POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

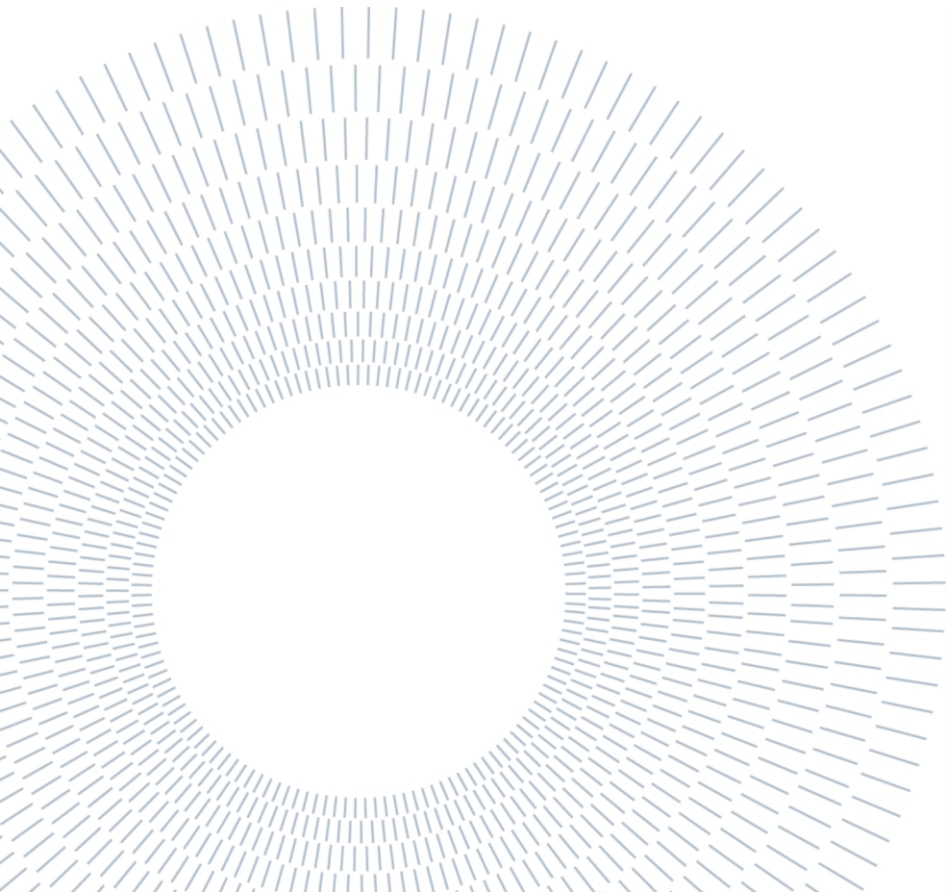
Pressed Piano Key Detection and Transcription by Visual Motion Analysis

TESI DI LAUREA MAGISTRALE IN
COMPUTER SCIENCE AND ENGINEERING-
INGEGNERIA INFORMATICA

Author: **Ali Özkaya**
Sidem Işıl Tuncer

Student ID: 10722495
10713895
Advisor: Vincenzo Caglioti

Academic Year: 2020-21



Acknowledgement

First, we would like to thank our supervisor Professor Vincenzo Caglioti for giving the opportunity to write this thesis and for supporting us. We also would like to thank our teacher in our BSc years Professor Yücel Yemez who we always looked up to and who led us into this field.

Finally, we would like to express our gratitude to our parents for always encouraging us to do our best.

Abstract

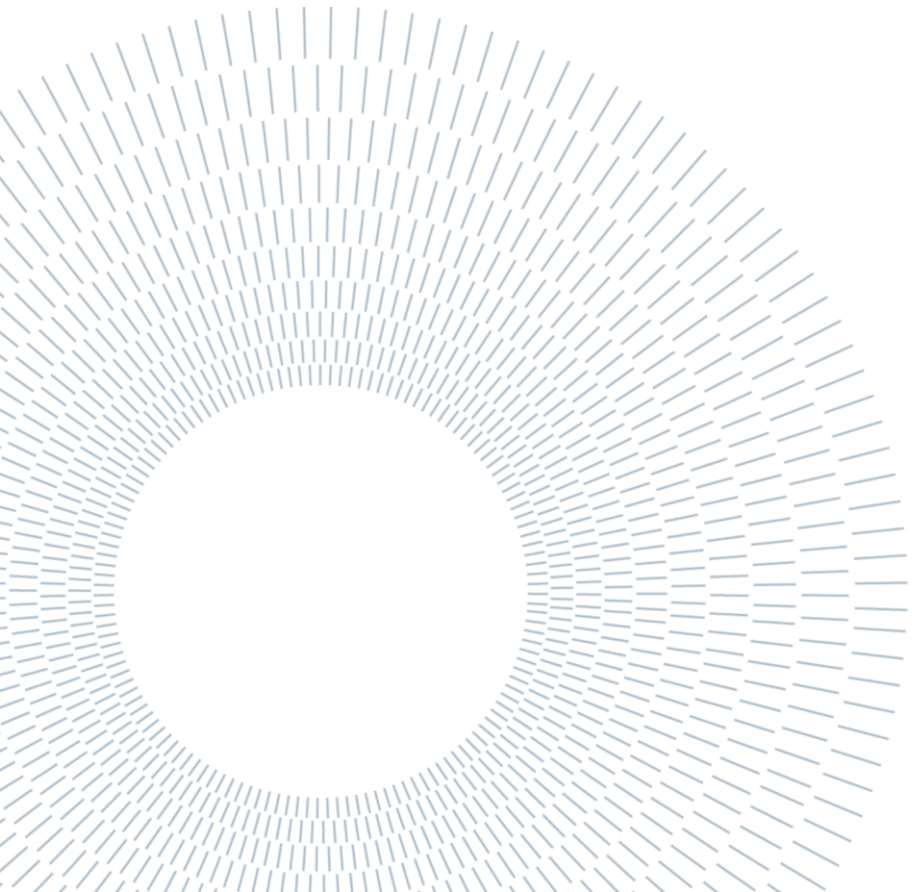
The aim of this thesis is to detect the pressed piano keys from two video sources by analyzing the piano player's fingers. The video sources are obtained from two cameras placed above and in front of the piano. Key aspect of this thesis is to figure out relevant visual features which can be used to determine key presses and devise an algorithm which utilizes these features. First, the piano player's hand was segmented from the background using color filters, leaving only the area of the hand in the footage. On the segmented image the areas of the hand were used to find the convex hull of the hands and convexity defects were used to detect fingertips along with a local extrema-based method. Since there were 2 different footages of the same scene it was required to match the corresponding calculated fingertips. To solve this problem, keys on the piano were detected and labeled with different methods for both videos. After finding which fingertip was on which key, fingertips from the two sources were matched accordingly. Comparing the calculated world coordinates based on the known setup between the two cameras helped in detecting key presses. Because of the angled recording of the front camera, whenever a fingertip presses a white key the depth of that fingertip was higher than the upper camera's calculation, which corresponds to the real-world depth. It was seen that when the player presses a key, this gap increases significantly. Therefore, a threshold was put on this gap to detect the pressed white keys. The black key press detection was handled utilizing the shadows they cast on the upper part of the piano while remaining dormant. This shadow is not visible when the key is pressed, so a dormant state mask was applied on each instance of the upper camera to reveal the black keys that are being pressed at each moment. In the end, to display what the algorithm deduces from this process in sound format a MIDI file is generated. Timings of the keys being detected as pressed and as removed are used in this process. Polyphonic sounds are handled by calculating the elapsed time between the current key press and the last key press or removal.

Key-words: computer vision, hand segmentation, fingertip detection, piano key, image filtering, convexity defect.

Sommario

Lo scopo di questa tesi è quello di rilevare i tasti premuti del pianoforte da due sorgenti video analizzando le dita del pianista. Le sorgenti video sono ricavate da due telecamere poste sopra e davanti al pianoforte. L'aspetto chiave è capire le caratteristiche visive rilevanti che possono essere utilizzate per determinare le presse chiave e ideare un algoritmo che utilizza queste caratteristiche. In primo luogo, la mano del pianista è stata segmentata dallo sfondo utilizzando filtri di colore, lasciando solo l'area della mano nel filmato. Sull'immagine segmentata le aree della mano sono state utilizzate per trovare lo scafo convesso delle mani e difetti di convessità sono stati utilizzati per rilevare i polpastrelli con un metodo basato su estreme locali. Dal momento che c'erano due diverse scene della stessa scena è stato necessario abbinare le dita calcolate corrispondenti. Per risolvere questo problema, i tasti sul pianoforte sono stati rilevati ed etichettati con metodi diversi per entrambi i video. Dopo aver trovato quale punta era su quale chiave, i polpastrelli delle due fonti sono stati abbinati di conseguenza. Confrontare le coordinate mondiali calcolate in base alla configurazione nota tra le due telecamere ha aiutato a rilevare le presse chiave. A causa della registrazione angolata della fotocamera anteriore, ogni volta che un dito preme un tasto bianco la profondità di quel dito era superiore al calcolo della fotocamera superiore, che corrisponde alla profondità del mondo reale. Si è visto che quando il giocatore preme un tasto, questo divario aumenta in modo significativo. Pertanto, è stata determinata una soglia su questo spazio per rilevare i tasti bianchi premuti. Il rilevamento della pressione dei tasti neri è stato gestito utilizzando le ombre che proiettavano sulla parte superiore del pianoforte pur rimanendo dormienti. Questa ombra non è visibile quando il tasto viene premuto, quindi una maschera di stato dormiente è stata applicata su ogni istanza della fotocamera superiore per rivelare i tasti neri che vengono premuti in ogni momento. Alla fine, per visualizzare ciò che l'algoritmo deduce da questo processo in formato audio un file MIDI viene generato. Temporizzazioni dei tasti rilevati come premuti e rimossi vengono utilizzati in questo processo. I suoni polifonici sono gestiti calcolando il tempo trascorso tra la pressa di tasti corrente e l'ultima pressa o rimozione di tasti.

Parole chiave: visione artificiale, segmentazione manuale, rilevamento della punta delle dita, pianoforte, filtraggio delle immagini, difetto di convessità.



Contents

Introduction

Musical Information Retrieving (MIR) has become an important research field with many real-world applications and one of its challenging tasks is to determine the notes played from an instrument. Especially the ones that are polyphonic such as piano is harder to analyze with multiple independent sounds at the same time. Audio based methods are mostly inaccurate, they suffer from this property and work better with monophonic instruments [5,9]. Musical instruments that can record the played notes is a reliable method, however it is expensive and not easily accessible [5]. Therefore, in this thesis a computer vision-based method is proposed reviewing many of today's significant tasks such as background subtraction, fingertip detection and 3D reconstruction. The proposed method can pave the way to wider access to online piano tutoring and would allow piano players to review their performance without relying on expensive equipment or private tutors.

Understanding the motion of a piano player's hand and detecting which keys are pressed throughout a piano piece from visual data are the main goals of this thesis. The proposed solution utilizes two cameras which are placed around the piano, one is placed vertically above the piano keys and the second one is placed on top of the piano, with and angled view towards the piano keys and the piano player. There are many difficulties in the process of inferring piano key presses from the two camera sources, such as complex and dynamic backgrounds, changes in illumination, and distortions in the videos. Finding various methods that will address these issues is our aim.

In this paper we propose a pipeline with a prior preprocessing phase, which runs an algorithm loop on each frame of the two video sources to segment the piano player's hand, calculate fingertips from the segmentation, calculate the world coordinates of the fingertips utilizing the known setup, and determine if a piano key is being pressed depending on the contradictory information coming from the two video sources. The output is a video from the upper camera view which displays the piano player's fingertips and the piano keys being pressed. The interpretation of the result is left to the user because the field of music does not offer clear goals or criteria to evaluate performance [13].

[1]The proposed solution should work on any similar setup after a calibration process of parameters depending on the camera setup, color distribution of the background, lighting in the room and the skin tone of the player. The success of the results depends on how well the hand segmented from the background and how accurate the fingertips are detected. If these steps are successful, pressed keys are detected with a high accuracy.

The structure of the paper is as follows:

- In Chapter 1 we review Computer Vision theory which is relevant to our study, including 2D geometry, projective geometry, image filtering, model fitting, and color spaces.
- In Chapter 2 we introduce the design details of our setup, the preprocessing phase, and our pipeline's steps, which include the segmentation of the hand, detection and matching of the fingertips and finally calculation of the world coordinates alongside inferring if a piano key is being pressed.
- In Chapter 3 we analyze the performance of our algorithm on its various stages of development with metrics such as precision, recall and accuracy. We also mention our reasoning behind the changes in the algorithm and their correlation with the improvements in the results.

[1] Access to code developed for this thesis https://github.com/aozkaya/Pressed-Piano-Key-Detection/

1. Chapter one

Theoretical Review

1.1 2D Geometry

In this section some of the basic geometry rules will be explained in the context of this thesis.

Distance between 2 points: Given P(x_1,y_1) and Q(x_2,y_2), the distance between two points is calculated by:

$$Distance(P,Q) = \sqrt{(x2 - x1)^2 + (y2 - y1)^2} \tag{1.1}$$

Distance between a point and a line: Given P(x1,y1) and Q(x2,y2) defines the line L and the distance of point X(x0,y0) to that line is defined by:

$$Distance(L,Q) = \frac{|(x2 - x1)(y1 - y0) - (x1 - x0)(y2 - y1)|}{\sqrt{(x2 - x1)^2 + (y2 - y1)^2}} \tag{1.2}$$

Intersection of 2 lines: Given **2** indefinitely long lines defined by points P1(x1,y1), P2(x2,y2) and Q1(x3,y3), Q2(x4,y4) the intersection point can calculated by the determinants:

$$x = \frac{\begin{vmatrix}\begin{vmatrix}x1 & y1\\ x2 & y2\end{vmatrix} & \begin{vmatrix}x1 & 1\\ x2 & 1\end{vmatrix}\\ \begin{vmatrix}x3 & y3\\ x4 & y4\end{vmatrix} & \begin{vmatrix}x3 & 1\\ x4 & 1\end{vmatrix}\end{vmatrix}}{\begin{vmatrix}\begin{vmatrix}x1 & 1\\ x2 & 1\end{vmatrix} & \begin{vmatrix}y1 & 1\\ y2 & 1\end{vmatrix}\\ \begin{vmatrix}x3 & 1\\ x4 & 1\end{vmatrix} & \begin{vmatrix}y3 & 1\\ y4 & 1\end{vmatrix}\end{vmatrix}} \quad y = \frac{\begin{vmatrix}\begin{vmatrix}x1 & y1\\ x2 & y2\end{vmatrix} & \begin{vmatrix}y1 & 1\\ y2 & 1\end{vmatrix}\\ \begin{vmatrix}x3 & y3\\ x4 & y4\end{vmatrix} & \begin{vmatrix}y3 & 1\\ y4 & 1\end{vmatrix}\end{vmatrix}}{\begin{vmatrix}\begin{vmatrix}x1 & 1\\ x2 & 1\end{vmatrix} & \begin{vmatrix}y1 & 1\\ y2 & 1\end{vmatrix}\\ \begin{vmatrix}x3 & 1\\ x4 & 1\end{vmatrix} & \begin{vmatrix}y3 & 1\\ y4 & 1\end{vmatrix}\end{vmatrix}} \tag{1.3}$$

Slope of a line: Given a line defined by points P(x_1,y_1) and Q(x_2,y_2) the slope is calculated by:

$$m = \frac{y2 - y2}{x2 - x1} \tag{1.4}$$

Angle between 2 intersecting lines: Given **2** lines defined by points P1(x_1,y_1), P2(x_2,y_2) and P3(x_3,y_3) that intersect at P2, the angle between them can be calculated by :

$$dot = (x1 - x2)(x3 - x2) + (y1 - y2)(y3 - y2) \tag{1.5a}$$

$$angle = arccos(\, dot/\big(Distance(P1, P2) * Distance(P2, P3)\big) \tag{1.5b}$$

1.2 Projective Geometry

Creation of images can be described as a translation from a 3-dimensional to a 2-dimentional projective space and projective geometry explains this translation in computer vision. Some of the geometric properties may not be preserved after this process such as angles, parallelism, distances, and ratios of distances etc. Projective geometry is also an important tool in computer vision for example used for camera calibration, scene reconstruction and object recognition.

