saw an explosion of computer applications. Through the installation of apps, the list of possible smartphone uses multiplies by tens of thousands and grows longer everyday. With smarter phone cameras coming to the market, there has been great interest in computer vision applications using real-time image processing in mobile devices.

This project develops an Android mobile application that uses real-time image processing techniques to provide a close simulation of playing the piano on a piece of paper with piano keys drawn or printed on it. The mobile device is placed on an elevated platform and at an angle for optimum view of the paper piano. When the application is started, the user switches on the piano and fingertip detection, fix the detected keys, and can then start playing on the paper piano. The application tracks fingertips for finger presses on the prefixed piano keys and then plays the corresponding sounds. The application is developed using OpenCV, a library for real-time image processing with over 2500 optimized algorithms. The application is tested on a Samsung Galaxy Note 4 under a single light condition in a small room. The empirical results showed that the proposed set of methodologies worked well for single fingertip keystrokes but failed partially for multiple fingertip keystrokes and more research is needed to implement a robust algorithm to accurately detect multiple keystrokes.

ACKNOWLEDGEMENTS

The author would like to express his heartfelt gratitude to all those people who contributed their time, energy, and effort into making this project possible.

The author would like to thank his supervisor, Mr Ravi Suppiah, for his guidance and for giving the author the opportunity to showcase his mobile application at NTU Open House 2016.

The author is also very grateful to his friends and family: Bryan Loy for lending his camera tripod, Tan Lai Hui for his Samsung Note 2, and Calvin Tan for his moral support at the beginning stage of the project.

TABLE OF CONTENTS

LIST OF FIGURES

1 INTRODUCTION

1.1 Background

Piano is a musical instrument that consists of 88 keys (collectively known as the keyboard) of which produce sounds of different tones and pitches when pressed. The piano was invented in 1700 by a harpsichord maker named Bartolomeo Cristofori, from Padua, in the Republic of Venice in northern Italy. “Piano” is short for “Pianoforte”, meaning that it can be played softer (piano) or louder (forte) in response to the player’s touch on the keyboard. Today, the piano is widely used in all types of music, including classical, popular, and jazz. There are three main types of pianos: Grand, Upright, and Electronic, with the upright piano being the most common type due to its cost, compactness and the warmness of its sound.

One of the barriers to playing the piano for recreation is the high cost of the piano itself. Many people may want to play a piano once in awhile, but are not always ready to spend more than a hundred dollars for that sake. Furthermore, owning a piano also requires space, regular maintenance and is not very portable.

Smartphones provide a good alternative to aforementioned problems with a tremendous cut in cost and convenience. The use of smartphones has become pervasive in our daily lives with a forecast of 2.08 billion users worldwide in 2016 [1]. Through musical smartphone applications, users can get a taste of playing the piano at a significantly low or even zero cost with limitations.

One type of piano mobile application lets users play by tapping on piano keys displayed on the mobile screen. However, such applications typically only cater up to one octave of piano keys. Moreover, each key has to be shrunk to a size smaller than that of a human’s finger in order to fit all the keys on the mobile screen. As such, there is a lack of realism in playing the piano using such mobile applications as users are unable to execute correct fingerings as they would on the real piano.

Several programs exist where image processing techniques are utilised to develop a novel form of playing the piano, commonly called the “Paper Piano”. A piano is drawn on a piece of paper for users to play on. By capturing images of the paper piano and the user’s fingers real-time through a video camera, the program applies image processing techniques to detect the piano and fingers, and plays sounds according to user’s keystrokes.

OpenCV (Open Source Computer Vision Library) is an open source computer vision software library that focuses on real-time vision applications. It is built to provide a common infrastructure for computer vision applications. Today, the library has more than 2500 optimized algorithms which can be used to detect and identify objects, track moving objects, and many more. OpenCV is written natively in C++ but also comes with interfaces in C, Python, Java, MATLAB and supports Windows, Linux, Android and Mac OS [14].

1.2 Statement of Problem

Little research has been done on piano mobile applications using image processing. Existing applications today still face problems such as failure to detect piano or fingers accurately, accidental keystrokes, latency, and more.

In addition, features such as multiple keystrokes, piano dynamics and articulations have yet to be fully explored.

1.3 Objectives

This project builds on current existing paper piano methodologies and encompasses the following objectives:

1) Propose a set of methodologies in piano, fingers, and touch detection, and mapping of touch to keystrokes for Virtual Piano.
2) Develop a functional Android mobile application of the paper piano using the proposed set of methodologies.

3) Implement dynamic touch detection through finger motion analysis in order to play piano dynamics (*p, mf, f* and *ff*)
4) Implement multiple fingertips detection and keystrokes using convex hull points and Douglas-Peucker points in order to play melodies, chords and harmonies using up to 5 fingers.

1.4 Report Organization

This report is organized into the following chapters:

Chapter 1: Introduction
This chapter covers the background information needed to understand the problem, outlines the statement of problem, and lists the objectives of this report.

Chapter 2: Literature Review
This chapter describes the related work and compares the findings with this project.

Chapter 3: Project Schedule
This chapter shows a chart-form project schedule that illustrates work to be done at various time intervals throughout the final year project duration.

Chapter 4: Design Documentation / Work Done
This chapter gives a full account of how the project work was carried out using the Incremental and Iterative software development model including phases of planning, requirement analysis and implementation.

Chapter 5: Data and Results
This chapter shows the results obtained from the application to verify the functionality of the application.

Chapter 6: Discussion of Results

This chapter discusses the inferences from observations found from Chapter 5.

Chapter 7: Conclusion

This chapter concludes the report and recommends improvement for the application in the future.

2 LITERATURE REVIEW

2.1 Piano Dynamics

In music, dynamics (or relative loudness) are instructions in musical notation to the performer about hearing the loudness of a note or phrase. More generally, dynamics may also include other aspects of the execution of a given piece [15].

The two basic dynamic indications in music are ***p*** (*piano)* and ***f*** *(forte)*, meaning “soft” and “loud” respectively. More subtle degrees of loudness or softness are indicated by ***mp*** *(mezzo-piano)*, and ***mf*** *(mezzo-forte)*, meaning “moderately soft” and “moderately loud” respectively. Beyond ***f*** and ***p***, there are also **pp** (pianissimo), **ff** (fortissimo), **ppp** (pianississimo) and **fff** (fortississimo), standing for very soft, very loud, very very soft, and very very loud respectively.

Playing dynamics on the piano depends on the speed of finger press on the piano keys. A quick drop of the finger will produce a *forte* sound while a slow fall will produce a soft *piano* sound.

2.2 Piano Detection

Piano detection is concerned with object recognition, which involves the task of finding and identifying objects such as the paper piano in an image or video sequence. Object recognition algorithms rely on matching, learning, or pattern recognition algorithms using appearance-based or feature-based techniques [2]. One effective technique is image segmentation by color, also known as color segmentation.

Several paper piano projects in the past have used color segmentation [3-5] followed by border-following algorithm to generate contours of the keys on the paper piano [4][5]. This method is simple and proved to work well under different lighting conditions and camera angles.

Color segmentation using HSV (Hue, Saturation, Value) colour space is generally preferred over RGB (Red, Green, Blue) colour space as it is more robust to changes in illumination. If the luminance component of HSV is eliminated, the system can become desensitized to shadows or illumination effects [6].

2.3 Hand Detection

Skin detection is possibly the most commonly used form of hand recognition, but it is much more effective when used together with other forms of recognition to eliminate false positives such as background objects of a similar color [6]. There are several approaches to build a skin color detector with the most basic one being a static skin color based thresholding. It has been observed that skin color lies in a very narrow range in H channel of HSV colorspace. The lower and upper hue thresholds are selected as (0,33) respectively. The Hue range provided is a generic threshold that will cover all the possible skin colors [7].

A skin color detector should ideally only detect the skin colored pixels and reject the pixels belonging to other objects or background. However, using a simple threshold based approach would also falsely detect some non skin colored pixels belonging to background or objects with similar hue color as skin like wood etc. Vegar Østhus and Martin Stokkeland (2013) assumed the largest contour found as the hand in their project as a solution to identify the hand contour as a whole [8]. This method is feasible under a controlled environment by ensuring that the hand is the largest skin contour in the image frames.

2.4 Fingertip detection

In Augmented Piano Reality, Zaqout and others (2015) used the lowest point of the hand contour as the effective point to play the piano [4]. The lowest point could be quickly found by iterating through all the points of the hand contour once by comparing the y-coordinate values. This method used is simple and efficient, but could only be used to track a single fingertip.

Thresholds of curvature is another technique used to identify isolated fingertips, but do not deliver dependable performance when two fingers are merged into a single region or when one finger occludes another. This technique is prone to assigning false-positives to finger-like objects, particularly knuckles [9].

A popular fingertip identification method commonly used in gesture detection is to create a convex hull around the identified hand. The vertices of the convex hull are then identified as fingertip candidates and further narrowed down to actual fingertips using convexity defects [6][8]. This technique seems promising for our project as it is able to detect multiple fingers with higher accuracy as compared to shadow analysis.

2.5 Touch detection

Muhd (2015) proposed an alternative mechanism of image processing touch detection by recognizing finger presses using texture characteristics obtained from gray level co-occurrence values [3]. The application worked well under different lighting conditions and was able to support multiple key-presses. One advantage of this method was that there was no need for hand and fingertip detection in every frame which can be computationally costly. However, the application required texture sampling of a pressed finger for each key region at the initial stage. Accuracy of key-presses was also heavily affected by the predetermined threshold values. Furthermore, the key-press regions were also bounded by rectangles which did not cover the full regions of the actual piano keys. Finally, this method does not seem to allow detection of dynamic key presses where analysis of fingertip velocity and/or force is required.

Adajania and others (2010) proposed a method called shadow analysis for touch detection [9]. If the percentage of shadow pixels around a fingertip is smaller than a fixed percentage, it is concluded that the fingertip is in contact with the touch surface. The technique is computationally inexpensive, does not require any special hardware, and can be applied on a frame-by-frame basis. However, shadow analysis demands very strong assumptions about the lighting and hand poses that will appear in images. These demands did not necessarily produce dependable performance in touch detection.

Adajania and others (2010) suggested using motion analysis technique, such as in [10]. For example, given the location of a fingertip in several consecutive images, one could calculate the fingertip's velocity and identify key-presses as by considering when the fingertip's velocity reverses direction. A similar method was used in [4], whereby a keystroke was deemed to have occurred when the effective point of a finger moved down and then up again while staying on the same key, or if the effective point stayed at the same position (1-2 pixel difference) for two consecutive frames.

3 MATERIAL/EQUIPMENT RESOURCES AND COSTING

3.1 Equipment

Table 1 lists the equipment required for the use of Virtual Piano mobile application. Detailed description and minimum requirement (if exact type of equipment is not available) of each item are listed under Description column.

Table 1. Equipment

No	Name	Description	Quantity
1	Samsung Galaxy Note 4	Or any Android device with minimum API 11	1
2	Paper Piano	● A4 size paper ● Piano layout must be drawn using thick black lines ● White keys must be fully enclosed by black lines ● Black keys must be filled with solid black	1
3	Phone holder/tripod	● Phone camera must be placed at an angle of 70-80 degrees with 4-5 cm of elevation from table when put on tripod	1
4	Book	4-5 cm thick to provide elevation if phone tripod does not	1

	(Optional)	provide any elevation	

3.2 Hardware Requirements

1. Notebook PC

3.3 Software Requirements

1. Eclipse (Mars)
2. Android SDK
3. JDK 6 or 7
4. OpenCV 3.0 (Library)
5. SourceTree (For source code management) - Optional
6. Vysor (Beta) - Optional
7. Adobe PhotoShop (Or any photo editing software) - Optional

3.4 Set up

Figure 1 shows the environment setup for the use of the application. The phone is placed with 70-80 degrees angle tilt in front of the user and the paper piano on an elevated platform of about 4-5cm.