1.2.1 Camera and perspective projection

Pinhole is the simplest imaging device that works on the principles of perspective projection geometry in terms of its mapping from 3D to 2D. While inspecting the perspective projection equations some assumptions will be made for the sake of simplicity. Origin of the world coincides with the center of the projection and the world z-axis is aligned with the camera axis.

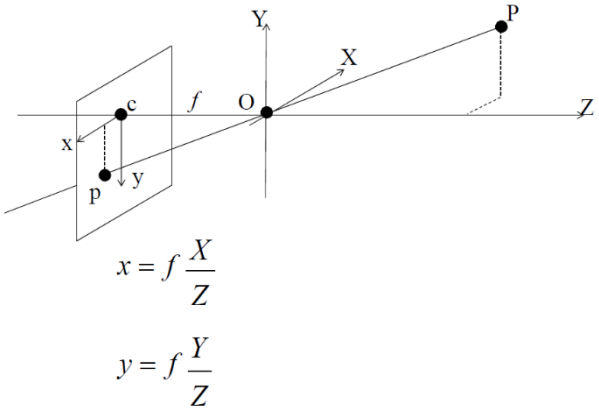


Figure 1: Perspective projection [1]

In Figure 1, center of the projection is shown with *O* and the plane on the left is the image plane. The distance between the inverted image plane and the projection center, which is denoted with *f*, is the focal length. The resulting 2D image becomes the intersection of the image plane and the straight lines between the camera center *O* and the points in 3D world. Following the similar triangles rules the equations in Figure 1 are obtained in perspective projection. It is important to notice that in projective projection any point on the line *OP* is mapped onto the same point on the image plane. When a plane on the world is parallel to the image plane, perspective projection only scales that surface. However, if the surface is not parallel to the image plane, then projective distortion occurs.

We can express the formula above in matrix form using homogeneous coordinates as,

$$\lambda \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} f & \cdots & \cdots & 0 \\ \vdots & f & \cdots & 0 \\ & \cdots & 1 & 0 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \tag{1.6}$$

where λ (**depth**) is equal to Z.

1.2.2 The Camera matrix

From the terms in equation 1.6 following notations are introduced

$$K = \begin{bmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & 1 \end{bmatrix}, \; x = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}, \; X = \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \tag{1.7}$$

and we derive

$$x = K[I_{3x3}|\mathbf{0}_{3x1}]X = PX \tag{1.8}$$

P is called **camera matrix** and it relates the image coordinates to object coordinates in the world.

In a general camera model, the matrix K in Equation (1.7) is actually the following matrix:

$$K = \begin{bmatrix} f_x & s & x_0 \\ 0 & f_y & y_0 \\ 0 & 0 & 1 \end{bmatrix} \tag{1.9}$$

Where *f* is the **focal length** same as before,

s is the **skew,**

and x_0, y_0 is the principle point which is the view of the focal point on the image.

Same focal length means that the number of pixels per unit distance in image coordinates in x and y directions are equal. Also, as f decreases the field of view of the camera becomes larger.

These parameters are called the intrinsic parameters of the camera and a camera is calibrated if K is known [4].

1.2.3 Projective Spaces

Some important terms will be introduced under this section and will be helpful to understand a very important step of this thesis.

The projective plane P^2 is composed of the union points of R^2 and ideal points. These **ideal points** form a line so called the **line at infinity** l_∞ in P^2. As explained in the previous sections if a plane in the world is not parallel to the image plane, the parallel lines on that plane are not projected as parallel to the image. Therefore, the previous concept line at infinity in other words **vanishing line** comes into view. Lines on the image plane that were parallel in the world intersect at a point called the **vanishing point.** Finally, combining all the vanishing points resulting from the parallel lines in the image form the vanishing line.

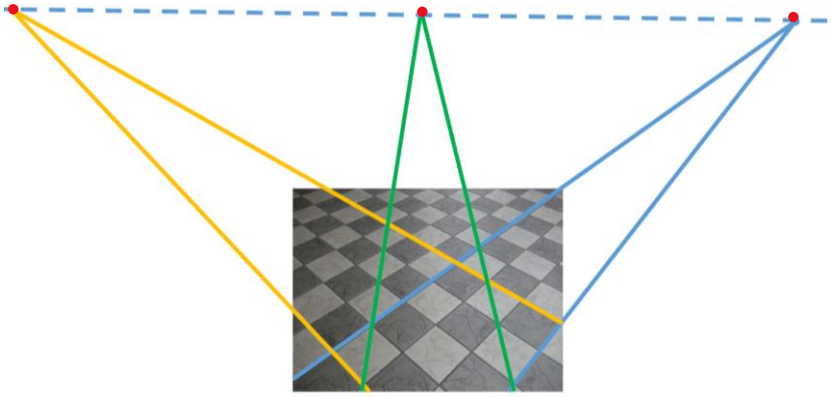


Figure 2: Vanishing points and Vanishing line [7]

In the figure the lines that are parallel in the real world intersect at red points which are vanishing points and the line that connects them is the vanishing line. They are not necessarily visible on the image.

1.2.4 Planar projective transformations

Planar projective transformations are the most general transformation in P^2. The mapping from P^2 to itself in this transformation is invertible. Therefore, the

collinearity is preserved which means that three points on the same line are still on the same line after the transformation. A mapping h can be shown as

$$h(\boldsymbol{x}) = H\boldsymbol{x} \tag{1.10}$$

where H is in the form of:

$$H = \begin{bmatrix} h_{1,1} & h_{1,2} & h_{1,3} \\ h_{2,1} & h_{2,2} & h_{2,3} \\ h_{3,1} & h_{3,2} & h_{3,3} \end{bmatrix} \tag{1.11}$$

This transformation is the composition of linear transformation and translation. It is important to notice that projectivities preserves the cross ratio of lengths but not the ratio of lengths as in affinities which will be explained shortly. Line at the infinity may not remain at the infinity.

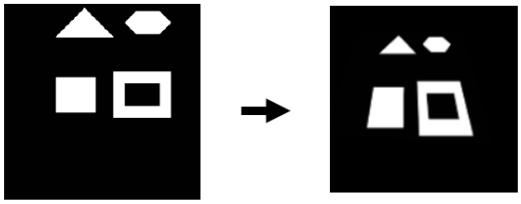


Figure 3: Projective transformation [7]

Depending on the structure of H there are other kinds of transformations and the geometric preservations changes for each of them.

Isometries: This transformation can be described as a 2D motion of a shape. Preserves angles, areas and distances.

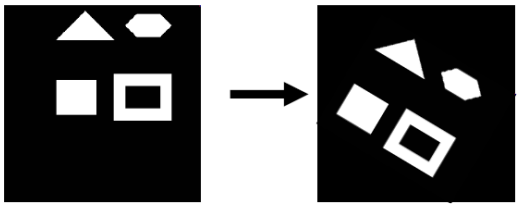


Figure 4: Isometric transformation [7]

Similarities: This transformation can be described as a 2D motion of a shape with an isotropic scaling. Preserves angles, ratio between lengths and shapes but not the areas and lengths.

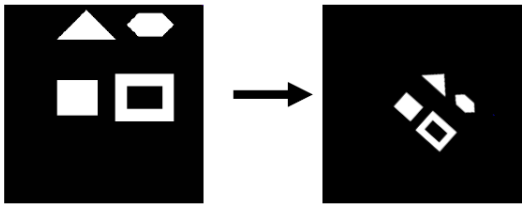


Figure 5: Similarity transformation [7]

Affinities: This transformation can be described as a translation after a rotation and non-isotropic scaling. Preserves parallelism, ratio of lengths on parallel lines, ratio of areas but not length ratios and angles. Line at the infinity remains in the infinity but can be mapped to a different point.

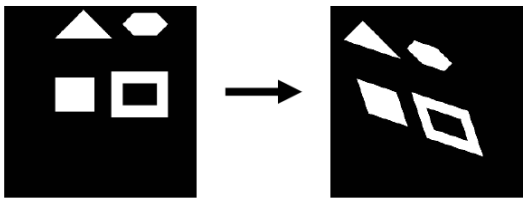


Figure 6: Affine transformation [7]

1.2.5 Cross Ratio

As told in the previous section cross ratio is invariant in projective transformations. Given for colinear points p1,p2,p3,p4 and their abscissas x1,x2,x3,x4 as in Figure 7, the cross ratio of x1,x2,x3 and x4 is given by the right side of the equation

$$CR(p1,p2,p3,p4) = \frac{\frac{x_1 - x_3}{x_1 - x_4}}{\frac{x_2 - x_3}{x_2 - x_4}} \tag{1.12}$$

and the cross ratios of both set of points are equivalent because of invariance property.

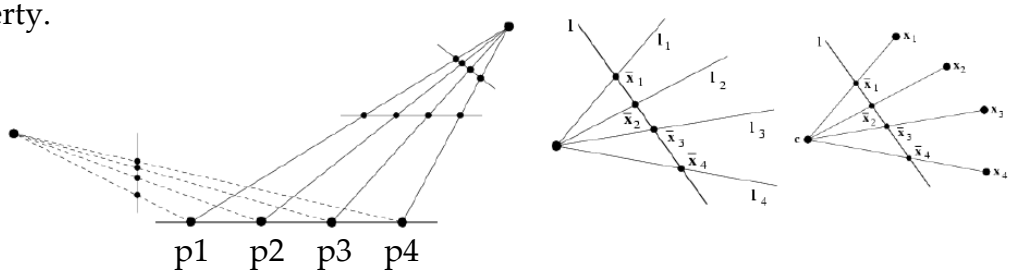


Figure 7: Cross ratio invariance [1]

1.3 Image Filtering

An image can be thought of as a function f that takes two variables pixel position x and y. This function $f(x,y)$ gives the intensity of that position and is defined over a rectangle with the outcome range of [0-255] for a greyscale image. For color images the outcome becomes the combination of Red, Green and Blue.

$$f(x,y) = \begin{bmatrix} r(x,y) \\ g(x,y) \\ b(x,y) \end{bmatrix} \tag{1.13}$$

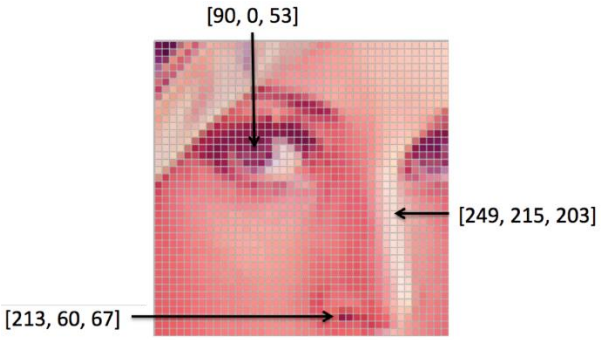


Figure 8: RGB Image [11]

As seen in the above figure the little rectangles can be seen as the elements of a matrix and the whole matrix forms the image.

When we apply a filter to an image, pixel properties changes but not their positions. In this way we can obtain important information about an image and extract useful features such as corners and edges. Masks are used for image filtering, and they are also matrices applied to a neighborhood of pixels. The elements in the mask matrix and the image matrix are multiplied in the operated area and summed to be assign to the middle pixel. An example operation can be seen in the following figure.

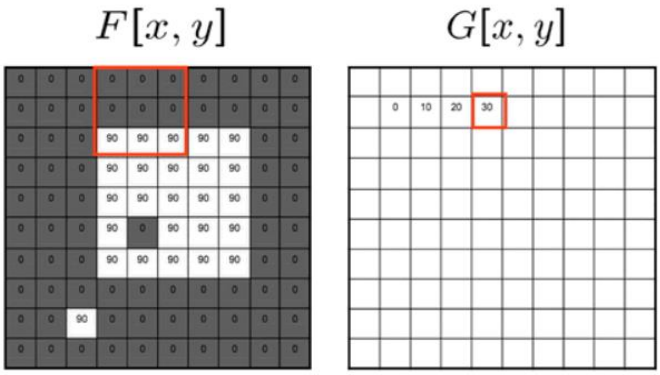


Figure 9: Image filtering [11]

Image filtering can be used for image segmentation which partitions the image according to similar features in pixels. In this way we can identify objects such as hands which is an important aspect in this thesis. For hand segmentation the shared feature can be the color or intensity and can be implemented based on thresholding. In Figure 10 the pixel intensities greater than 100 are assigned as intensity 255 (white) and others are assigned as intensity 0 (black).

Figure 10: Image segmentation [11]

1.3.1 2D Convolution

Filters can be expressed using convolution and used for operations like edge detection. 2 Matrices are used as before; a mask called the kernel is applied to the input image, and the output is another image. We apply the kernel on the input image by sliding over the image.

$$f = \boxed{10}\boxed{50}\boxed{60}\boxed{10}\boxed{20}\boxed{40}\boxed{30}$$

$$g = \boxed{1/2}\boxed{1/2}\boxed{1/2}$$

Let's suppose f is the image and g is the kernel. If we want to find the output value on the 4th pixel of f we should arrange the filter as the following.