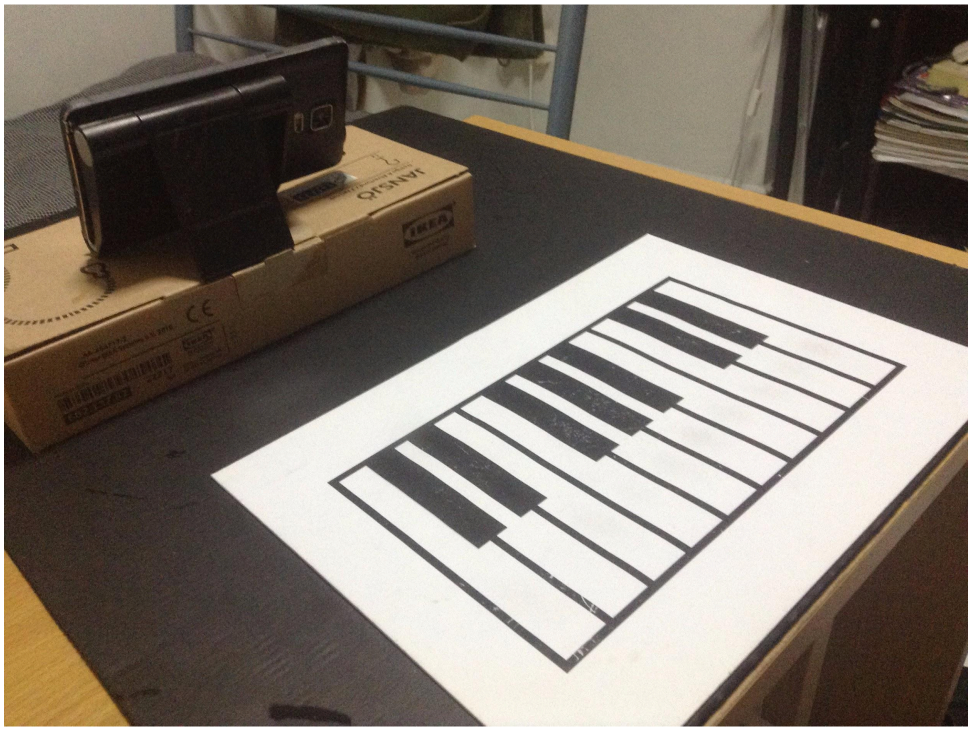


Figure 1. Set up for use of Virtual Piano mobile application.

3.5 Piano Layout

10 white keys and 7 black piano keys which cover slightly more than one octave of keys were printed on a paper as shown in Figure 2. This layout was used for development and testing.

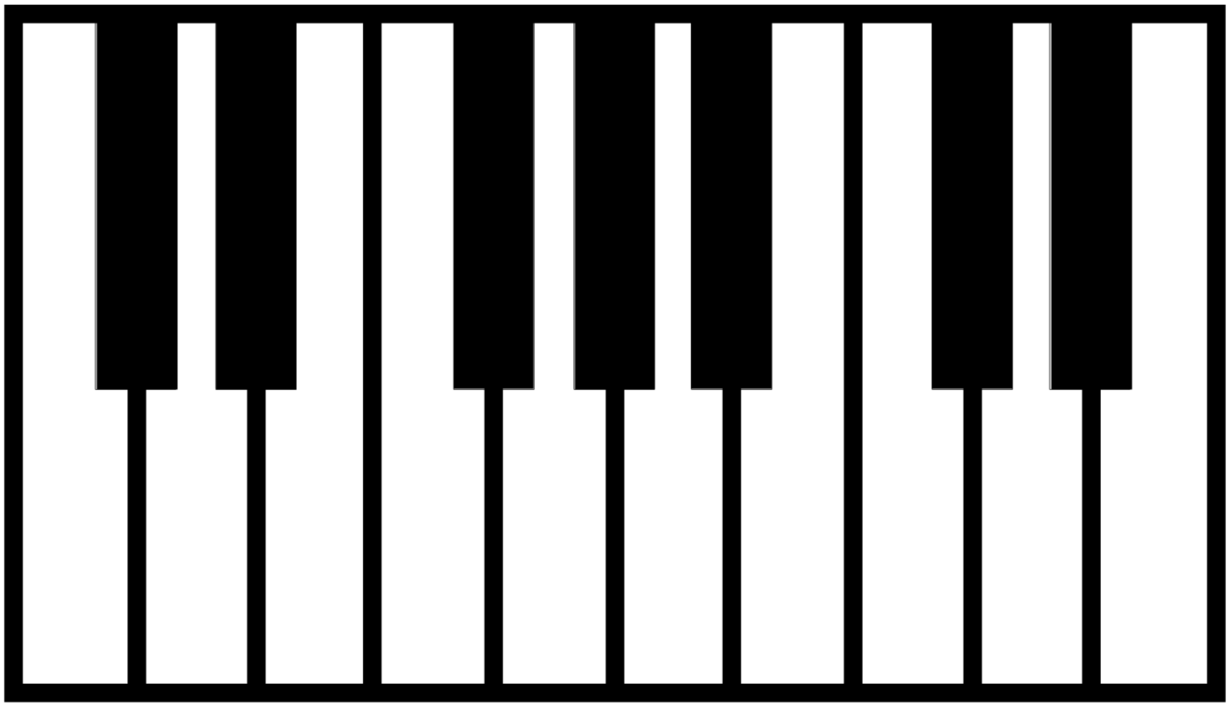
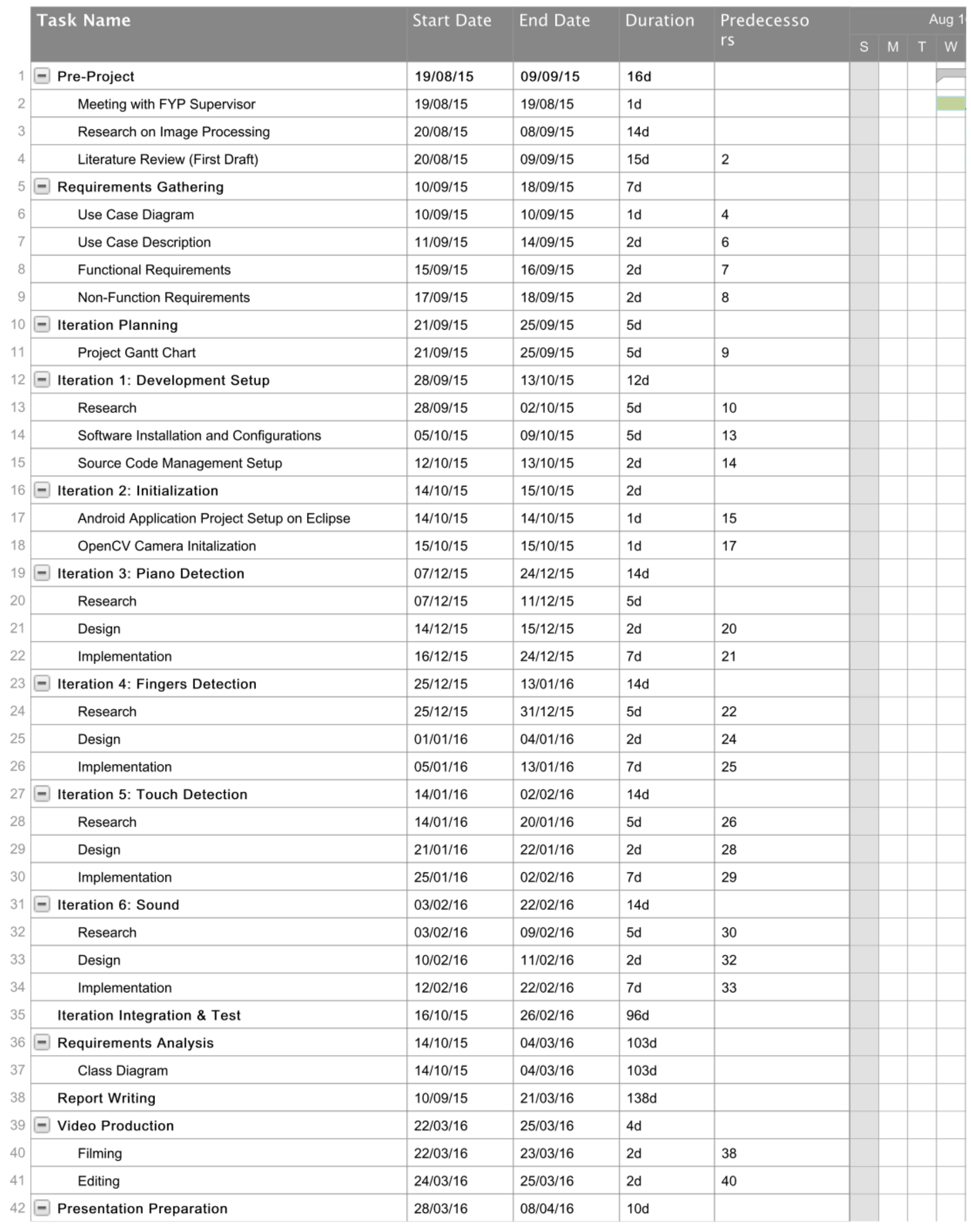


Figure 2. Piano layout.

4 PROJECT SCHEDULE

Figure 3 shows the entire schedule for this final year project. The work was planned in this manner based on the incremental and iterative methodology. Hence, the project was completed in iterations.



	Task Name	Start Date	End Date	Duration	Predecessors
1	**Pre-Project**	**19/08/15**	**09/09/15**	**16d**	
2	Meeting with FYP Supervisor	19/08/15	19/08/15	1d	
3	Research on Image Processing	20/08/15	08/09/15	14d	
4	Literature Review (First Draft)	20/08/15	09/09/15	15d	2
5	Requirements Gathering	10/09/15	18/09/15	7d	
6	Use Case Diagram	10/09/15	10/09/15	1d	4
7	Use Case Description	11/09/15	14/09/15	2d	6
8	Functional Requirements	15/09/15	16/09/15	2d	7
9	Non-Function Requirements	17/09/15	18/09/15	2d	8
10	Iteration Planning	21/09/15	25/09/15	5d	
11	Project Gantt Chart	21/09/15	25/09/15	5d	9
12	Iteration 1: Development Setup	28/09/15	13/10/15	12d	
13	Research	28/09/15	02/10/15	5d	10
14	Software Installation and Configurations	05/10/15	09/10/15	5d	13
15	Source Code Management Setup	12/10/15	13/10/15	2d	14
16	Iteration 2: Initialization	14/10/15	15/10/15	2d	
17	Android Application Project Setup on Eclipse	14/10/15	14/10/15	1d	15
18	OpenCV Camera Initalization	15/10/15	15/10/15	1d	17
19	Iteration 3: Piano Detection	07/12/15	24/12/15	14d	
20	Research	07/12/15	11/12/15	5d	
21	Design	14/12/15	15/12/15	2d	20
22	Implementation	16/12/15	24/12/15	7d	21
23	Iteration 4: Fingers Detection	25/12/15	13/01/16	14d	
24	Research	25/12/15	31/12/15	5d	22
25	Design	01/01/16	04/01/16	2d	24
26	Implementation	05/01/16	13/01/16	7d	25
27	Iteration 5: Touch Detection	14/01/16	02/02/16	14d	
28	Research	14/01/16	20/01/16	5d	26
29	Design	21/01/16	22/01/16	2d	28
30	Implementation	25/01/16	02/02/16	7d	29
31	Iteration 6: Sound	03/02/16	22/02/16	14d	
32	Research	03/02/16	09/02/16	5d	30
33	Design	10/02/16	11/02/16	2d	32
34	Implementation	12/02/16	22/02/16	7d	33
35	Iteration Integration & Test	16/10/15	26/02/16	96d	
36	Requirements Analysis	14/10/15	04/03/16	103d	
37	Class Diagram	14/10/15	04/03/16	103d	
38	Report Writing	10/09/15	21/03/16	138d	
39	Video Production	22/03/16	25/03/16	4d	
40	Filming	22/03/16	23/03/16	2d	38
41	Editing	24/03/16	25/03/16	2d	40
42	Presentation Preparation	28/03/16	08/04/16	10d	

Figure 3. Project Schedule.