10	50	60	10	20	40	30
0	0	1/2	1/2	1/2	0	0

After we multiply the overlapping elements and put the sum to the output result would be 45 in the 4th element. If we do this to every pixel, the final image would be the following:

$$h = \boxed{30}\boxed{60}\boxed{60}\boxed{45}\boxed{30}\boxed{45}\boxed{35}$$

1.3.2 Edge Detection

Again, we will think about the images as functions and see how we can take derivative on a discrete function like images. This derivative will be an approximation since we are not working on a continues function.

$$\frac{\partial f}{\partial x} = \lim_{\epsilon \to 0} \left(\frac{f(x+\epsilon) - f(x)}{\epsilon} \right) \tag{1.14}$$

The above equation is the definition of derivative, and it can be expressed as a convolution since it is linear and shift invariant. The approximation can be made with the below formula which is a convolution with the kernel $\boxed{1}\boxed{-1}$. This kernel is vertical, and it can also be applied to images as horizontal.

$$\frac{\partial f}{\partial x} \approx \lim_{\epsilon \to 0} \left(\frac{f(x_{n+1}) - f(x_n)}{\Delta x} \right) \tag{1.15}$$

Using this derivation, we can detect the edges along a row or column. It helps to see rapid changes in the intensity of images pixels as seen in Figure 11.

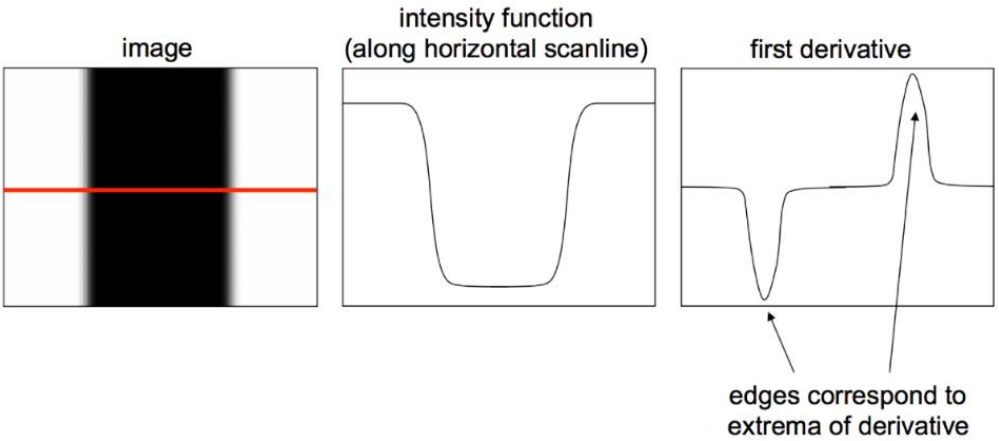


Figure 11: Edge detection [11]

Canny Edge Detector: This is a commonly used edge detector in the field. The following steps are applied to the images:

1. Smoothing with Gaussian filter because without smoothing, the derivation results in so much noise and the edges cannot be identified. Taking the x and y derivatives afterwards.
2. Finding the gradients of each pixel with orientation and magnitude information.
3. Applying Non-Maximum Suppression to find the maximum point along a curve. Since the gradient is orthogonal to the edge if we follow its orthogonal, we follow the edge direction. We find the local maxima and extract 1D segment for non-maximum suppression.
4. Hysteresis thresholding to link the edges along the correct curves.

1.3.3 Dilation and Erosion

So far, we have seen linear filters which basically multiplies and sums the matrix elements. There are also non-linear filters such as median filter, ordered statistics-based filters and threshold filters. Erosion and dilation are non-linear filters applied to binary images. They are important for defining boundaries of objects and segmentation.

Erosion: This operation reduces the boundaries of the objects when applied to binary images. It replaces the pixels with the minimum of its neighborhood in a certain area. Its mask is also a binary image.

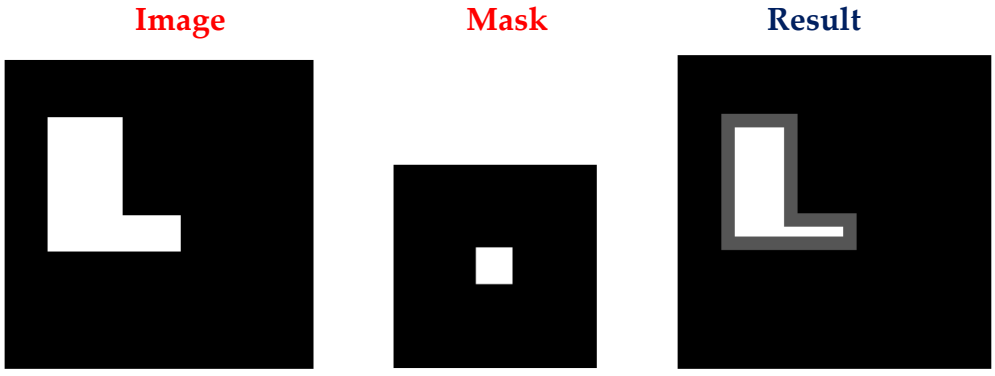


Figure 12: Erosion [7]

In Figure 12 it is observed that the gray area is removed after applying erosion with the given binary mask.

Dilation: This operation enlarges the boundaries of the objects in a binary image. It replaces the colors of pixels with the maximum of its neighborhood in a certain area.

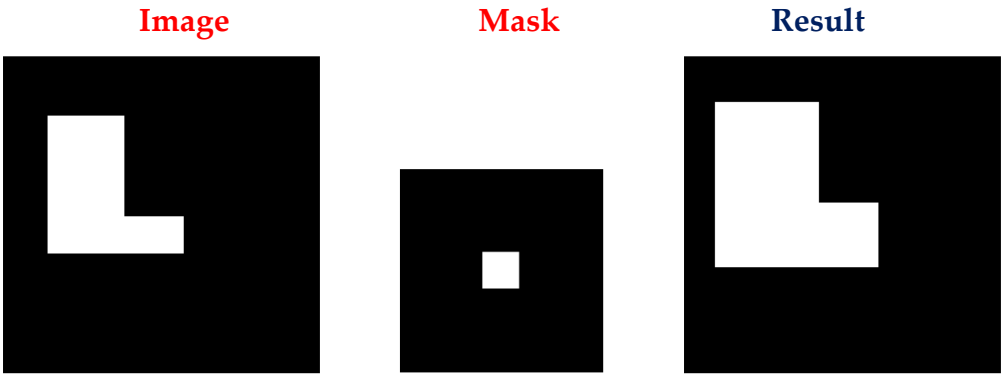


Figure 13: Dilation [7]

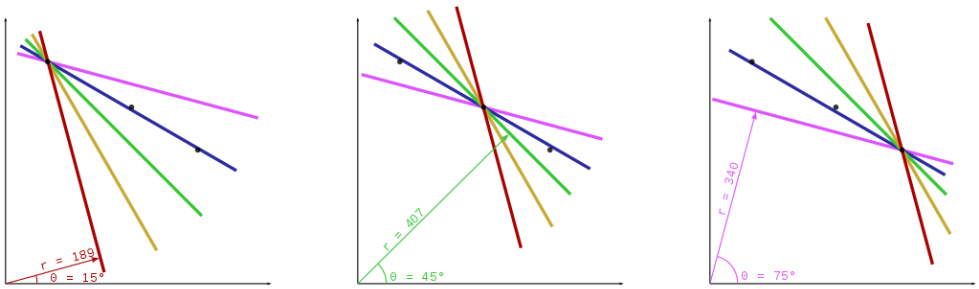
Figure 13 shows the boundaries of the original shape is enlarged after applying dilation with the given binary mask.

Using these masks one after the other has different effects on the image. If we apply dilation first, the object contour is smoothed, and it may create some bridges. If we apply erosion first, again it smooths the object contour but may create some gaps. It is useful for eliminating protrusions on the objects.

1.4 Model Fitting

Line fitting is an important operation in this thesis so it will be explained in this section. Here we are talking about the line as a model and its association with a set of features is called fitting. This is a difficult process in terms of finding the right model in multiple models and the noise. Therefore, a **voting** system is applied instead of checking all the feature combinations by fitting a model to each of them. In voting features vote for the models they fit. Noisy features' votes will not be very important since they won't be consistent with the majority. A commonly used voting technique for line fitting is called **Hough Transform.** The following steps are used in this algorithm:

1. Record the votes for each possible line that the edge points lie.
2. Chose the lines that get the votes above a threshold.



θ	r
15	189.0
30	282.0
45	355.7
60	407.3
75	429.4

θ	r
15	318.5
30	376.8
45	407.3
60	409.8
75	385.3

θ	r
15	419.0
30	443.6
45	438.4
60	402.9
75	340.1

Figure 14: Hough transform example [3]

To explain it on an example the above figure will be used. According to algorithm's parameters several lines are drawn on each data point which they are on. The arrows are the support lines of each drawn line perpendicular to it and passes the origin expressed with 2 parameters rho and theta. We choose to represent lines in this form which is polar representation because usual $y = mx + b$ representation has problems like infinite values and vertical line representation. Now we express the lines with the following form

$$\rho = \mathrm{xcos}\,\theta + \mathrm{ysin}\,\theta \tag{1.16}$$

where ρ is the length and θ is the angle to the x axis as shown in Figure 14. Therefore, in **Hough Space** an edge point represented with 1.17 generates a cosine curve.

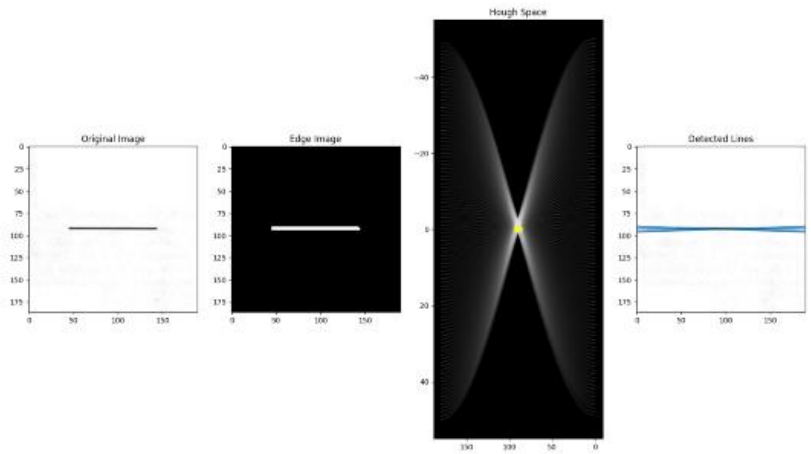


Figure 15: Line detection in an image [13]

In the figure the yellow circle is the representation of the detected line in Hough Space. Intersection of cosine curves of points means that the points lie on that line represented with (ρ,θ) pair. Hough transform algorithm finds the intersections larger than a chosen threshold and returns them as lines.

1.5 Color Spaces

A color space is a domain which is used to express colors. To encode a color in a color space, a color model is used which is a mathematical representation of a color using tuples of numbers. In association with a color space, these tuples are used to produce the color in a digital environment. Typically consisting of three or four tuples, these scalars are stored in a matrix to represent the color of each pixel of an image. The color space that a color model is associated to determines the expressiveness of the model, and its ability to preserve colors in case of a conversion from another color format.

1.5.1 RGB Color Space

The most common color space is the RGB color space, which represents a color as a combination of the primary colors red, green, and blue. These three colors are chosen because by themselves they cover most of the color space that the human eye can perceive. In RGB color model there are three scalars to represent the three primary colors. The values of the scalars have a range of [0-255], higher values describing higher intensity. RGB is an additive color model, which means its three components are added to produce the resulting color. RGB is the default color space used by OpenCV when processing videos, but it is not ideal for hand segmentation because the clustering of skin color is much wider than other color spaces.

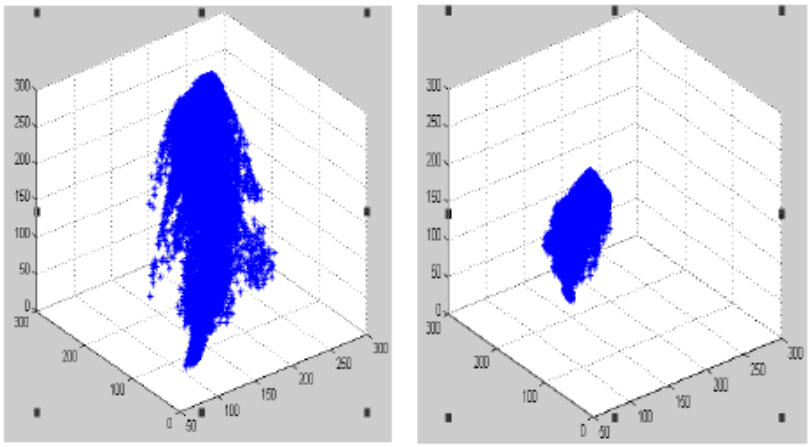


Figure 16: White skin color clustering for RGB (left) and YCbCr (right) [2]

Figure 16 shows a comparison of the RGB space with the YCbCr space on the clustering of skin color, a more compact clustering is preferred because it allows the implementation of a tighter filter to eliminate the background better [2]. In the methodology part the RGB color space is referred as BGR color space because the ordering of the tuples in OpenCV implementation.

1.5.2 YCbCr Color Space

The YCbCr color space explicitly uses luma as Y in its color model which is beneficial for hand segmentation [10]. Cb and Cr values are obtained by subtracting the blue and red colors respectively from the luma, the OpenCV implementation of conversion from RGB color space to YCbCr color space is as follows [8],

$$Y = 0.299R + 0.587\,G + 0.114\,B \tag{1.17a}$$

$$Cb = (R - Y)\,0.713 + 128 \tag{1.17b}$$

$$Cr = (B - Y)\,0.564 + 128 \tag{1.17c}$$

This color space offers a more compact clustering as mentioned, and it has a smaller overlap between the skin colors and non-skin colors [2]. YCbCr color space is an absolute color space, unlike RGB, which means the difference between colors is directly related with their model representation's vector distance. These facts alongside the simple transformation from the default RGB color space makes YCbCr space ideal for hand segmentation. In the methodology chapter the YCbCr color space is referred as YCrCb color space because of the ordering of the tuples in OpenCV implementation.

1.5.3 HSL Color Model

HSL, standing for hue, saturation and lightness is another color model for RGB color space. Hue shows the dominant color of the pixel. Hue values 0^{o}, 120^{o} and 240^{o} represent pure primary colors red, green, and blue respectively and the values in between correspond to the colors in between. Saturation gives the concentration of gray inside the color; it is a value ranging from 0 to 1 with a shade of gray at 0 and the pure color at 1. Lightness shows the black and white balance of the color on a scale from 0 to 1, corresponding to full black and full white respectively at the extremes. HSL values are refit to fit between 0 and 255 scale for 8-bit images, the conversion method of OpenCV from RGB to HSL is as follows [8],

$$V_{max} = max(R, G, B) \tag{1.18a}$$

$$V_{min} = min(R, G, B) \tag{1.18b}$$

$$H = \frac{1}{2}\begin{cases} 60\frac{(G-B)}{V_{max}-V_{min}}\, if\ V_{max} = R \\ 120 + 60\frac{(B-R)}{V_{max}-V_{min}}\, if\ V_{max} = G \\ 240 + 60\frac{(R-G)}{V_{max}-V_{min}}\, if\ V_{max} = B \\ 0\ if\ R = G = B \end{cases} \tag{1.18c}$$

$$S = 255\begin{cases} \frac{V_{max}-V_{min}}{V_{max}+V_{min}}\ if\ L < 0.5 \\ \frac{V_{max}-V_{min}}{2-(V_{max}-V_{min})}\ if\ L \geq 0.5 \end{cases} \tag{1.18d}$$

$$L = 255\ \frac{V_{max}+V_{min}}{2} \tag{1.18e}$$

In the methodology chapter the HSL color space is referred as HLS color space because of the ordering of the tuples in OpenCV implementation.