5 DESIGN DOCUMENTATION

5.1 Project Lifecycle

The software development lifecycle model chosen for this project was the incremental and iterative model, where the application was designed, implemented and tested incrementally (a little more is added each time) until the complete application was finished.

The application was decomposed into several small components, each of which was designed and built separately. This model of development also helped ease the traumatic effect of building a complete application from scratch all at once.

Using this model encompassed two major advantages. Firstly, regression testing was conducted after each iteration. During this testing, faulty elements of the application were quickly identified because few changes were made within each single iteration. Secondly, it was easier to test and debug as compared to other methods of software development because relatively smaller changes were made during each iteration. This allowed for more targeted and rigorous testing of each component within the overall application.

5.2 Requirements Elicitation

This section documents the following areas through requirements elicitation:

1) Functional Requirements
2) Non Functional Requirements
3) Quality Attributes
4) Use Case Diagram
5) Use Case Description

5.2.1 Functional Requirements

1. System shall display the camera image on the screen upon opening of the application
2. System shall provide a button for user to toggle between front and rear camera

3. System shall provide a button for user to toggle piano detection
    a. System shall display blue solid contours on top of all white keys when piano detection is turned on
    b. System shall display red solid contours on top of all black keys when piano detection is turned on
    c. System shall stop displaying both red and blue solid contours when piano detection is turned off
4. System shall provide a button for user to set the piano keys upon successful detection of piano keys
    a. System shall draw yellow outlines of all white keys when piano layout is set
    b. System shall draw green outlines of all black keys when piano layout is set
5. System shall provide a button for user to toggle hand detection
    a. System shall display convex hull outline in blue when hand detection is turned on
    b. System shall stop displaying convex hull outline when hand detection is turned off
6. System shall provide a button for user to toggle between one and two hands mode
    a. System shall detect and display convex hull outline in blue when second hand is detected during two-hands mode.
    b. System shall stop detecting second hand when in one-hand mode.
7. System shall provide a button for user to toggle between single and multiple fingers mode
    a. System shall draw a red circle on the effective fingertip(s)
8. System shall provide a button for user to toggle between detection of piano layout 1 and piano layout 2
9. System shall provide a dropdown menu for user to change camera size
    a. Size options shall be displayed in ‘(width) x (height)’ format, e.g. ‘640 x 480’.
10. System shall play a note that corresponds to the key pressed by the user on the paper piano

5.2.2 Non-Functional Requirements

1. System shall not crash if no piano keys are detected when piano detection mode is on.
2. System shall not crash if no hand(s) or finger(s) is/are detected when finger detection mode is on.
3. In the event of any other failure, system shall restart the application. Application shall not close and stop working entirely.
4. System shall check for availability of front facing camera and display the option to toggle between front and rear camera on the menu if so.
5. The function of each buttons shall be easily understood by the user intuitively.
6. User shall be able to infer the buttons to press in order to play on the paper piano by the button labels.

5.2.3 Quality Attributes

Two quality attributes were identified for Virtual Piano mobile application as follow, which were fulfilled through the functional and nonfunctional requirements listed above:

1. Usability
2. Correctness

5.2.4 Use Case Diagram

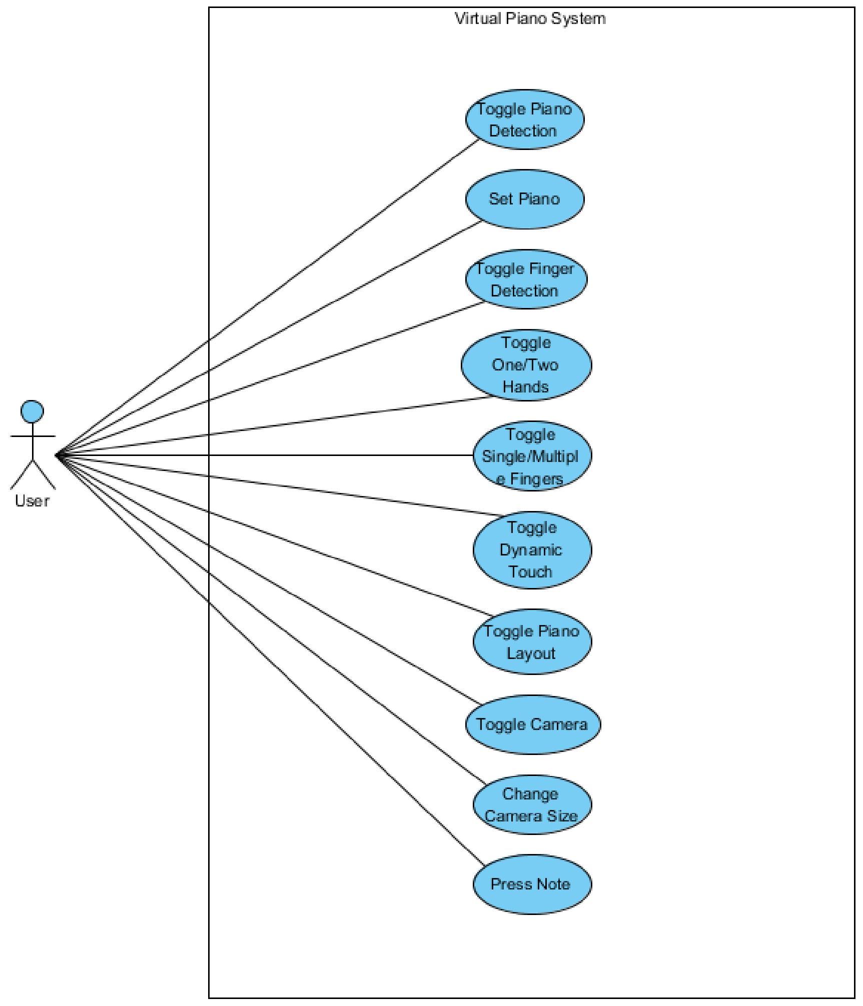


Figure 4. Use Case Diagram of Virtual Piano System

5.2.5 Use Case Description

All use case descriptions can be found under Appendix 1 at the end of this report.