2. Chapter two

Methodology

2.1 Design Details

Two cameras are used: one placed on top of the piano keys looking downwards, and other in front of the piano keys looking at the direction of the keys with an angle. Their lenses were in the same alignment and video quality was 60 Hz 1080p for both videos.

Figure 17: Camera Setup

Figure 18: Camera setup with lengths

Cameras were fixated using 2 tripods and a box to give the angle to the front camera. They were stationary during the whole recording and measurements were carried out as precise as possible as shown in the Figure 2. Upper camera's image plane is parallel to the white key surface, front camera is angled with 26.2 degrees downwards. Upper camera's measurements will be used in the further sections in order to find the intrinsic parameter f and 3D world coordinates of the fingertips according to perspective projection. Front camera's measurements will not be used for finding 3D world coordinates due to distortion, instead cross ratio invariance will be utilized for an approximation.

2.2 Preprocessing

Preprocessing step is done on the initial frames of both videos to calculate relevant information about the piano keys. Left, right and bottom edges of the white and black piano keys are calculated, which are later used in fingertip indexing and matching. The preprocessing step requires the piano player's hands to be away from the piano keys on both cameras to get an unobscured image of the keys from the first

frame. A separate photograph of the piano keys is used in similar works for preprocessing [14], but most cameras have different image properties for videos and photographs, like resolution, field of view and image filters. These differences along with the possibility of the photograph and the video not being aligned was our incentive to pursue a different method.

2.2.1 Key Segmentation

The segmentation process starts with a YCrCb color filter for the upper camera and YCrCb and HLS filter for the front camera. We experimented with different color spaces and decided that YCrCb is best suited for this task, a second filter was applied to the front camera because it has a wider view which includes several background objects. Locations of the piano keys are important because they are on the same plane, so they offer information about the transformation between the two cameras. Calibrated values for the keys and the result are as follows,

```
//top YCrCb key filter  //front YCrCb key filter  //front HLS key filter
int K_Y_MIN = 100;      int K_Y2_MIN = 0;         int K_H_MIN = 0;
int K_Y_MAX = 255;      int K_Y2_MAX = 255;       int K_H_MAX = 255;
int K_Cr_MIN = 0;       int K_Cr2_MIN = 0;        int K_L_MIN = 145;
int K_Cr_MAX = 145;     int K_Cr2_MAX = 255;      int K_L_MAX = 255;
int K_Cb_MIN = 0;       int K_Cb2_MIN = 117;      int K_S_MIN = 0;
int K_Cb_MAX = 255;     int K_Cb2_MAX = 130;      int K_S_MAX = 255;
```

Canny edge detection and then Hough Line transform are applied on the filter results shown in Figures 19 and 20 to detect the horizontal bottom edge lines of the white and black keys. Alternatively, Sobel edge detector could be used for upper camera instead of Canny, but Canny is more commonly used in literature [15]. It was not feasible for the front camera to use Sobel edge detector because the edges are not vertical nor horizontal. Applying Probabilistic Hough Line transform to edges gave better results for detecting vertical lines because of the possibilities of parameters like minimum line length and maximum line gap. It provides more accurate edge detection after a precise tuning. Vertical lines in front camera are used for vanishing point calculation while regular Hough transform is used to determine the horizontal lines which represent the white and black key's bottom edges.

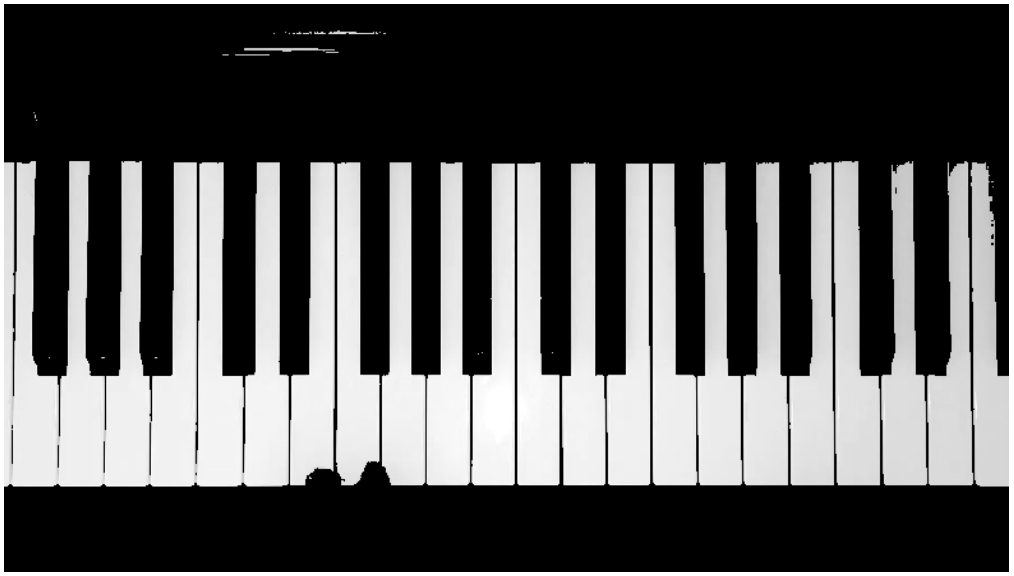


Figure 19: Upper camera filter results

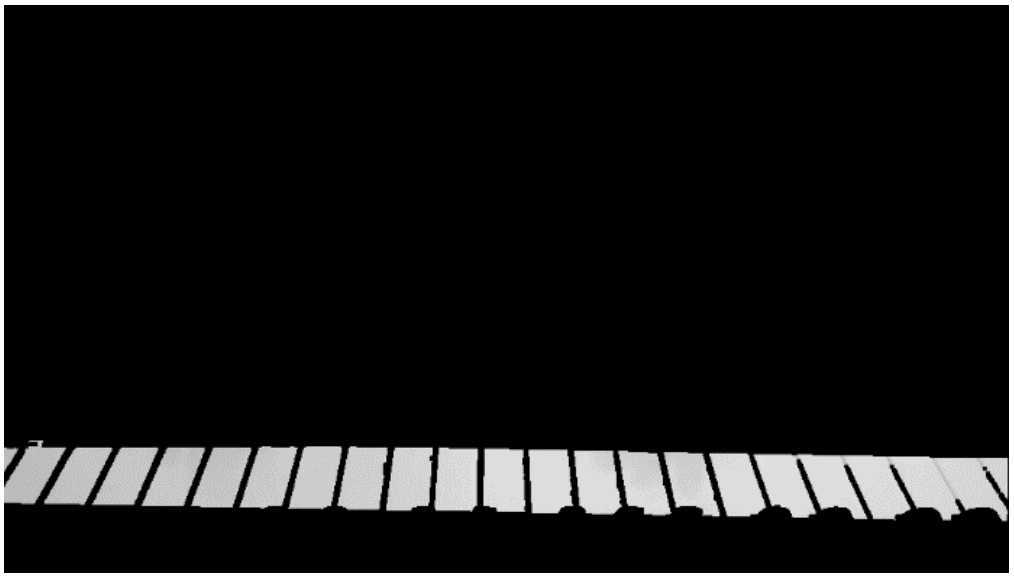


Figure 20: front camera filter results

A threshold of 80 and 110 votes were used for the regular Hough Line Transform of the upper and front cameras respectively to produce the horizontal lines. Hough transform can return multiple lines for the same real-world line because of distortions in the camera image [15]. To address this issue, nonhorizontal lines and horizontal lines which are closer than camera height/80 to a previously calculated horizontal line were eliminated to get potential white and black key lines. Because the view of the keyboard is inverted between the cameras, the line with the lowest Y

value on image coordinates is considered the white key line for the upper camera while the line with the highest Y value is considered the white key line for the front camera. The black key line for both cameras is the line with the second most Y value. Calculated white key line is drawn in red, and the black key line is drawn in green in Figures 22 and 25.

Figure 21: Upper camera canny edge detection

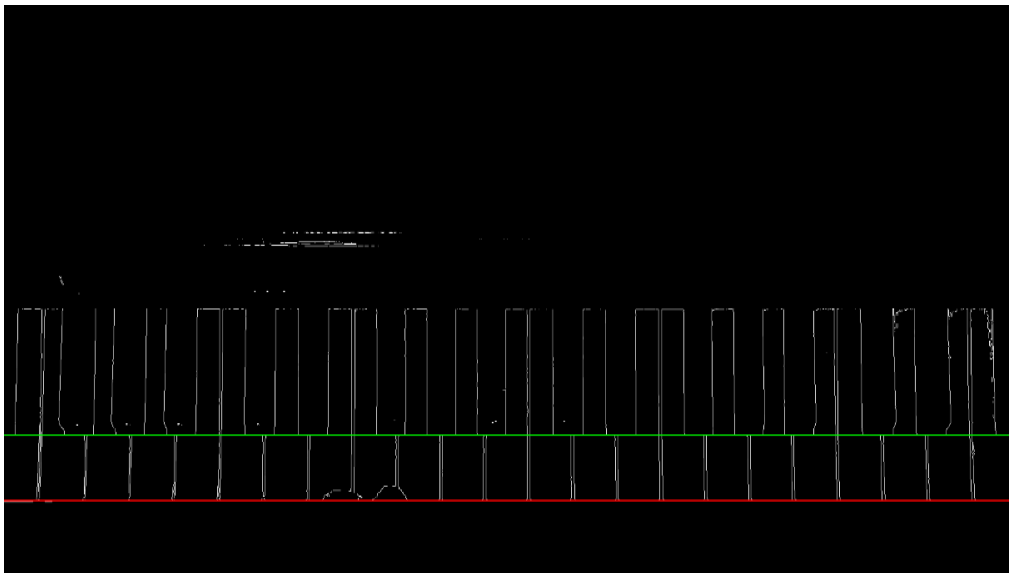


Figure 22: Upper camera black and white key lines

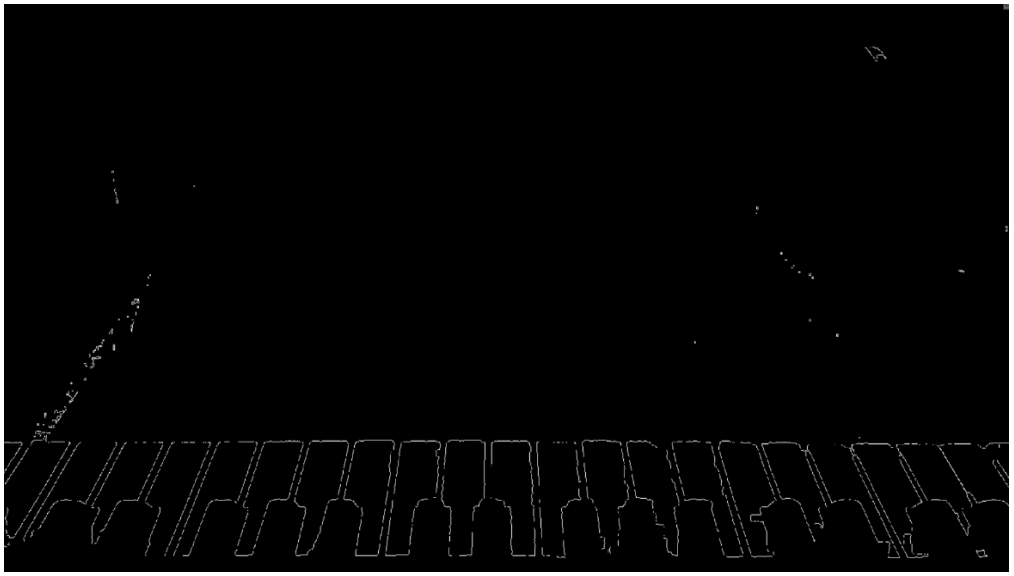


Figure 23: Front camera canny edges

The vertical lines coming from the front camera Probabilistic Hough Line transform represents the side edges of the piano keys and since they are parallel in the real world, their vanishing point on the white key surface is calculated by averaging their intersections. Figure 24 and 25 depicts the extended image with the vertical lines in blue, their intersection points in cyan and the average of the intersection points in white, which is the vanishing point. The Figures are extended on the y dimension to display the intersection points and the vanishing point which normally lie outside the camera view.

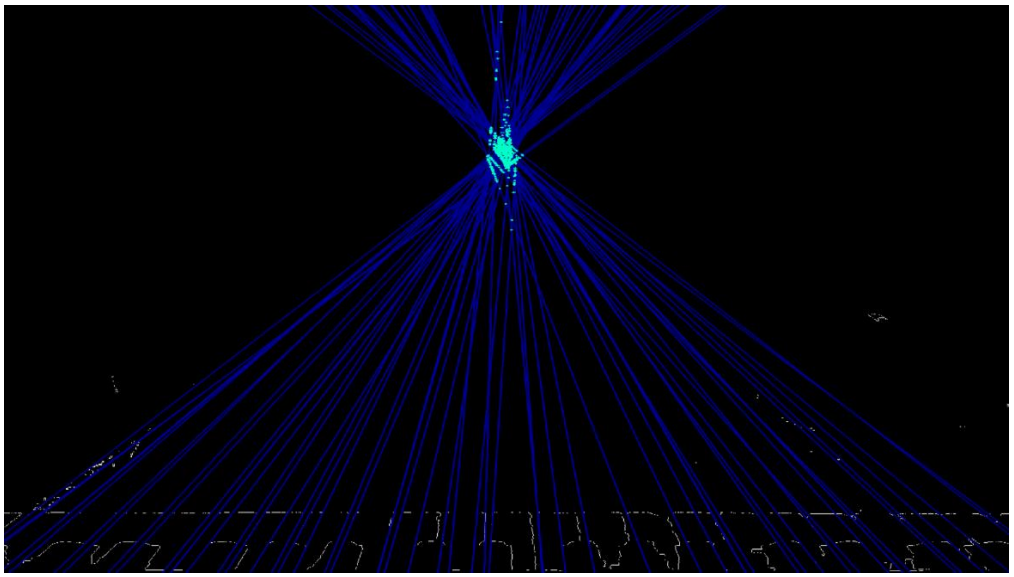


Figure 24: Extended front camera intersections of parallel lines

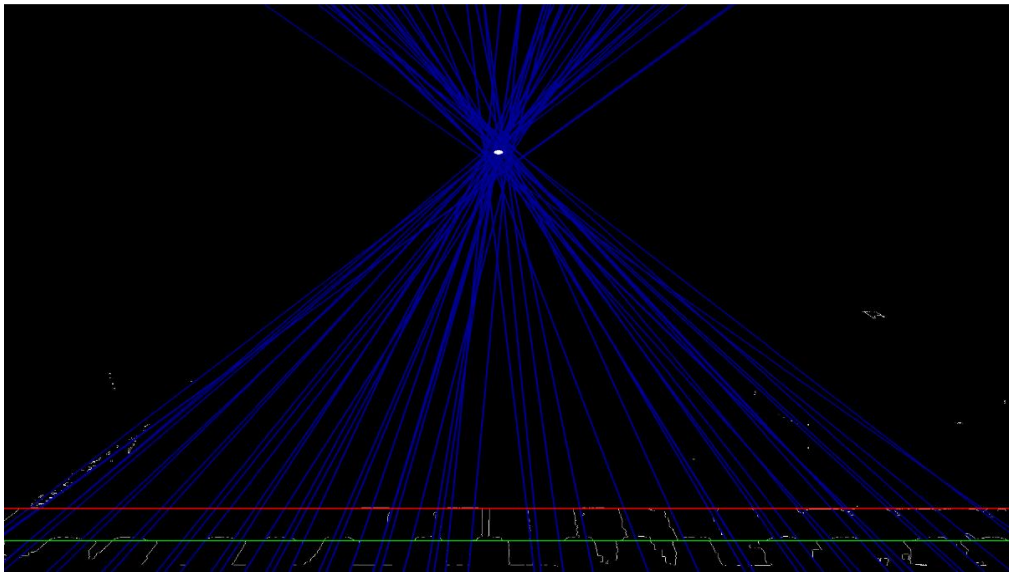


Figure 25: Front camera black and white key bottom edge lines and vanishing point

For the black key segmentation, a setup specific method was used. The red line above the piano keys is extracted from the upper camera image using the white and black bottom key edge lines. While dormant, the black keys cast a shadow on this line and when the keys are pressed this shadow disappears, allowing the key press to be recognized. YCrCb and HSL color filters are applied to the first 10 frames of the video to segment the shadows cast by the black keys. Then, erode and dilate operations are applied and part of the image which lies below the black key line is eliminated to reduce noise. These 10 samples are combined with an *AND* operation to further eliminate noise and only leave behind black key locations as rectangles as seen in Figure 26.

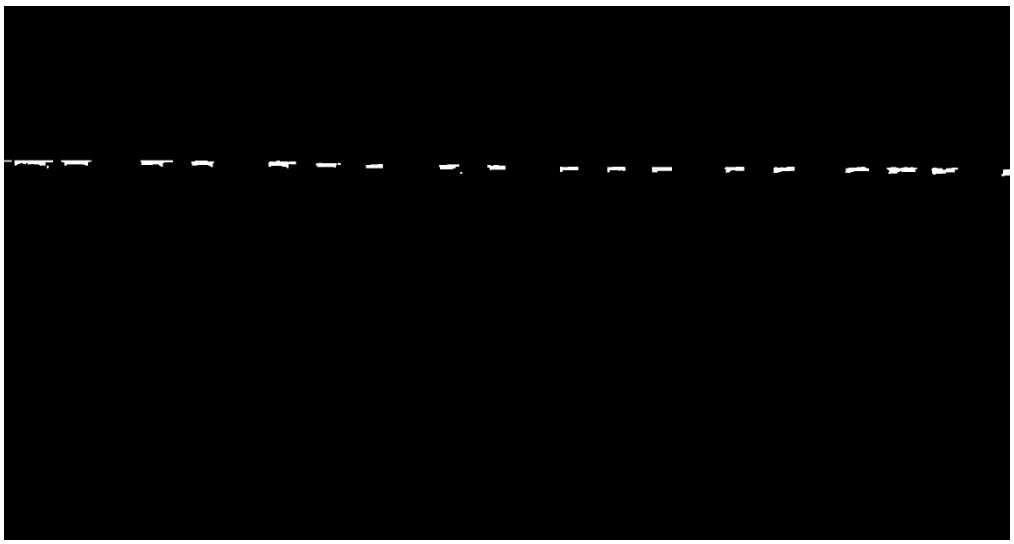


Figure 26: Filtered black key shadows

2.2.2 Key Indexing

Because of the angle of the front camera and the resulting distortion in the image, estimating the *X* world coordinate of fingertips directly is unreliable and cannot be used to match the fingers from the two different camera sources. Matching the piano keys and using them to match fingers offered a better solution because they reside on the same plane, corresponding keys of the two cameras were given the same index with the help of an equalizing factor of 2. To match the white piano keys, a mask was applied to the initial frame, which removed all the edges which did not reside between the white and black key lines. Canny edge detection was applied to this matrix to differentiate the edges.

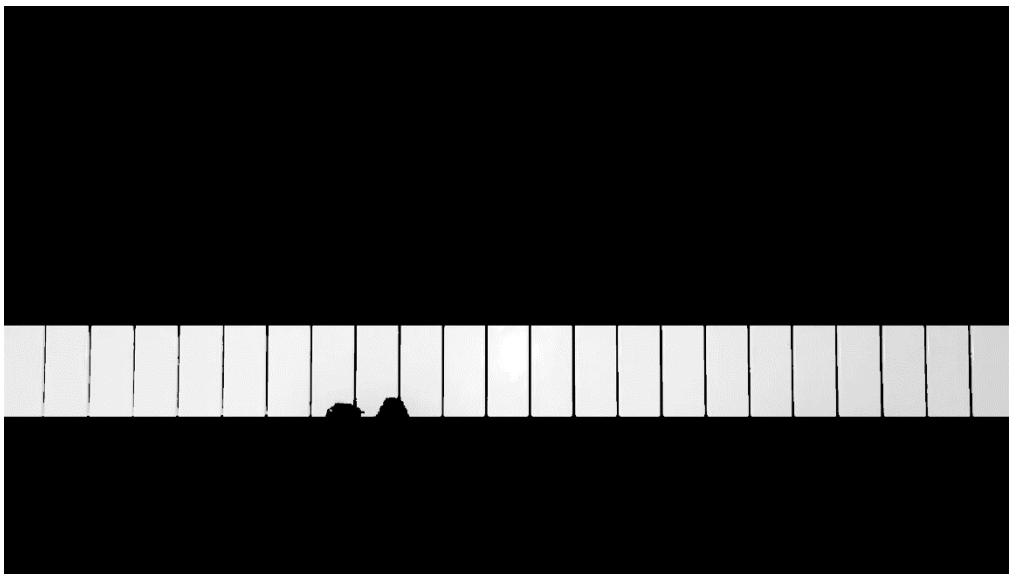


Figure 27: Segmentation of the white keys on upper camera

For the upper camera, the intermediate horizontal line between the white and black key bottom edge lines were calculated and the intersections of this line and the canny edges were used to find the *X* coordinates of the lines which separates the white keys. Vertical lines which pass through these intersection points are considered the side edges which separates the white keys, and the piano keys are indexed according to these lines.

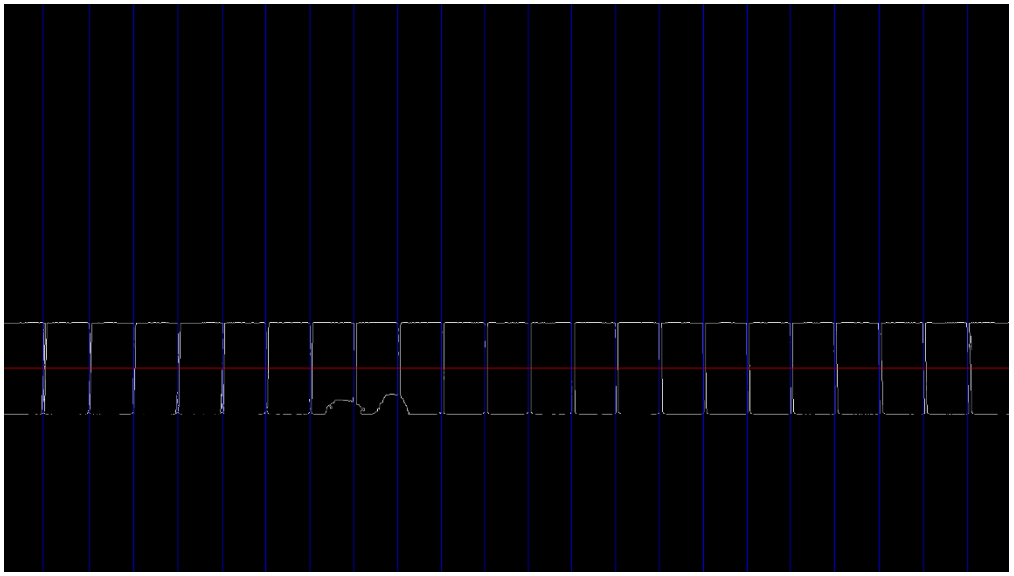


Figure 28: White: Canny edge detection on Figure 27, Red: Intermediate lines between white and black key lines, Blue: Calculated white key side edges

For the front camera, calculating a single X coordinate was not enough to represent a separation line between keys because of the distortion, so Probabilistic Hough Line Detection was used on the canny edges to calculate the separating lines. Only lines with slope greater than 1 are considered a candidate side edge and lines that represent the same side edge are eliminated with a distance threshold. Similar with the upper camera, these lines are used to index the piano keys.

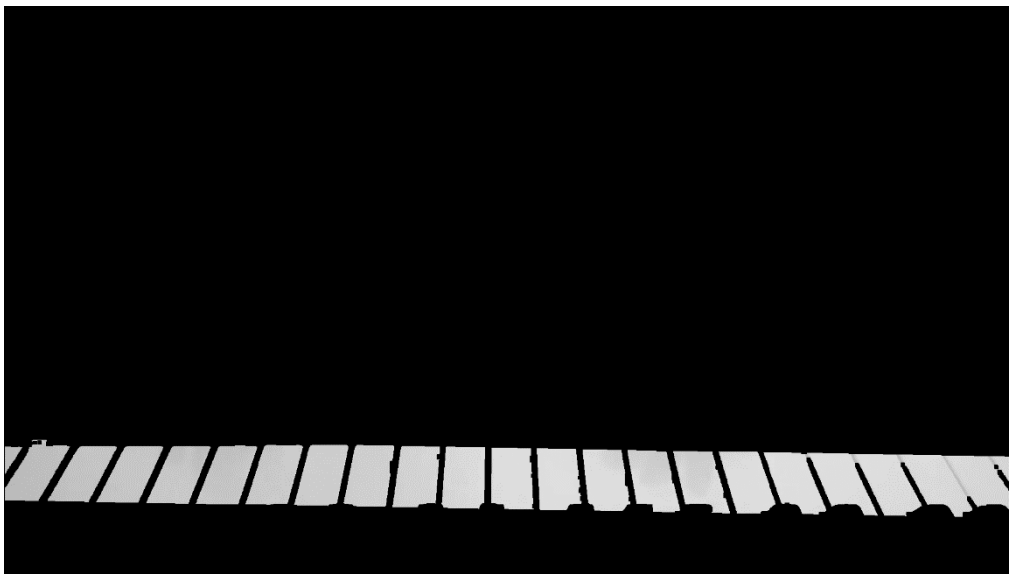


Figure 29: Segmentation of the white keys on front camera

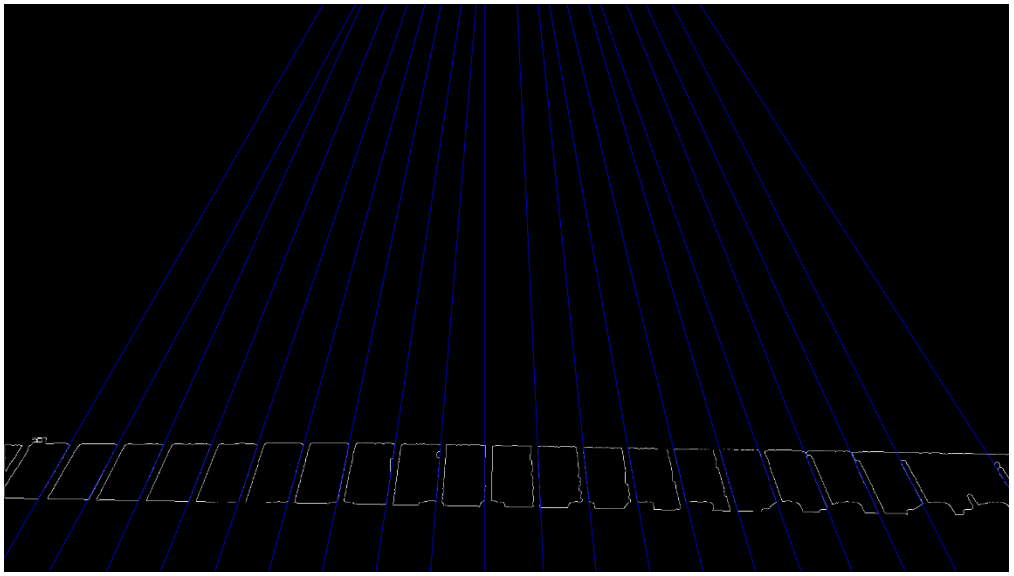


Figure 30: White: Canny edge detection on Figure 29, Blue: Calculated white key side edges

2.3 Hand Segmentation

The segmentation process starts with a YCrCb color filter for the upper camera and YCrCb, HLS and BGR filter for the front camera. The YCrCb and HLS filters were implemented with trackbars for calibration, they are simple filters which use OpenCV's cvtColor function to convert the color space and inRange function to eliminate pixels with lower values than the min and higher values than the max [8]. The calibrated filter values for the test video are as follows:

```
//top filter
int Y_MIN = 70;
int Y_MAX = 198;
int Cr_MIN = 141;
int Cr_MAX = 255;
int Cb_MIN = 0;
int Cb_MAX = 256;
```

```
//front YCrCb filter
int Y2_MIN = 92;
int Y2_MAX = 217;
int Cr2_MIN = 0;
int Cr2_MAX = 180;
int Cb2_MIN = 0;
int Cb2_MAX = 255;
```

```
//front HLS filter
int H_MIN = 0;
int H_MAX = 105;
int L_MIN = 0;
int L_MAX = 255;
int S_MIN = 0;
int S_MAX = 255;
```

The BGR filter of the front camera works different than the other filters. Instead of using min and max values, it does pairwise comparison of blue, green, and red intensities of a pixel to check if their difference is within a threshold of 35. This is used to eliminate the white-grayish colors while segmenting the hand, as white and grayish colors share similar intensities for all three components on the BGR color space. As seen in Figure 30, non-skin color parts of images are concentrated at black and white colors which share similar BGR values while skin color is closer to the red axis [6].