5.3 Class Diagram

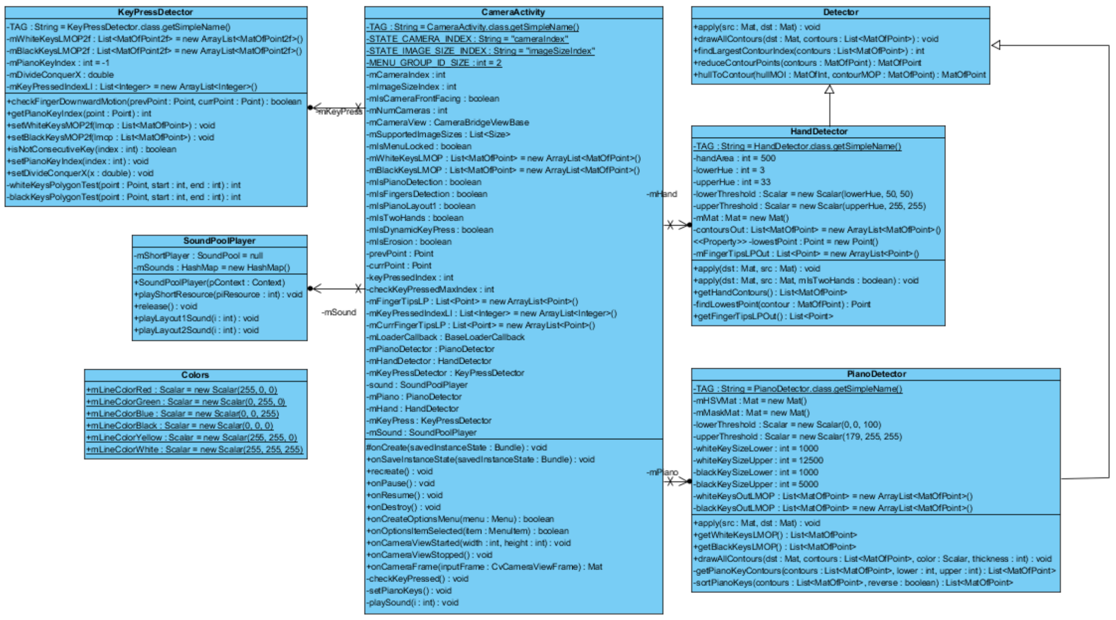

KeyPressDetector
-TAG : String = KeyPressDetector.class.getSimpleName()
-mWhiteKeysLMOP2f : List<MatOfPoint2f> = new ArrayList<MatOfPoint2f>()
-mBlackKeysLMOP2f : List<MatOfPoint2f> = new ArrayList<MatOfPoint2f>()
-mPianoKeyIndex : int = -1
-mDivideConquerX : double
-mKeyPressedIndexLI : List<Integer> = new ArrayList<Integer>()
+checkFingerDownwardMotion(prevPoint : Point, currPoint : Point) : boolean
+getPianoKeyIndex(point : Point) : int
+setWhiteKeysMOP2f(lmop : List<MatOfPoint>) : void
+setBlackKeysMOP2f(lmop : List<MatOfPoint>) : void
+isNotConsecutiveKey(index : int) : boolean
+setPianoKeyIndex(index : int) : void
+setDivideConquerX(x : double) : void
-whiteKeysPolygonTest(point : Point, start : int, end : int) : int
-blackKeysPolygonTest(point : Point, start : int, end : int) : int
-mKeyPress
SoundPoolPlayer
-mShortPlayer : SoundPool = null
-mSounds : HashMap = new HashMap()
+SoundPoolPlayer(pContext : Context)
+playShortResource(piResource : int) : void
+release() : void
+playLayout1Sound(i : int) : void
+playLayout2Sound(i : int) : void
-mSound
Colors
+mLineColorRed : Scalar = new Scalar(255, 0, 0)
+mLineColorGreen : Scalar = new Scalar(0, 255, 0)
+mLineColorBlue : Scalar = new Scalar(0, 0, 255)
+mLineColorBlack : Scalar = new Scalar(0, 0, 0)
+mLineColorYellow : Scalar = new Scalar(255, 255, 0)
+mLineColorWhite : Scalar = new Scalar(255, 255, 255)
CameraActivity
-TAG : String = CameraActivity.class.getSimpleName()
-STATE_CAMERA_INDEX : String = "cameraIndex"
-STATE_IMAGE_SIZE_INDEX : String = "imageSizeIndex"
-MENU_GROUP_ID_SIZE : int = 2
-mCameraIndex : int
-mImageSizeIndex : int
-mIsCameraFrontFacing : boolean
-mNumCameras : int
-mCameraView : CameraBridgeViewBase
-mSupportedImageSizes : List<Size>
-mIsMenuLocked : boolean
-mWhiteKeysLMOP : List<MatOfPoint> = new ArrayList<MatOfPoint>()
-mBlackKeysLMOP : List<MatOfPoint> = new ArrayList<MatOfPoint>()
-mIsPianoDetection : boolean
-mIsFingersDetection : boolean
-mIsPianoLayout1 : boolean
-mIsTwoHands : boolean
-mIsDynamicKeyPress : boolean
-mIsErosion : boolean
-prevPoint : Point
-currPoint : Point
-keyPressedIndex : int
-checkKeyPressedMaxIndex : int
-mFingerTipsLP : List<Point> = new ArrayList<Point>()
-mKeyPressedIndexLI : List<Integer> = new ArrayList<Integer>()
-mCurrFingerTipsLP : List<Point> = new ArrayList<Point>()
-mLoaderCallback : BaseLoaderCallback
-mPianoDetector : PianoDetector
-mHandDetector : HandDetector
-mKeyPressDetector : KeyPressDetector
-sound : SoundPoolPlayer
-mPiano : PianoDetector
-mHand : HandDetector
-mKeyPress : KeyPressDetector
-mSound : SoundPoolPlayer
#onCreate(savedInstanceState : Bundle) : void
+onSaveInstanceState(savedInstanceState : Bundle) : void
+recreate() : void
+onPause() : void
+onResume() : void
+onDestroy() : void
+onCreateOptionsMenu(menu : Menu) : boolean
+onOptionsItemSelected(item : MenuItem) : boolean
+onCameraViewStarted(width : int, height : int) : void
+onCameraViewStopped() : void
+onCameraFrame(inputFrame : CvCameraViewFrame) : Mat
-checkKeyPressed() : void
-setPianoKeys() : void
-playSound(i : int) : void
-mHand
-mPiano
Detector
+apply(src : Mat, dst : Mat) : void
+drawAllContours(dst : Mat, contours : List<MatOfPoint>) : void
+findLargestContourIndex(contours : List<MatOfPoint>) : int
+reduceContourPoints(contours : MatOfPoint) : MatOfPoint
+hullToContour(hullMOI : MatOfInt, contourMOP : MatOfPoint) : MatOfPoint
HandDetector
-TAG : String = HandDetector.class.getSimpleName()
-handArea : int = 500
-lowerHue : int = 3
-upperHue : int = 33
-lowerThreshold : Scalar = new Scalar(lowerHue, 50, 50)
-upperThreshold : Scalar = new Scalar(upperHue, 255, 255)
-mMat : Mat = new Mat()
-contoursOut : List<MatOfPoint> = new ArrayList<MatOfPoint>()
<<Property>> -lowestPoint : Point = new Point()
-mFingerTipsLPOut : List<Point> = new ArrayList<Point>()
+apply(dst : Mat, src : Mat) : void
+apply(dst : Mat, src : Mat, mIsTwoHands : boolean) : void
+getHandContours() : List<MatOfPoint>
-findLowestPoint(contour : MatOfPoint) : Point
+getFingerTipsLPOut() : List<Point>
PianoDetector
-TAG : String = PianoDetector.class.getSimpleName()
-mHSVMat : Mat = new Mat()
-mMaskMat : Mat = new Mat()
-lowerThreshold : Scalar = new Scalar(0, 0, 100)
-upperThreshold : Scalar = new Scalar(179, 255, 255)
-whiteKeySizeLower : int = 1000
-whiteKeySizeUpper : int = 12500
-blackKeySizeLower : int = 1000
-blackKeySizeUpper : int = 5000
-whiteKeysOutLMOP : List<MatOfPoint> = new ArrayList<MatOfPoint>()
-blackKeysOutLMOP : List<MatOfPoint> = new ArrayList<MatOfPoint>()
+apply(src : Mat, dst : Mat) : void
+getWhiteKeysLMOP() : List<MatOfPoint>
+getBlackKeysLMOP() : List<MatOfPoint>
+drawAllContours(dst : Mat, contours : List<MatOfPoint>, color : Scalar, thickness : int) : void
-getPianoKeyContours(contours : List<MatOfPoint>, lower : int, upper : int) : List<MatOfPoint>
-sortPianoKeys(contours : List<MatOfPoint>, reverse : boolean) : List<MatOfPoint>

Figure 5. Class Diagram of Virtual Piano System

5.4 Implementation

The implementation of this project was categorized into 5 major steps as follow:

1. Initialization (CameraActivity.java)
2. Piano Detection (PianoDetector.java)
3. Finger Detection (HandDetector.java)
4. Touch Detection (KeyPressDetector.java)
5. Play Sound (SoundPoolPlayer.java)

5.4.1 Initialization

The camera was set up by creating an instance of OpenCV’s abstract class called CameraBridgeViewBase, which represents a live camera feed. The default camera size was set to 640 x 480.

Captured frames from the video stream were converted to RGB color format (Figure 6) for use in the subsequent major steps.

5.4.2 Piano Detection

The 3 main steps taken for this section were detection of white keys, computation of piano mask around white keys, and detection of black keys.

First, the RGB frames were converted to HSV color space. Figure 6 and 7 shows the images before and after RGB to HSV color conversion.

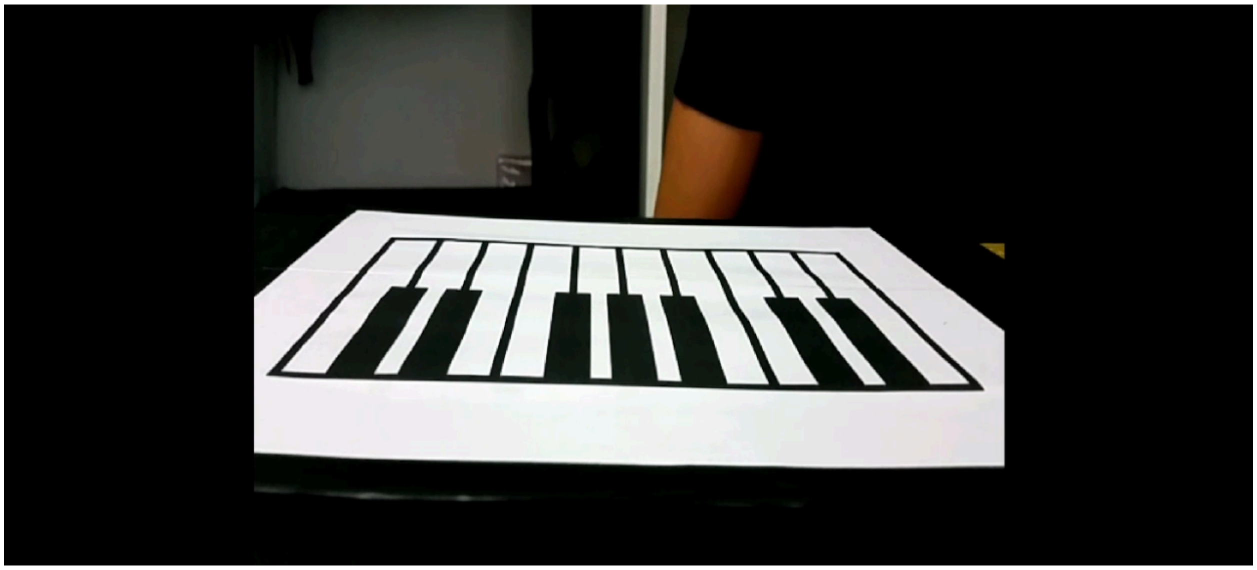


Figure 6. RGB image of captured frames from video stream.

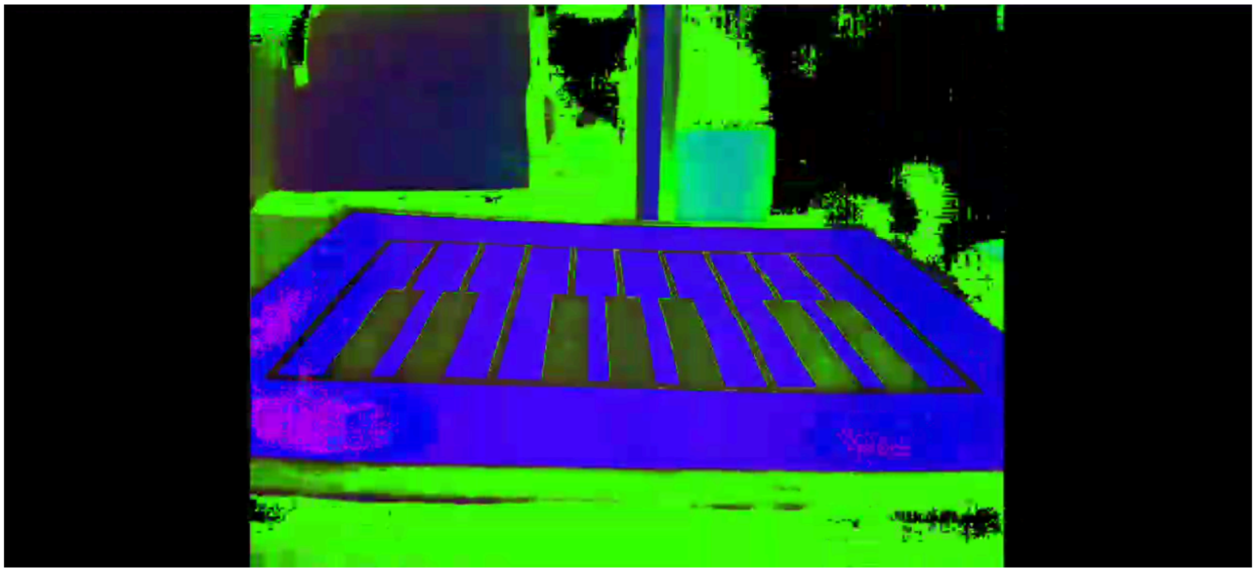


Figure 7. HSV image after color conversion.

White-colored objects were then extracted by applying HSV thresholding. It was observed that the Value of white-colored objects tended to fall in the range of 100 to 255. Hence, the lower and upper HSV threshold values used were (0, 0, 100) and (179, 255, 255) respectively.

The function *inRange* in opencv's core library is a thresholding method that reduces the given image to a binary image. It converts any pixels within the lower and upper threshold to white and the rest of the pixels to black. Using this thresholding method, we were left with a binary image to work with (Figure 8).