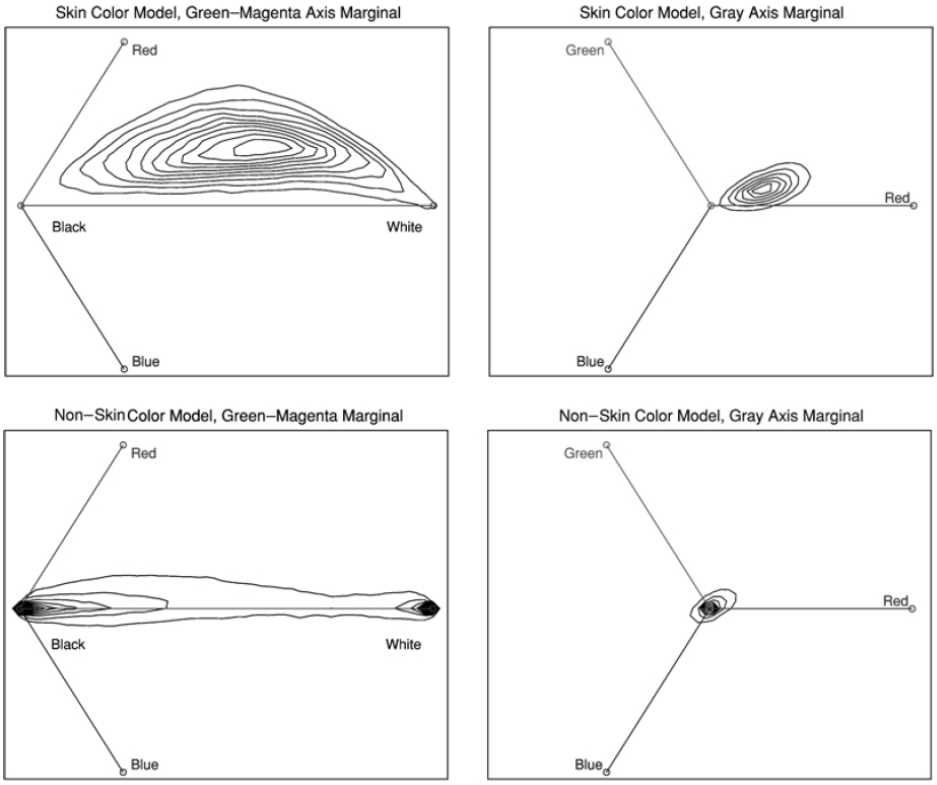


Figure 30: Distribution of skin and non-skin colors in images [6]

Figure 31 shows an example segmentation result only using the BGR filter. This filter is used specifically for the front camera because the shadows cast by the fingers onto the keyboard proved impossible to segment from the hands with the use of traditional filters.

Figure 31: Segmentation using only the BGR filter

The abundance of background objects and the position of the light sources which casted a shadow on the piano players fingertips at certain angles were detrimental on the results obtained from the front camera, so two additional filters were used. After applying the filters, the image is eroded to eliminate noise elements.

Figure 32: Upper camera segmentation mask

Figure 33: Front camera segmentation mask

2.4 Fingertip Detection

After hand segmentation, contours are extracted from the segmented image and the two contours with the largest sizes are considered the hands of the piano player. To detect the fingertips of the player, the convex hulls of these hand contours are calculated, which is the smallest convex set that contains it. These convex hulls are used by the convexity defects method and the local maxima method to calculate the fingertips.

2.4.1 Convexity Defects Method

The difference between the convex hull and the contour gives the convexity defects, which correspond to the empty space between the fingers. Each convexity defect has a start, concave and end point which are useful in determining if the defect represents a space between two fingers. If the angle between the 3 points is more than the threshold of 90 degrees, the defect is considered as noise. This threshold was set considering the physiology of the hand, as it is unlikely for two adjacent fingers to be more than 90 degrees apart. If a pair of points satisfies this constraint, the start and end points are considered as fingertip candidates. Fingertip candidates must have a lower *Y coordinate* than the white key bottom edge line to be considered fingertips. Finally, the distance of a new candidate to the previously determined fingertips must be greater than 40 pixels to eliminate overlaps because the index, middle and ring fingers are associated with two different convexity defects.

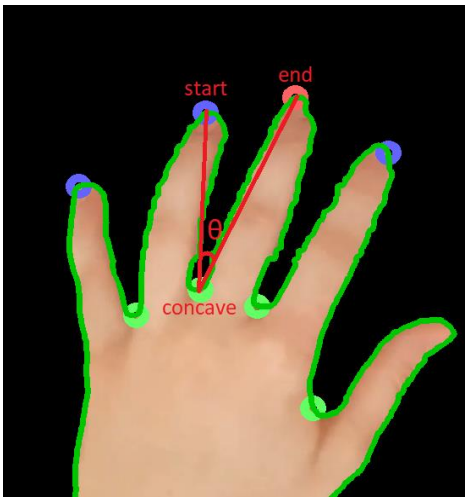


Figure 34: Convexity defect points and their angle

Convexity defects method is useful for fingertip detection because it provides reliable true positives and is indifferent to the orientation of the hand [16]. However, there can be difference between the fingertips detected by the two cameras because some fingers may not be detected when they are too close to another finger, this causes the cavity between them to disappear. It is also common for a piano player's hands to be bent while playing the piano [14], causing the fingertips to not lie on the convex hull and therefore prevents them from being recognized as a fingertip candidate. For these reasons we also applied a local maxima method to detect the fingertips missed by the convexity defects method.

2.4.2 Local Extrema Method

In this method, pixels of the contour are scanned to find local extrema. First the initial pixel is compared with the last one to determine if the sequence is increasing or decreasing and from that point onwards it searches for a change in this pattern to find local extrema as seen in Algorithm 1.1. We are interested in local maxima for the front camera and local minima for the upper camera because of the orientation of the hand. When a local extremum is found, the point or points on the convex hull with the same X coordinate are calculated. Because the convex hull only stores the corners of the hull, we calculate the lines between these corners, and amongst them the line which has a pixel with the same X coordinate. Finally, we compare the Y coordinate of the point on the convex hull with the local extremum to determine its vertical distance to the convex hull, if this distance is higher than a threshold then the extremum is considered as noise. Figure 35 depicts an example result of the algorithm, maxima are in blue; minima are in green and potential fingertips are in red.

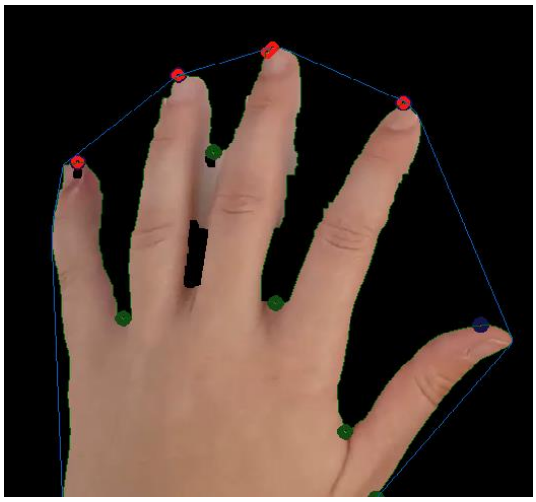


Figure 35: Local Extrema of Upper Camera

This method is used alongside convexity defects method because it can detect fingertips even if there are no cavities between them or if the fingertip does not lie on the convex hull. However, in some cases where fingers are close to each other there is no local extrema on the fingertip, but the convexity defects method can detect the tip if there is a cavity on the other side. Also, if the thumb or the little finger bends to reach a key, the maxima can lie on the finger joint instead of the tip so both methods have their merits.

Algorithm 1.1: Local Extrema

```
input  :  n : number of items in arr,  arr : array of elements
output :  all local maxima and minima elements in input array arr
1       vector<int> mx, mn;
2
3       if arr[0]>arr[1] then mx.push(arr[0]);
4       else if arr[0]<arr[1] then mn.push(arr[0]);
5
6       for i=1,2,3,…..n-1 do
7           if arr[i-1] > arr[i] and arr[i] > arr[i+1] then mn.push(arr[i]);
8           else if arr[i-1] < arr[i] and arr[i] > arr[i+1] then
9          mx.push(arr[i]);
10
11           if arr[n-1] > arr[n-2] then mx.push(arr[n-1]);
12           else if arr[n-1] < arr[n-2] then mn.push(arr[n-1]);
13       end
```

2.4.3 Fingertip Indexing

Each detected fingertip is compared with the key side edge lines of its respective camera to find the closest line. Afterwards, the second closest line is used to determine on which side the fingertip lies in respect to its closest line. This information is used to index each fingertip with the index of the piano key that it is currently residing on. Finally, the location of the fingertip in relation to the key is

calculated by comparing its distance to the two key side edge lines. This locational value is a value between 0 and 1, the value converges to 1 as the fingertip gets closer to the higher indexed key side edge line.

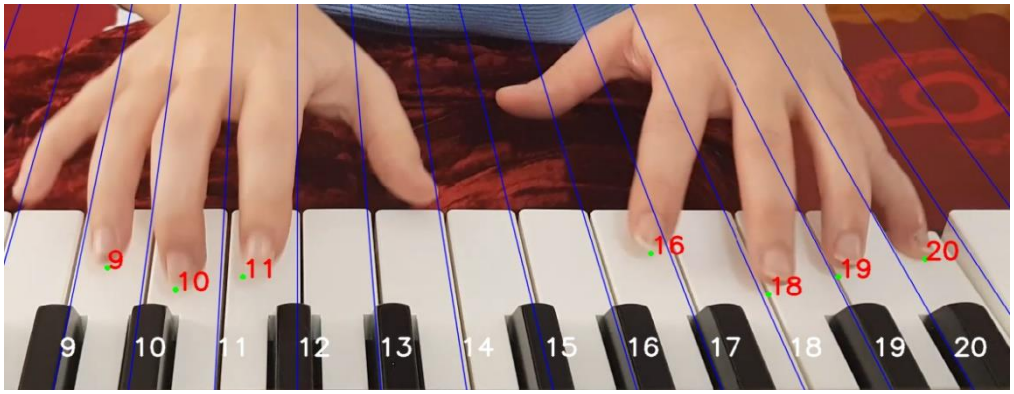


Figure 36: Front Camera Indexing Example

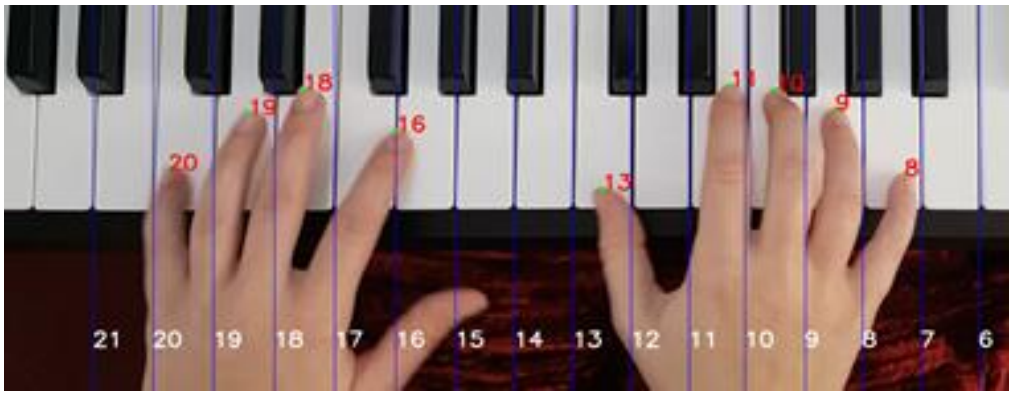


Figure 37: Upper Camera Indexing Example

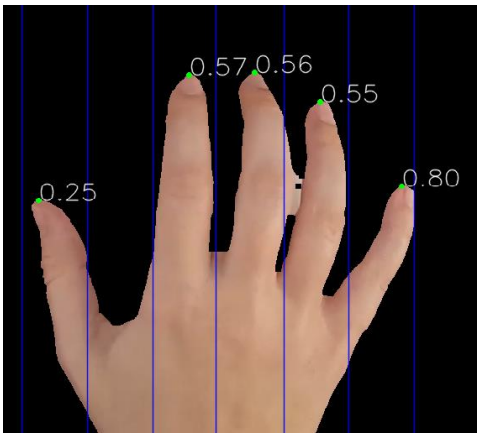


Figure 38: Locational values with respect to white key lines

2.4.4 Merging Convexity defects and Local extrema fingertip detection methods

For the upper camera, initially the fingertips calculated by the convexity defects method are inserted into the fingertips array. Then, each of the fingertips that are calculated by the local extrema method are compared with each element of the fingertips array to determine if they have the same key index or if they are closer to each other than a threshold of 30 to see if they refer to the same fingertip. If a match is detected with an element of the fingertips array, the fingertip candidate with the lower Y value is considered as correct and if it is the value coming from the local extrema method, it replaces the one from the convexity defects method. If no match is found, the fingertip candidate from the extrema method is directly added to the fingertip array. In Figure 39 the fingertips candidates coming from the convexity defects method are in blue and the candidates coming from the local extrema method are in red; the green circles indicate that the fingertip candidate is selected as a fingertip.

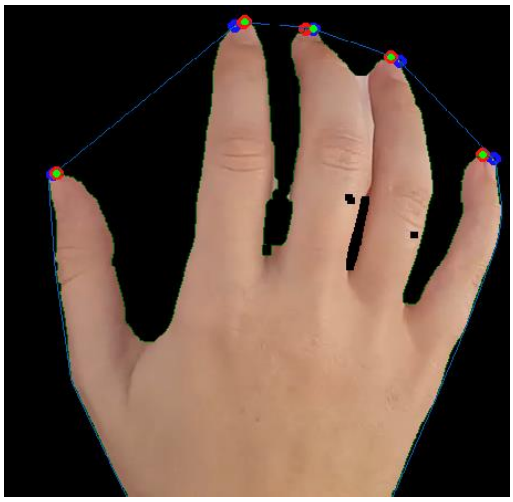


Figure 39: Upper Camera Fingertips

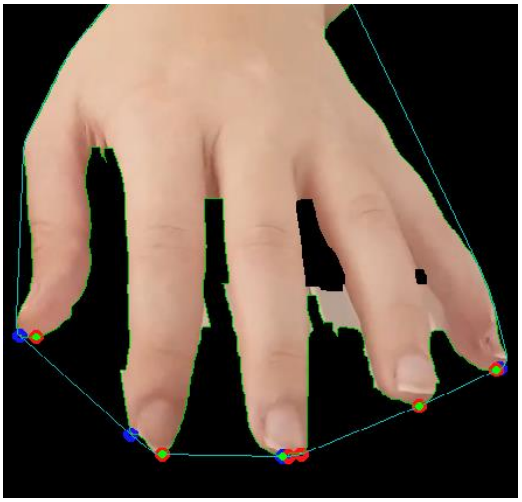


Figure 40: Front Camera Fingertips

The procedure is similar for the front camera but being in the same index is not taken into consideration while determining if two fingertip candidates refer to the same fingertip. Because of the angled view of the front camera, two different fingers being on the same key is a common occurrence. Instead, a larger distance threshold of 50 is used to eliminate fingertip candidates referring to the same key and in case of a match the candidate with the higher Y value is considered as correct. Figure 40 depicts the front camera results using the same color coding with Figure 39.