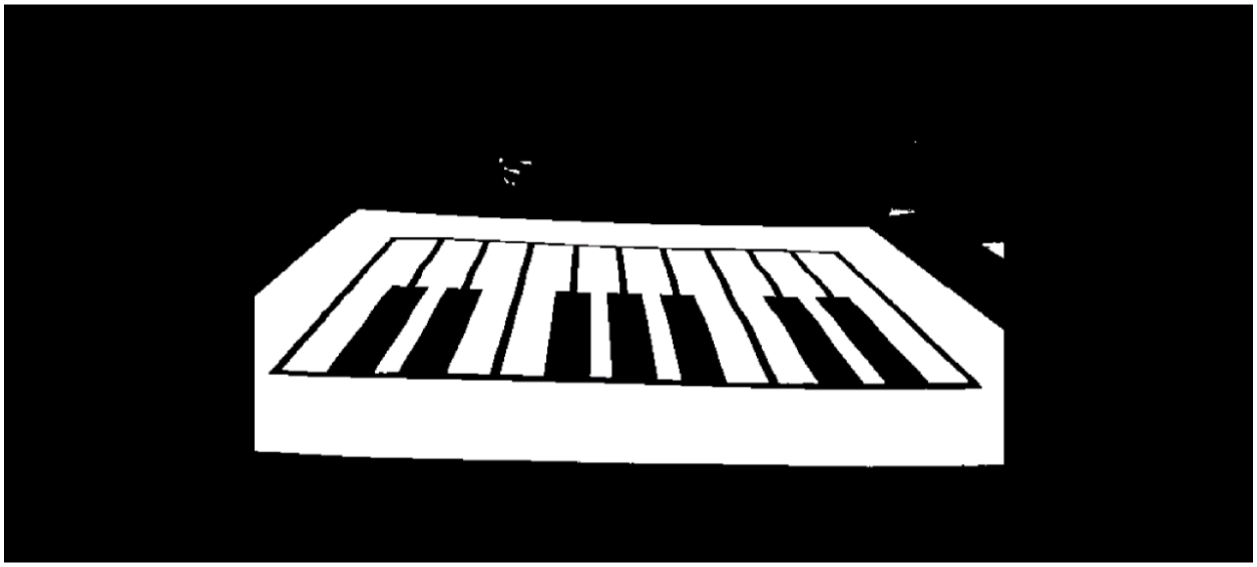
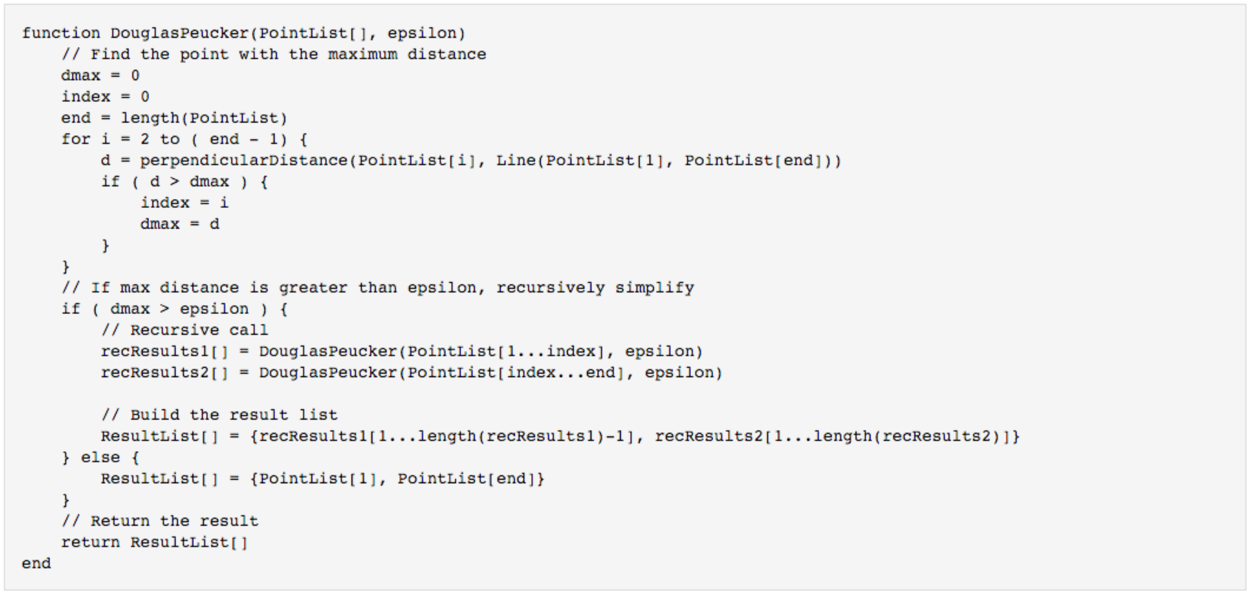


Figure 8. Binary image after thresholding in HSV.

From the binary image, contour detection on the white regions was applied using border-following algorithm. The points of each contour were then reduced using Douglas-Peucker algorithm (Figure 9).



```
function DouglasPeucker(PointList[], epsilon)
    // Find the point with the maximum distance
    dmax = 0
    index = 0
    end = length(PointList)
    for i = 2 to ( end - 1) {
        d = perpendicularDistance(PointList[i], Line(PointList[1], PointList[end]))
        if ( d > dmax ) {
            index = i
            dmax = d
        }
    }
    // If max distance is greater than epsilon, recursively simplify
    if ( dmax > epsilon ) {
        // Recursive call
        recResults1[] = DouglasPeucker(PointList[1...index], epsilon)
        recResults2[] = DouglasPeucker(PointList[index...end], epsilon)

        // Build the result list
        ResultList[] = {recResults1[1...length(recResults1)-1], recResults2[1...length(recResults2)]}
    } else {
        ResultList[] = {PointList[1], PointList[end]}
    }
    // Return the result
    return ResultList[]
end
```

Figure 9. Pseudo code for Douglas-Peucker algorithm [11].

The area of each contour was calculated and eliminated if it did not fall within a specified area threshold. The lower and upper area thresholds for white keys were 1000 and 12500 respectively. This threshold range was found to be effective for detecting piano keys correctly while maximizing placement of the paper piano within the camera frame. It was also effective in eliminating small or large non piano key objects in the background. Figure 10 below displays the final contours of the white keys on the paper piano.

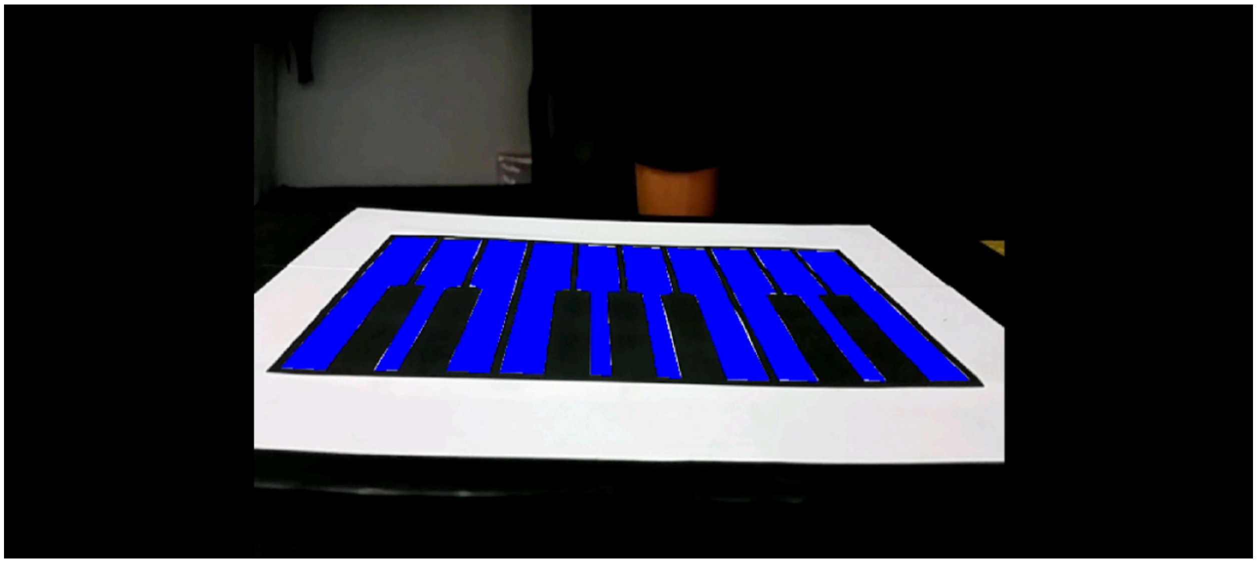


Figure 10. Final contours of white keys.

The remaining contours were sorted from left to right using merge sort algorithm (Figure 11). Merge sort is a fast and efficient sorting method which always runs in Θ(n lgn) time. The sorting was done with respect to the top left points of each contour's bounding rectangle.

```
func mergesort( var a as array )
     if ( n == 1 ) return a

     var l1 as array = a[0] ... a[n/2]
     var l2 as array = a[n/2+1] ... a[n]

     l1 = mergesort( l1 )
     l2 = mergesort( l2 )

     return merge( l1, l2 )
end func

func merge( var a as array, var b as array )
     var c as array

     while ( a and b have elements )
          if ( a[0] > b[0] )
               add b[0] to the end of c
               remove b[0] from b
          else
               add a[0] to the end of c
               remove a[0] from a
     while ( a has elements )
          add a[0] to the end of c
          remove a[0] from a
     while ( b has elements )
          add b[0] to the end of c
          remove b[0] from b
     return c
end func
```

Figure 11. Merge sort algorithm [12].

The convex hull of the white keys was then computed and used as a piano mask for detecting the black piano keys subsequently (Figure 12).

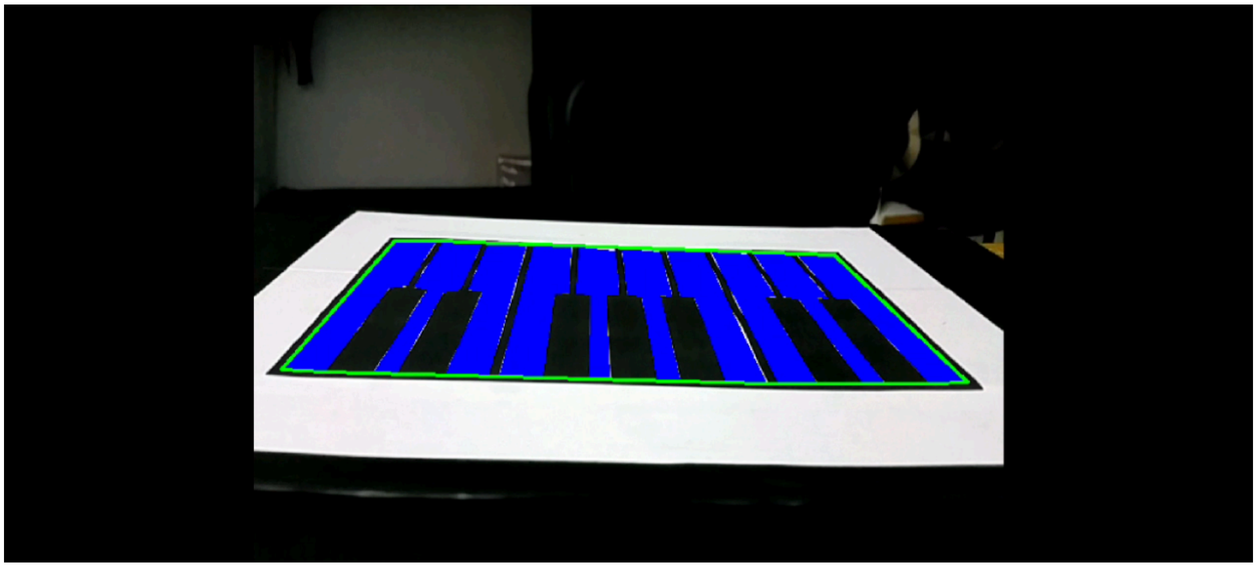


Figure 12. Convex hull of white keys drawn in green outline.

For detection of black keys, the same algorithm used to detect white keys could be applied by simply inverting the binary image previously found. However, there was a need to remove unwanted lines in the piano layout that were also black in color.

Morphological dilation was applied to the binary image. Through dilation, we were able to remove the unwanted piano lines by shrinking the size of black regions while expanding the size of white regions as shown in Figure 13. The image was then inverted and dilated to get the resultant image shown in Figure 14.

Piano mask was then applied to get the final image in Figure 15 which was suitable for detecting black keys by following the same contour detection steps used for the white keys.

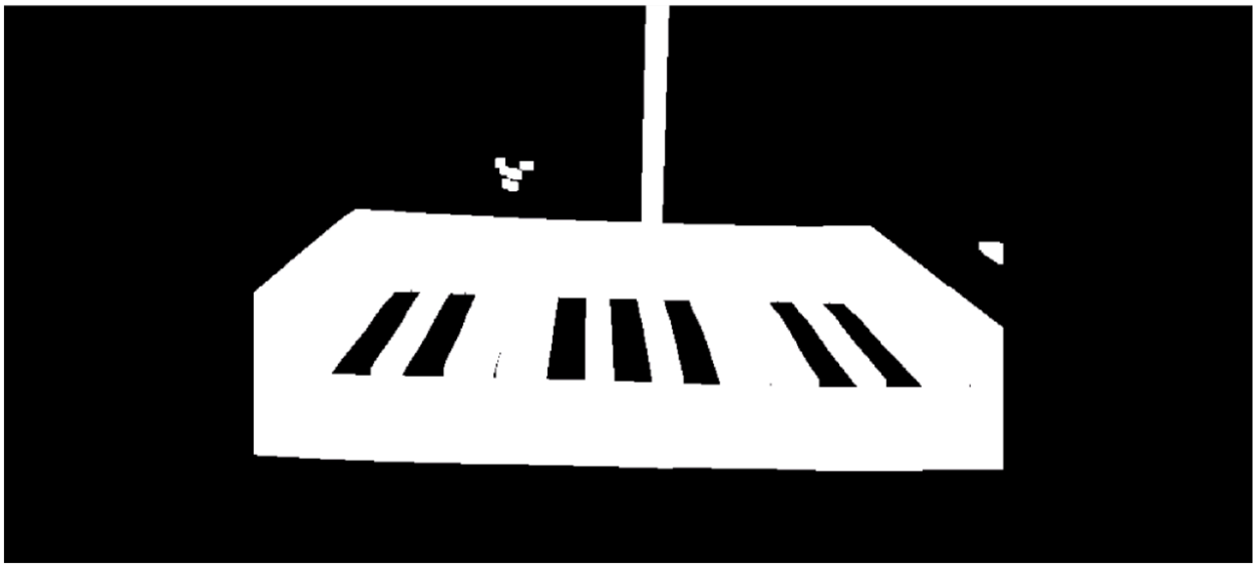


Figure 13. Dilated image to remove unwanted black line of the piano layout.

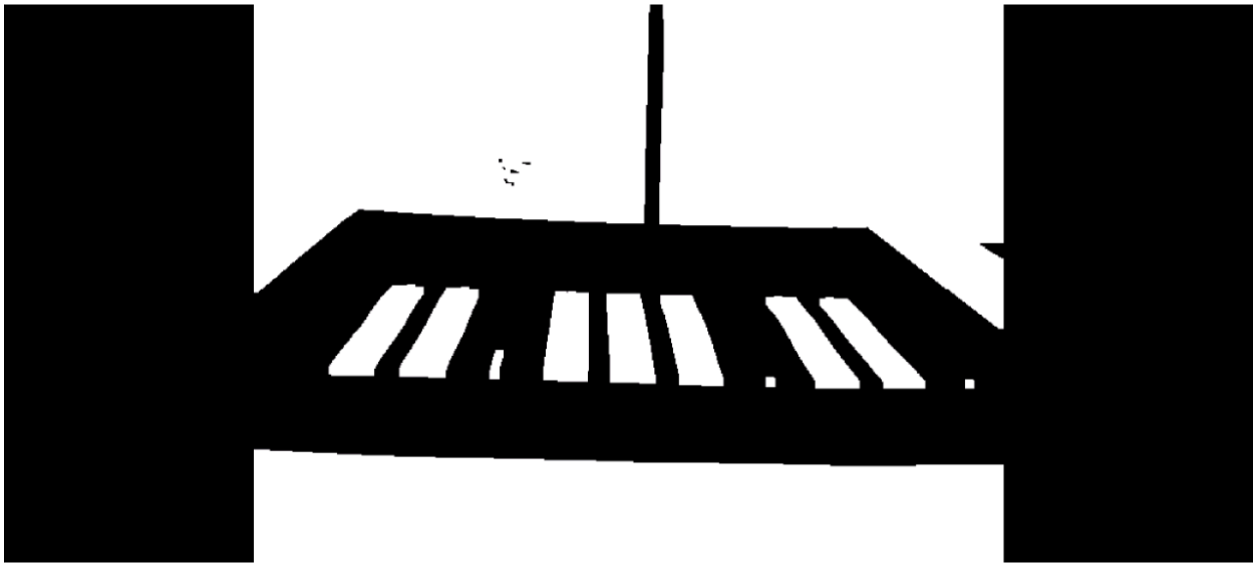


Figure 14. Inverted and dilation result of Figure 13.

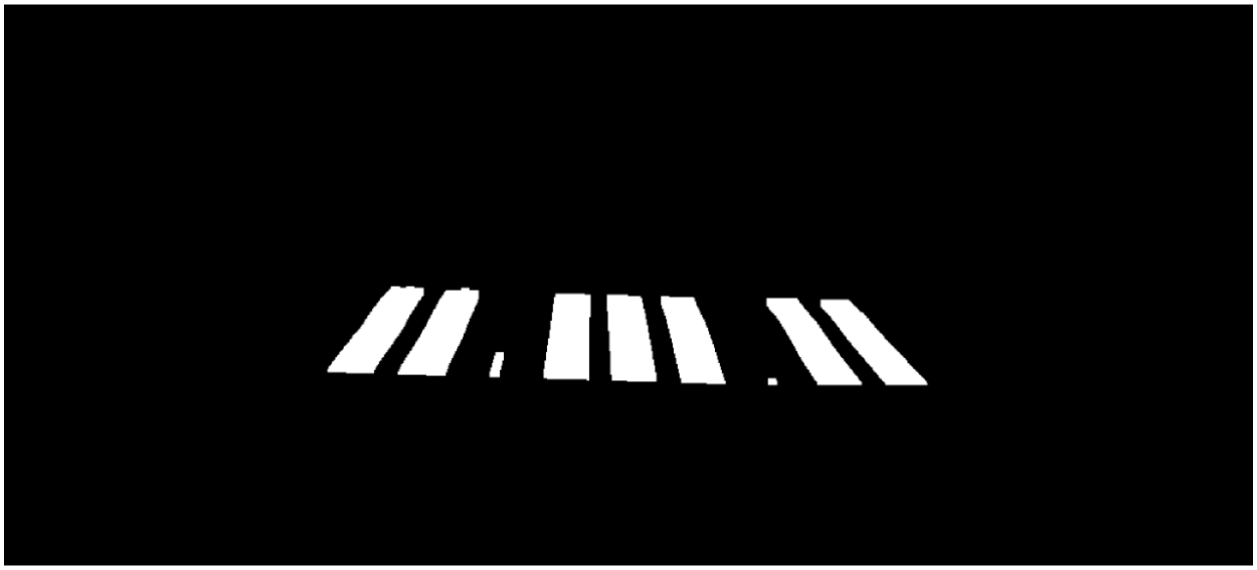


Figure 15. After applying piano mask.

5.4.3 Fingertip Detection

The steps taken in this section were skin detection, hand detection, and fingertip detection. Skin-colored pixels fell in the lower and upper HSV threshold range of (3, 50, 50) and (30, 255, 255) respectively [7]. Figure 16 shows the result of HSV thresholding using the aforementioned threshold range. Using the same steps taken for piano keys detection, skin-colored contours were computed using border-following algorithm and refined using Douglas-Peucker algorithm. The contour with the largest area amongst all was then assumed to be the hand contour (Figure 17).

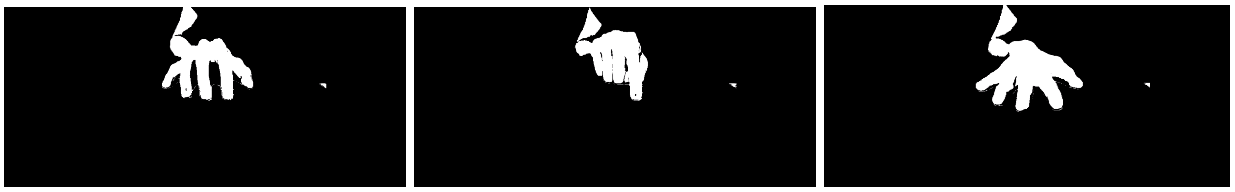


Figure 16. Image after applying static skin color based thresholding in HSV color space.

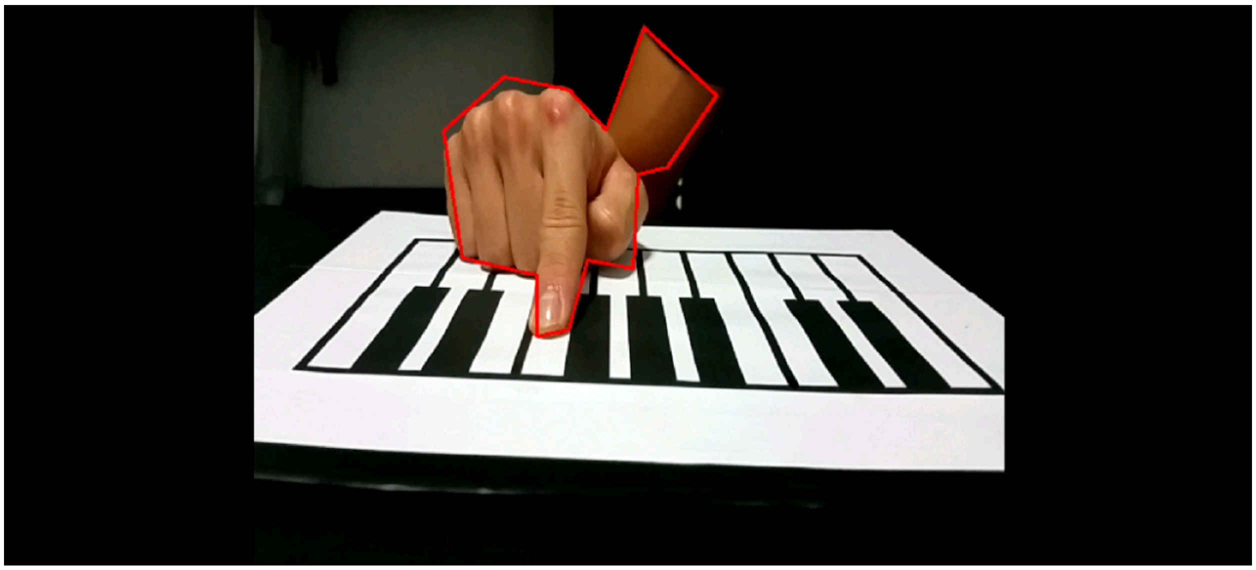


Figure 17. Hand contour

Four different methods of fingertip tracking were explored as follows:

1) One hand, single finger
2) Two hands, single finger each
3) One hand, multiple fingers
4) Two hands, multiple fingers

5.4.3.1 One hand with single finger

The lowest point of the hand contour was taken to be the effective fingertip point used for playing the piano. This algorithm was sufficient for playing the piano using one finger.