2.5 3D World Coordinates Calculation

Transformation from upper camera to world is implemented with perspective projection equations since the upper camera was parallel to the keyboards surface. X and Z coordinates can be extracted from the upper camera and height information is not available. Mi key's white key border is accepted to be the origin in the world coordinates as seen in the Figure 41.

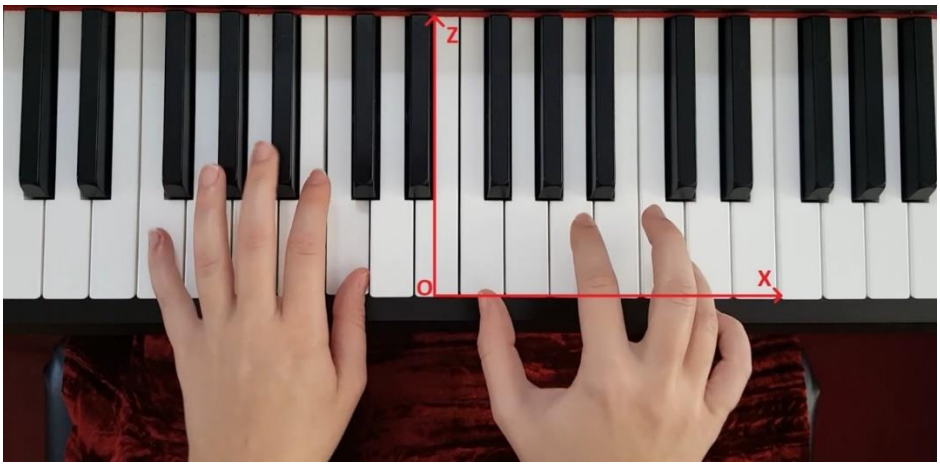
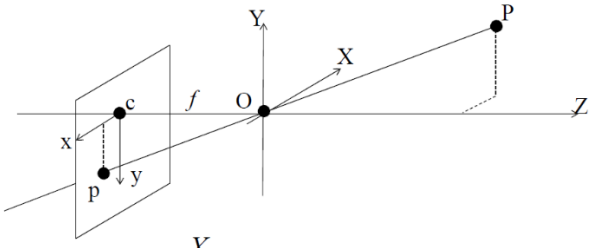


Figure 41: Upper camera respective world coordinates (X in world = X in OpenCV, Z in world = Y in OpenCV)

Transformation from front camera to world is implemented with using the cross-ratio invariance on the white key surface. To compare fingertip depths in both videos we aimed to calculate Z coordinate which is the same Z world coordinate with the upper camera as seen in the Figure 42. Here X and Y coordinates were irrelevant forour methods, so we won't be calculating them.

Figure 42: Front Camera respective world coordinates (X in world = X, Z in world will be calculated with cross ratio in OpenCV)

2.5.1 Upper Camera to World Coordinates



$$x = f\frac{X}{Z}$$

$$y = f\frac{Y}{Z}$$

According to the above perspective projection equation f and Z are needed to be known in order to find world coordinates X and Z(depth).

Distance from upper camera to white keys: 43.5 cm

White key top edge y coordinate in image: 308

White key bottom edge y coordinate in image: 799

White key length in pixels: 799 – 308 = 491

White key real world length: 14.6 cm

Upper camera parameter *f*: 491 * 43.5 / 14.6 ~ 1463

With the calculated f we could find X and Z coordinates of every fingertip in real world. We compared the values in random frames from the video and they were found to be almost the same with the real-world values. The precision is shown in the last 2 digits for example 1370 means 13,70 cm. The error is found to be negligible, and the accuracy is high enough to calculate which fingertip lies inside which key's borders.

For the Z coordinates we did the same calculations and found the distance between bottom white key edge and the fingertips as seen in Figure 44. Precision is shown in the last 2 digits as before.

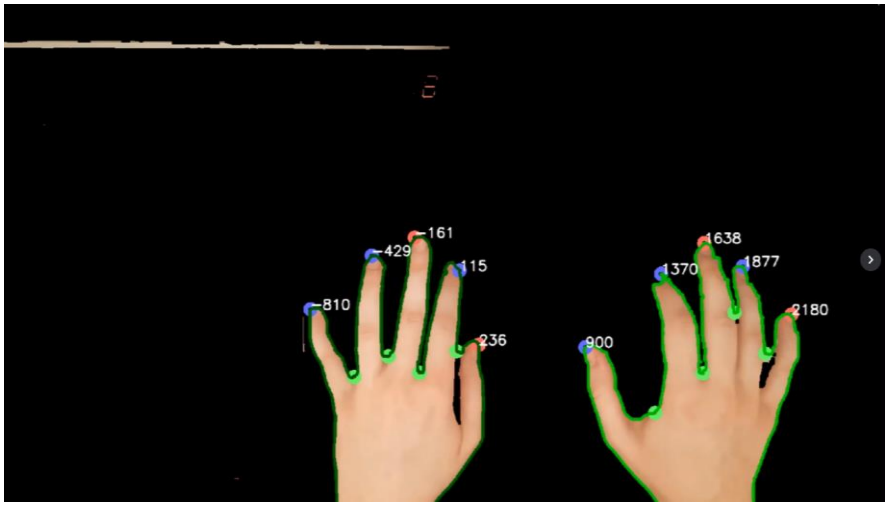


Figure 43: Transformation from image coordinates to world coordinate X

Until now we found *X* coordinates and *Z* coordinates which represent the depth of the finger. However, we will only use *Z* coordinates in the future because matching the fingertips from *X* coordinates was not possible due to distortion in the front camera. This will be shown in the Fingertip Matching section.

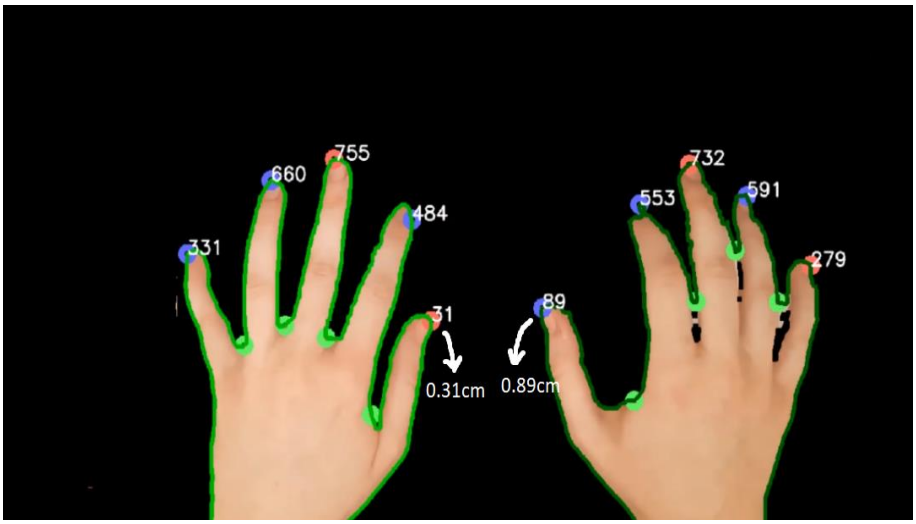


Figure 44: Transformation from image coordinates to world coordinate Z

2.5.2 Front Camera to World Coordinates

In this section we will explain how we extracted depth information from front camera using Cross ratio and vanishing point. When a fingertip is on a white key, it can be assumed to be in the same level with white key's surface so we can use cross ratio rule on this surface. We need 4 points for this and from the front camera footage

we see that there can be only 3 points on the same line that is on the white key's plane as seen in Figure 45.

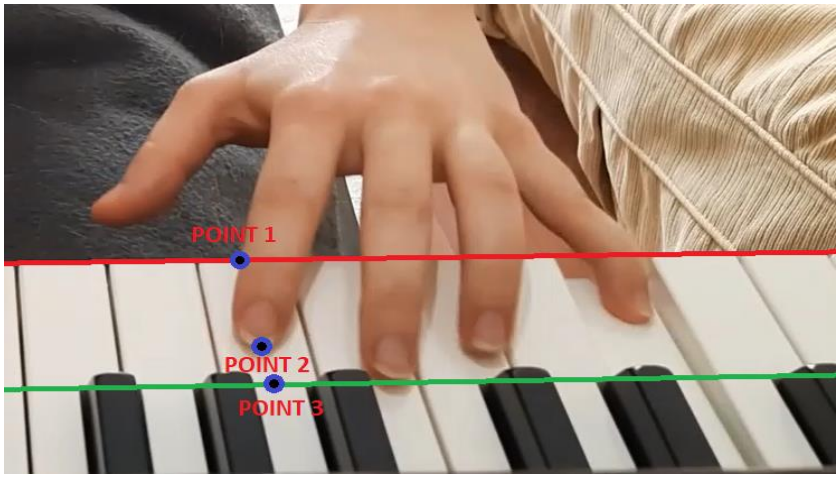


Figure 45: Three points on the white keys surface

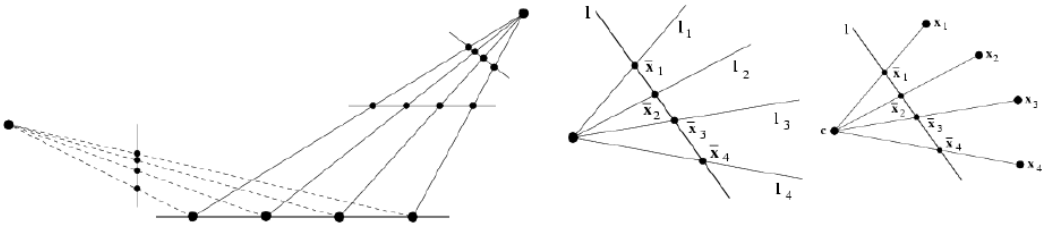
Point 1: White key top edge

Point 2: Fingertip

Point 3: Black key and white key border

To be able to apply cross ratio rule we need 1 more point which is the vanishing point. A set of parallel lines in the world plane which are the white key borders intersect in a point in the image which is the vanishing point. For the front camera we used this rule and the cross-ratio invariance in order to find the depth information of the fingertip on a white key.

$$CR(p_1,p_2,p_3,p_4)=\frac{\frac{x_1-x_3}{x_1-x_4}}{\frac{x_2-x_3}{x_2-x_4}}$$



p1 p2 p3 p4

As explained in the key segmentation section, we found the vanishing point and used it as Point 4. We will use the naming below from now on. These points can be seen in the Figure 46.

WP: White key top edge

FT: Fingertip

BP: Black key and white key border

VP: Vanishing Point

Cross ratio was found in the front camera as below.

$$CR = \frac{BP - WP \;/\; BP - VP}{FT - WP \;/\; FT - VP} \qquad (2.1a)$$

This value is equal to the value below in real world. Vanishing point corresponds to infinity in real world so they should be discarded. The points are taken from Figure 45.

$$= \frac{Point\ 3 - Point\ 1 / Point\ 3 - \infty}{Point\ 2 - Point\ 1 / Point\ 2 - \infty} \qquad (2.1b)$$

The formula becomes as shown here after we discard infinity distances.

$$= \frac{Point\ 3 - Point1}{Point\ 2 - Point1} \qquad (2.1c)$$

We need the value $Point\ 2 - Point1$ which is the depth of a fingertip in real world, and we know the value of $Point\ 3 - Point1$ which is measured as 5 cm on the piano in real world. (5 cm is 500 in the program to show 2-digit precision).

Therefore, the implemented formula in the project becomes:

$$Point\ 2 - Point1 \quad = \frac{(FT - WP / FT - VP) * 500}{BP - WP \;/\; BP - VP} \qquad (2.1d)$$

But the resulting Z coordinate of the front camera is different than the real depth because of the angle of the camera. And this gap is higher than the real depth when a finger presses a key. Besides when a finger is not touching the white keys and the fingertip is in the air cross ratio rule does not apply, this results in the calculated depth being less than the actual depth.

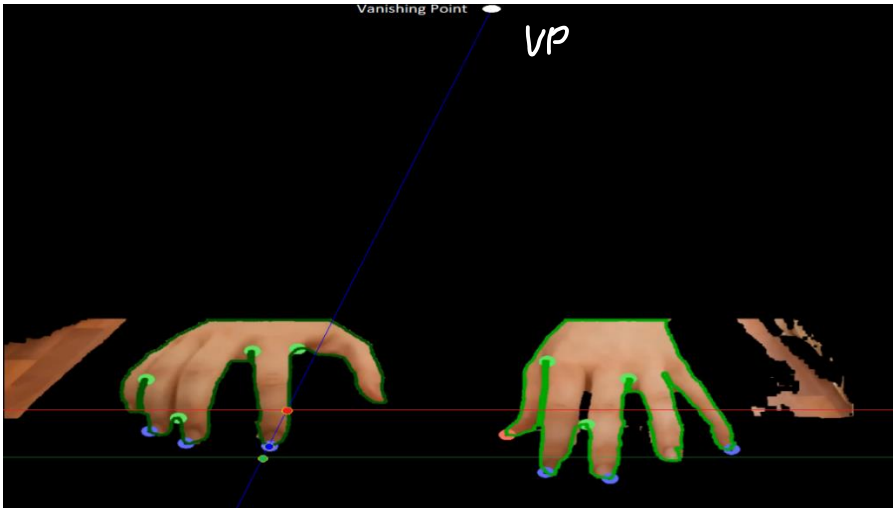


Figure 46: Four points needed to calculate cross ratio

2.6 Fingertip Matching

Calculated fingertips of the upper and front camera are compared depending on their indexes. When there is a match, the difference of their locational value with respect to white key lines is checked and if it is less than 0.25, they are considered a potential match. For each upper camera fingertip, the potential front camera match with the closest locational value is taken as its pair and pressed key detection is done comparing their calculated world coordinates.