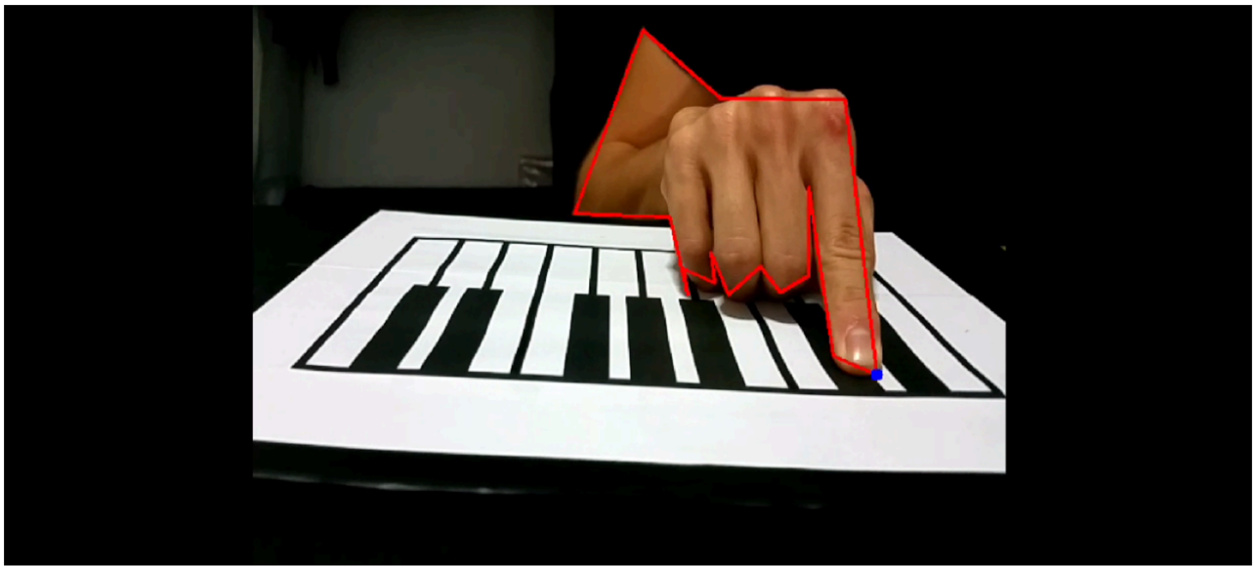


Figure 18. Detection of single fingertip based on lowest point of hand contour.

5.4.3.2 Two hands with single finger each

The second largest contour found during skin detection was taken to be the second hand contour. The lowest point of the hand contour was then taken to be the effective fingertip point as well.

5.4.3.3 One hand with multiple fingers

The convex hull of the hand contour was found in order to locate the finger tips. As seen in Figure 19, the fingertip points fell in the piano mask region. Therefore, the remaining points were taken to be fingertip points after eliminating points that were outside the piano region.
As the app was a real-time application and speed was of concern, the amount of computation was best kept minimum. Hence, further computations to label finger indexes or finding hand centroid were unnecessary.

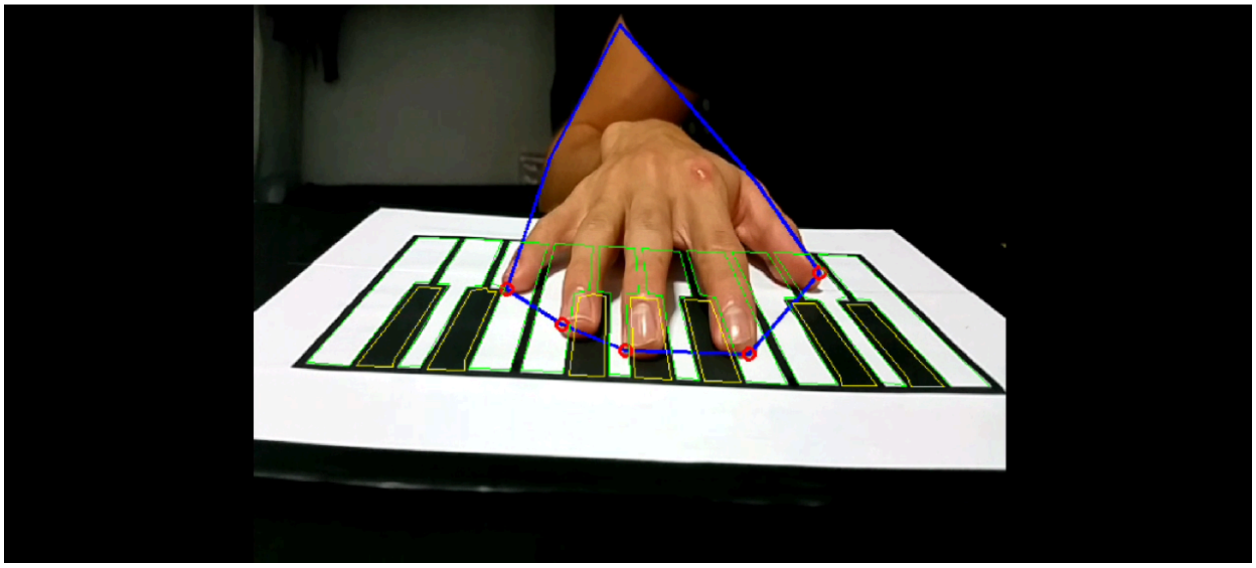


Figure 19. Convex hull of hand and multiple fingertips detection

5.4.3.4 Two hands with multiple fingers

Similar to the two hands with single finger methodology, two hands with multiple fingers mode was developed by duplicating the steps taken for single hand with multiple finger and applying to the second largest hand contour detected.

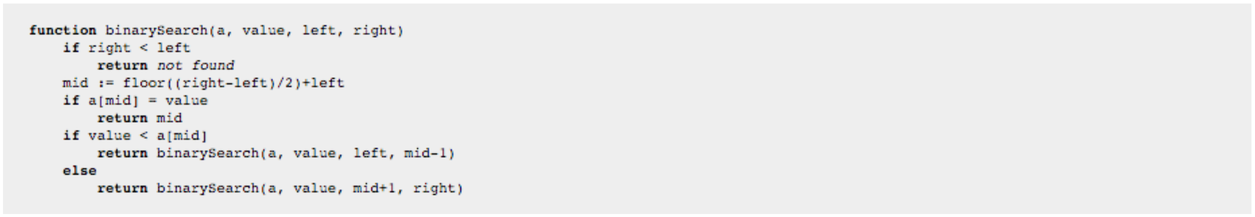
5.4.4 Touch Detection

Effective touches were deemed as movement of fingertips in a downward motion. This was tracked by checking the y-coordinate difference of fingertip points between 2 consecutive frames (frame A and frame B). When the y-coordinate difference was higher than a certain integer value, a touch was considered to have happened.

5.4.5 Mapping of Touch to Keystrokes

The fingertip point in frame B was then checked to see if it fell within any piano key contour region using Point-In-Polygon algorithm and if so, the piano key index was returned. A simple binary search algorithm was applied to reduce the number of Point-In-Polygon tests which effectively reduced the search complexity by O(lg n).

Binary search (Figure 20) is an efficient algorithm for searching through an ordered list of elements. It works by recursively dividing the list into halves, keeping only the half that could contain the item, until the wanted element is narrowed down to just one possible location.



```
function binarySearch(a, value, left, right)
    if right < left
        return not found
    mid := floor((right-left)/2)+left
    if a[mid] = value
        return mid
    if value < a[mid]
        return binarySearch(a, value, left, mid-1)
    else
        return binarySearch(a, value, mid+1, right)
```

Figure 20. Binary Search Algorithm [13].

After narrowing down the index of the pressed key in frame B, the index was compared with the pressed key index of the previous frame. If the indexes were the same, the app would abort playing of the note. It was discovered that when such a situation occurred, a touch could not have happened as the fingertip had not yet been lifted up to execute another touch on the same note. Hence, it was concluded that two successive touches could only happen within 3 consecutive frames due to the fluctuation of fingertip points caused by factors such as noise and shadows and were false positives of touches.

5.4.6 Dynamic Touch Detection

The relative loudness of each keystroke was determined by the y-coordinate difference between two consecutive frames as follow:

piano: 3 -10

mezzo-piano: 11 - 20

forte: 21 - 30

fortissimo : > 30

5.4.7 Sound

Android provides three classes that allow the app to play sounds; Media Player, Audio Track, and SoundPool. SoundPool class is suited for short sound clips and can play multiple sound clips simultaneously and hence was used in our application.

6 DATA AND RESULTS

The application is designed with Android 5.1 (Lollipop) being the intended target, and Android 3.0 (Honeycomb) being the minimum supported version. It was tested on a Samsung Galaxy Note 4 with Android 5.1.1 in a single room lighting condition as shown in Figure 1.

6.1 Test Cases

Successful piano detection was defined as correct detection of all piano keys. 1-Note Verification was defined as correct detection of a single fingertip and playing of the correct sound when a key was pressed. Table 2 shows the results for the test cases executed on our application.

Table 2. Application test result using Samsung Galaxy Note 4

Test Case	Result
Piano Detection	Passed
1-Note Verification	Passed
Dynamic touch	Passed
Playable Song	Passed
Accidental Keystrokes	Failed

6.2 Latency

Table 3 shows the results for latency of key presses in various finger detection modes. Latency is rated on a scale of 0 to 3:

A rating of 0 indicates lack of any perceivable lag.

A rating of 1 indicates low lag (0.5 to 1 second); playable

A rating of 2 indicates high lag (1 to 2 seconds); only functional

A rating of 3 indicates extreme lag; impossible to use the application

Table 3. Latency of key presses in various finger detection modes

Finger Mode	Latency
Single Finger	0
Multiple Fingers	0

6.3 Playing objectives in multiple fingers mode

Table 4 shows the results of the playing objectives when multiple fingertips were derived using Douglas-Peucker (DP) points and Convex Hull points of the hand contour.

Table 4. Results of playing objectives using DP and Convex Hull points to derive multiple fingertips

Objective	Result	
	DP	Convex Hull
Playing of harmonies	Failed	Passed
Playing of chords	Failed	Passed
Playing of melodies	Failed	Failed

7 DISCUSSION OF RESULTS

7.1 Proposed Methodology and Dynamic Touch

The proposed set of methodologies successfully detected all piano keys and single fingertip modes correctly. We were able to play a complete song in the single finger modes. The dynamic touch methodology picked up the correct loudness in response to our finger movements as well.

7.2 Accidental Keystrokes

The suggested keystroke methodology used in our application worked but with 2 types of accidental key press observed. First type of accidental key press was dual sound when playing a note. This was caused by fluctuation of fingertip position between three consecutive video frames due to false detection of shadow as the effective fingertip point. The convergence of a fingertip and its shadow during a finger press caused the effective fingertip point to jump down to the shadow and then back up in 3 consecutive frames. The small fluctuation was registered as a finger press by the KeyPress Detector module.

A new variable, keyPressedIndex, was introduced to solve this dual sound problem. The role of *keyPressedIndex* was to check for playing of the same piano key within 3 consecutive frames. It was analyzed that it was impossible to execute two key presses of the same note in 3 consecutive frames. Hence, such scenarios were declared as single keypresses instead of two. With the introduction of this new variable, the dual sound problem was solved.

The second type of accidental keystroke occurred during the movement of the fingertip from a white key to a black key. The movement of fingertip from a white key to a black key produced a significant y-coordinate difference which met the condition of a finger press.

Nonetheless, the new suggested keystroke methodology provided a more realistic simulation of key press on a real piano.

7.3 Multiple Fingers

The multiple fingertips detection algorithm using DP points of the hand contour failed the objectives of playing harmonies, chords and melodies. Using of Convex Hull points also failed in playing of a melody too but passed in playing of chords and harmonies.

For DP method, there were too many accidental convex points within the hand contour which caused the fingertip reduction algorithm to fail. The accidental convex points were caused by shadow of the hand which was mistaken as part of the hand during skin filtering.

For Convex Hull method, fingertip reduction algorithm correctly approximated the fingertip points and hence was able to play chords. However, since each fingertip need to be part of the convex hull to be detected, some fingertips would disappear during playing of melody when they do not form part of the convex hull. This algorithm does not adapt well to piano fingering techniques such as the ‘Thumb Under’.

8 CONCLUSIONS

In this research we proposed a set of methodologies to develop an Android mobile application of the paper piano using image processing techniques with OpenCV library. After analysing related work, we proposed to use HSV thresholding followed by border-following algorithm to detect piano keys and hand contours, and motion analysis techniques for touch detection.

We successfully developed a functional paper piano mobile application, detecting 10 white and 7 black keys with new features to detect multiple keystrokes and dynamic touch. Our application was tested on a Samsung Galaxy Note 4 smartphone in a single room lighting condition with varying performance depending on the number of hands and fingers used. The application displayed acceptable performance for single finger keystrokes with no perceivable lags and allowed users to play simple melodies on the paper piano. The application failed to detect multiple keystrokes efficiently to play melodies but was able to play harmony and chords using 2 and 3 fingers respectively. Our application fulfilled our first 3 objectives well and partially for the 4th objective. In conclusion, there is still room for further improvement and possible future work may focus on:

- Developing a more robust algorithm for multiple fingertips and touch detection for 5 fingers.
- More features to achieve a closer simulation of playing a real piano such as piano articulations for users to play staccato, accent, tenuto, etc.
- Improving the detection algorithms to better detect piano keys and hand in all lighting and background conditions.

REFERENCES

[1] Number of smartphone users* worldwide from 2014 to 2019 (in millions). (August 2015). Statista, The Statistics Portal. Retrieved 02:12, March 21, 2016, http://www.statista.com/statistics/330695/number-of-smartphone-users-worldwide/

[2] Object recognition methods in computer vision. (19 May 2012). In MathWorks. Retrieved 02:12, March 21, 2016, http://www.mathworks.com/discovery/object-recognition.html?requestedDomain=www.mathworks.com

[3] Muhd N. (2015). Virtual Piano. Nanyang Technological University.

[4] Zaqout, I., Elhissi, S., Jarour, A., & Elowini, H. (2015). Augmented Piano Reality. IJHIT, 8(10), 141-152. http://dx.doi.org/10.14257/ijhit.2015.8.10.13

[5] F. Huang, Y. Zhou, Y. Yu, Z. Wang and S. Du, "Piano AR: A Markerless Augmented Reality Based Piano Teaching System", Third International Conference on Intelligent Human-Machine Systems and Cybernetics (IHMSC), IEEE, Washington, DC, USA, vol. 2, (2011) August 28, pp. 47 – 52.

[6] X. Zabulist, H. Baltzakist and A. Argyrost, "Vision-based Hand Gesture Recognition for Human-COmputer Interaction," Crete.

[7] Farhad Dadgostar and Abdolhossein Sarrafzadeh. An Adaptive Real-time Skin Detector Based on Hue Thresholding: A Comparison on Two Motion Tracking Methods . In: Pattern Recogn. Lett. 27.12 (Sept. 2006), pp. 1342 1352. issn:0167-8655. doi: 10.1016/j.patrec.2006.01.007. url: http://dx.doi. org/10.1016/j.patrec.2006.01.007.

[8] Hand Tracking And Recognition with OpenCV (in millions). (12 August 2013). Retrieved 02:12, March 21, 2016, http://sa-cybernetics.github.io/blog/2013/08/12/hand-tracking-and-recognition-with-opencv/

[9] Y. Adajania, J. Gosalia, A. Kanade, H. Mehta, and N. Shekokar. Virtual keyboard using shadow analysis. In Emerging Trends in Engineering and Technology (ICETET), 2010 3rd International Conference on, page 163 to 165, 2010.

[10] Sumit Srivastava and Ramesh Chandra Tripathi. Real time mono-vision based customizable virtual keyboard using finger tip speed analysis. In Masaaki Kurosu, editor, Human-Computer Interaction. Interaction Modal- ities and Techniques, volume 8007 of Lecture Notes in Computer Science, page 497 to 505. Springer Berlin Heidelberg, 2013.

[11] Ramer–Douglas–Peucker algorithm. (2016, January 29). In Wikipedia, The Free Encyclopedia. Retrieved 08:22, December 19, 2015, from https://en.wikipedia.org/w/index.php?title=Ramer%E2%80%93Douglas%E2%80%93Peucker_algorithm&oldid=702273359

[12] Merge sort. (19 May 2012). In The Algorithmist. Retrieved 05:02, January 9, 2016, from http://www.algorithmist.com/index.php/Merge_sort#Pseudo-code

[13] Binary search. (10 April 2011). Code Codex, The Free Code Wiki. Retrieved 04:18, February 21, 2016, from http://www.codecodex.com/wiki/Binary_search#Pseudocode

[14] “OpenCV Documentation”, [Online] (2014) November 15, Available from: http://docs.opencv.org/modules/core/doc/intro.html.

[15] Dynamics (music). (2016, March 2). In Wikipedia, The Free Encyclopedia. Retrieved 02:52, March 21, 2016, from https://en.wikipedia.org/w/index.php?title=Dynamics_(music)&oldid=707837336

Appendix 1

Use Case ID:	1
Use Case Name:	Toggle Piano Detection

Actors:	User
Description:	User toggles on/off piano detection mode
Trigger:	User presses on ‘Detect Piano’ button
Preconditions:	Camera is directed at the paper piano
Postconditions:	1 Off to On: Blue solid polygons are drawn over detected white keys and red solid polygons are drawn over detected black keys. 2 On to Off: System stops detecting piano keys
Normal Flow:	1.1 User presses on ‘Detect Piano’ button 1.2 System executes postconditions
Alternative Flows:	
Exceptions:	1.0.E.1 System does not detect any piano keys 1.0.E.1.1 System does not draw any polygons
Includes:	-
Priority:	High
Frequency of Use:	High

Use Case ID:	2

Use Case Name:	Set Piano

Actors:	User
Description:	User instructs the system to register the detected contours as piano keys
Trigger:	User presses on ‘Set Piano’ button
Preconditions:	Piano detection mode is on
Postconditions:	System converts detected contours into piano keys and stores in the system
Normal Flow:	1.1 User presses on Set Piano’ button 1.2 System executes postconditions
Alternative Flows:	-
Exceptions:	1.0.E.1 System did not detect any piano keys 1.0.E.1.1 System does nothing
Includes:	Piano Detection
Priority:	High
Frequency of Use:	High

Use Case ID:	3
Use Case Name:	Toggle Finger Detection

Actors:	User
Description:	User toggles on/off finger detection mode
Trigger:	User presses on ‘Detect Finger’ button

Preconditions:	User’s hand(s) is inside camera’s view
Postconditions:	1 Off to On: Red outline is drawn around detected hand(s) and blue dot is drawn on detected fingertip point(s). 2 On to Off: System stops detecting hand(s) and finger(s)
Normal Flow:	1.1 User presses on ‘Detect Finger’ button 1.2 System executes postconditions
Alternative Flows:	-
Exceptions:	1.0.E.1 System does not detect any hand 1.0.E.1.1 System does not draw anything
Includes:	-
Priority:	High
Frequency of Use:	High

Use Case ID:	4
Use Case Name:	Toggle One/Two Hands

Actors:	User
Description:	User switches between one or two hands detection mode
Trigger:	User presses on ‘Hands’ button
Preconditions:	User’s hand(s) is inside camera’s view
Postconditions:	1 One to Two Hands : Red outline is drawn around second detected hand(s) and blue dot is drawn on detected fingertip point(s). 2 Two to One Hand: System stops detecting second hand

Normal Flow:	1.1 User presses on 'Hands' button 1.2 System executes postconditions
Alternative Flows:	-
Exceptions:	1.0.E.1 System does not detect first hand 1.0.E.1.1 System does not draw anything 1.0.E.2 System does not detect any second hand 1.0.E.2.1 System does not draw anything
Includes:	-
Priority:	High
Frequency of Use:	High

Use Case ID:	5
Use Case Name:	Toggle Single/Multiple Fingers

Actors:	User
Description:	User switches between single or multiple fingers detection mode
Trigger:	User presses on 'Fingers' button
Preconditions:	User's hand(s) is inside camera's view
Postconditions:	1 Single to Multiple : System draw blue dots over all fingertip points detected 2 Multiple to Single: System detects only single fingertip
Normal Flow:	1.1 User presses on Fingers button 1.2 System executes postconditions
Alternative Flows:	-

Exceptions:	1.0.E.1 System does not detect any hand 1.0.E.1.1 System does not draw anything 1.0.E.2 System does not detect more than one fingertip 1.0.E.2.1 System continues to detect single fingertip until more fingertips are detected.
Includes:	-
Priority:	High
Frequency of Use:	High

Use Case ID:	6
Use Case Name:	Toggle Dynamic Touch

Actors:	User
Description:	User toggles on/off dynamic touch mode
Trigger:	User presses on ‘Dynamic Touch’ button
Preconditions:	-
Postconditions:	Dynamic Touch detection is on
Normal Flow:	1.1 User presses on ‘Dynamic Touch’ button 1.2 System executes postconditions
Alternative Flows:	-
Exceptions:	-
Includes:	-
Priority:	High
Frequency of Use:	High

Use Case ID:	7
Use Case Name:	Toggle Piano Layout

Actors:	User
Description:	User toggles between Piano layout 1 and 2
Trigger:	User presses on ‘Layout’ button
Preconditions:	-
Postconditions:	System keeps track of layout to be used
Normal Flow:	1.1 User presses on ‘Layout’ button 1.2 System executes postconditions
Alternative Flows:	-
Exceptions:	-
Includes:	-
Priority:	High
Frequency of Use:	High

Use Case ID:	8
Use Case Name:	Toggle Camera

Actors:	User
Description:	User toggles between front-facing and rear-facing camera
Trigger:	User presses on Camera button
Preconditions:	-

Postconditions:	System displays live feed from the next camera
Normal Flow:	1.1 User presses on ‘Camera’ button 1.2 System executes postconditions
Alternative Flows:	-
Exceptions:	1.0.E.1 Phone only has one camera 1.0.E.1.1 System reloads but uses back the only camera available
Includes:	-
Priority:	Low
Frequency of Use:	Low

Use Case ID:	9
Use Case Name:	Change Camera Size

Actors:	User
Description:	User changes camera resolution
Trigger:	User presses on ‘Size’ button
Preconditions:	Phone allows different camera resolutions
Postconditions:	System switches camera resolution
Normal Flow:	1.1 User presses on Size button 1.2 System displays available camera sizes 1.3 User presses on desired camera size 1.4 System reloads and displayed capture frames in the desired resolution

Alternative Flows:	-
Exceptions:	-
Includes:	-
Priority:	Low
Frequency of Use:	Low

Use Case ID:	10
Use Case Name:	Press Note

Actors:	User
Description:	User plays the piano
Trigger:	User presses a piano key
Preconditions:	Piano keys has been set Finger detection mode is on User executes a ‘touch’ motion
Postconditions:	System responds by playing the corresponding sound of the key pressed.
Normal Flow:	1.1 User presses on a piano key 1.2 System detects a touch 1.3 System processes touch and plays the corresponding sound
Alternative Flows:	-
Exceptions:	1.0.E.1 User executed a touch motion outside all piano key regions 1.0.E.1.1 System does nothing

Includes:	-
Priority:	High
Frequency of Use:	High