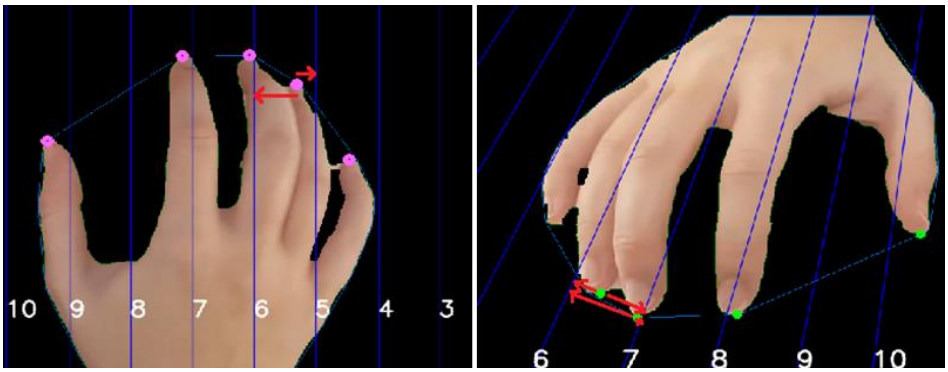


Figure 47: Comparison of locational value with respect to white key lines

2.7 White Key Press Detection

After matching the fingers, their calculated world coordinates are compared to determine if the white key that they are indexed on is pressed. The Z world coordinate of the front camera tends to be less than the Z world coordinate of the upper camera when the fingertip of the player is above the piano keys plane. This is because we calculate the Z coordinate according to the keyboard plane and the view angle causes the projection of the fingertip on this plane to be further away, causing its Z world coordinate to be less than the actual value which is obtained from the upper camera.

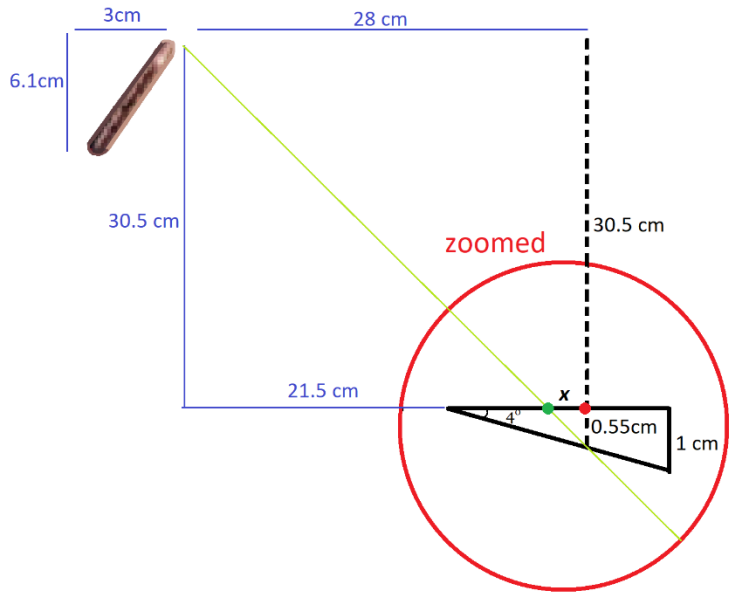


Figure 48: Pressed key detection threshold calculation

When the Z world coordinate of the front camera is less than the Z world coordinate of the upper camera, it is implied that the fingertip of the piano player is below the keyboard planar surface, which suggests a key is being pressed. We applied a lower threshold to determine if this value is a key press. Calculation of the threshold 50 which corresponds to 0.5 cm is shown in the Figure 48. According to the measurements when a finger presses the piano key on the point where the red circle is, front camera detects that point where the green circle is because of the angle. To put a threshold between the actual distance from the edge of the white keys and the approximated distance of the front camera x in Figure 48 is calculated. The point in red circle where the upper camera is looking is taken to be the reference point for all other points where the piano player can press a white key. Therefore, the distance

between the red and green circle determined the threshold. Similar triangles rule is used to obtain the following equation.

$$\frac{0.55}{30.5 + 0.55} = \frac{x}{28}$$

x is found to be 0.4959 and rounded up to 0.5 from the equation.

If -50 > $Z_{upper} - Z_{front}$, then the key is classified as pressed. The pressed keys are highlighted in cyan in the output video. To highlight pressed keys, we use the horizontal red line which shows the top of the keyboard, the horizontal white key line which shows the bottom of the white keys and the key indexing vertical lines. Figure 49 depicts the Z_{upper} values in blue and Z_{front} values in red and a detected key press.

Figure 49: Z_{upper} and Z_{front} values along with a detected key press

2.8 Black Key Press Detection

The black key mask from the first 10 frames of the video is subtracted from the subsequent filtered red lines on each iteration to display the color change on each frame. Contours are calculated on the difference and their area is compared with a threshold of 80 to determine if it is a key press. Contours that pass the threshold value are considered potential black key presses and their corresponding black key is calculated using the black key shadows of the initial frame for the horizontal location and the red line and the black key bottom edge line for the vertical location. Fingertips are iterated over to see if any of them fall inside this calculated black key

rectangle, and if any fingertip falls inside the rectangle, then this rectangle is highlighted in orange to display the key press as seen in Figure 50. The corresponding difference between the black key mask and the red line of the current frame is depicted in Figure 51, alongside the area of the detected contour. Alternatively, the lighting change of the black keys upon being pressed can also be used to detect black key presses, but it is subject to the noise coming from the piano player's hands [14].

Figure 50: A detected black key press

Figure 51: Black key mask difference at the time of Figure 50

2.9 MIDI Transcription

Midi is a file format to represent notes to play by an audio file. It doesn't contain the audio data itself but information like notes played, the duration of them, when they are played and how loud they are played are included inside midi files. In this thesis, it is important to see, as well as hear what this program deduces from the videos of a piano piece played. Therefore, a midi file is created using the detected pressed keys. As can be seen below the midi file is the transcription of every moment the piece is played into piano sheets.

Figure 52: An instance of some notes played

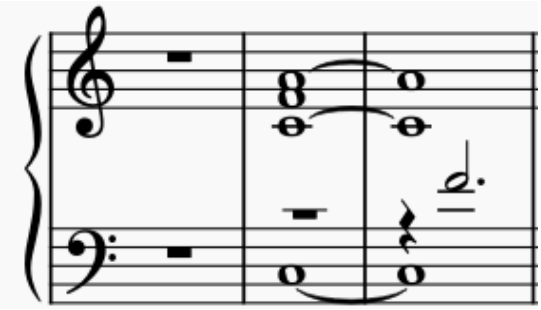


Figure 53: Midi file transcription of Figure 52

In Figure 52, by the right hand the notes *do, fa, la* and by the left hand the note *do* by one octave below are played. In the piano sheet shown by Figure 53 correspondence of the played notes are successfully transferred to midi file.

To score a note in the program, two functions are called, one for recording keystroke one for recording the removal. These functions add up the first arguments from the previous function calls which represent the time of press and remove. Second arguments are the encoding of the piano notes to identify them and the last one represents the loudness. To calculate the first argument, a frame counter and an elapsed time variable is used. Frame counter stores the current time while the program is running, and elapsed time stores the difference between the current time and the last time when a note is pressed or removed. Second argument is calculated by using previously explained key indexing by linking them to the corresponding hexadecimal codes of the notes.

3. Chapter 3

Experimental Results

In this chapter we show the results that we obtained from the “The Happy Farmer” piano piece from Schumann. We also present our results from the previous versions of our algorithm to clarify our reasoning behind the chances that we made. Accuracy, recall and precision of key press detection are used as evaluation metrics, and the tables are divided to show results from left and right hand. Initial and intermediate results are obtained using the “Happy_Farmer_Front” and “Happy_Farmer_Upper” videos while the final result is obtained using the “trial 2 front” and “trial 2 upper” videos.

3.1 The Happy Farmer Initial Results

Initially we experimented using only the convexity defects method for fingertip detection and did not take calculated fingertips into consideration for the black key detection. BGR white color filter was not implemented, and fingertips were matched based on the closeness of their calculated Z world coordinates instead of their estimated X coordinate on the indexed key. The results were as follows:

Table 1: Initial white key pressed detection result

White Keys	*Left Hand*	*Right Hand*	*Total*
Correct predictions	73	117	190
Missed key presses	24	94	118
False positives	29	9	38
Recall	75%	55%	62%
Precision	72%	93%	83%
Accuracy	58%	53%	55%

Table 2: Initial black key pressed detection result

Black Keys	*Left Hand*	*Right Hand*	*Total*
Correct predictions	14	26	40
Missed key presses	0	0	0
False positives	2	0	3*
Recall	100%	100%	100%
Precision	88%	100%	93%
Accuracy	88%	100%	93%

Using only the convexity defects method for fingertip calculation proved problematic on some frames of the front camera because the piano player has a wide range of finger motions and in some instances of the hand, the fingertips do not intersect with the convex hull which prevents them from being recognized. If a finger cannot be recognized on either camera, it cannot be matched and therefore a key press cannot be calculated, so we decided to incorporate the local extrema method.

The "*" present for the black key total false positives is caused by an instance where the upper camera refocuses, causing all black keys to be recognized as pressed for this instant. This error cannot be attributed to a fingertip or hand and is counted as a +1 because the key presses share a common cause. We decided to check the upper camera fingertips to confirm a finger is on the black key which is detected to be pressed to avoid situations like this.

3.2 The Happy Farmer Intermediate Results

In this version we combined the convexity defects method with the local extrema method to improve recall, but because our segmentation was lacking, we had to rely on smoothed contours for the local extrema method. This resulted in false positive key presses increasing and therefore the overall accuracy is not increasing substantially.

Table 3: Intermediate white key pressed detection result

White Keys	*Left Hand*	*Right Hand*	*Total*
Correct predictions	92	170	262
Missed key presses	5	41	46
False positives	54	40	94
Recall	95%	81%	85%
Precision	63%	80%	74%
Accuracy	61%	68%	65%

We initially used the local extrema method with smoothed contours because the segmentation was worse without the BGR white color filter, and without smoothing too many extrema points were being calculated. This was detrimental for our results because the locations of the fingertips were not precise enough which caused fingers to be mismatched. Fingers being mismatched is prone to cause false positives as fingers have varying world Z coordinates.

Hand segmentation needed to be improved to get rid of the smoothing, so we devised a specialized BGR filter to distinguish between the white keys and hands.

The addition of the BGR gray color filter tremendously improved our segmentation and allowed us to better tune other filters to lose less information about the hand while still segmenting the background. A side-by-side comparison can be seen below in Figure 54. Also, a new video was taken with a strong light source behind the piano to diminish the shadows that fall on the keyboard except the area under the hand.

To further decrease false positive key presses, we changed our fingertip matching algorithm to consider the X orientation of the fingertip on its indexed white key instead of the world Y coordinate.

Figure 52: Segmentation Improvements

3.3 The Happy Farmer Final Results

Table 4: Final white key pressed detection result

White Keys	*Left Hand*	*Right Hand*	*Total*
Correct predictions	80	181	261
Missed key presses	17	30	47
False positives	4	3	7
Recall	83%	86%	85%
Precision	95%	98%	97%
Accuracy	79%	85%	83%

Table 5: Final black key pressed detection result

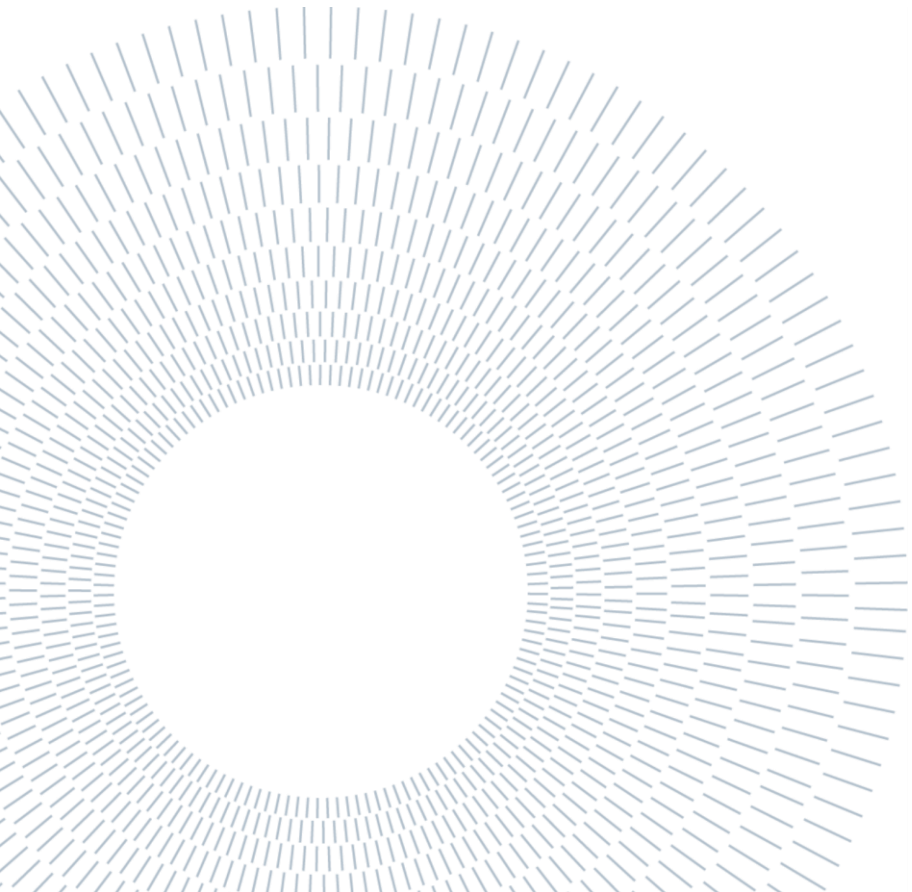
Black Keys	*Left Hand*	*Right Hand*	*Total*
Correct predictions	14	26	40
Missed key presses	0	0	0
False positives	1	0	1
Recall	100%	100%	100%
Precision	93%	100%	98%
Accuracy	93%	100%	98%

As seen from the results, precision is 15% better than the previous best result and recall is on par with the previous best result. Recall staying the same can be attributed to the reason of the missed keys. Missed key presses for the white keys happen mainly because of front camera fingertips being obscured by black keys or upper camera fingertips being obscured by other fingers. Former results in the front camera fingertip's world coordinates Y to be calculated lower than the actual value, causing a key press to fall below the white key press threshold. The latter problem outright prevents fingertip matching and therefore causes a miss.

Detection of the black keys are much more accurate than the white keys despite only relying on the upper camera because of the additional color information. A similar method for white keys can be explored, but it is outside the scope of this project which relies on two cameras and the contradicting positional information they provide. However, the application of the color method can prove difficult depending on the type of the piano or the lighting conditions of the room.

4. Conclusion and future development

In this thesis a completely vision-based pipeline is proposed to transcribe the notes played on a piano by any player excluding any audio or digital instrument-based help. The program includes five stages of preprocessing, hand segmentation, fingertip detection, fingertip matching and pressed key detection. Preprocessing is crucial for our system to work in every environment and for any player because they are the extrinsic parameters of the program. Another important point is that the hand segmentation step majorly determines the success rate of pressed key detection compared to the other stages, because the rest of the algorithm relies on it to locate the hand contour. The system proposed here lacks the capacity of capturing fingertips in every frame due to architectural impediments. For example, sometimes black keys intervene between the fingertips and even fingers obscure other fingers from time to time due to usage of one each camera. These problems can be attributed to the shape of the piano not allowing a clearer view for the cameras, or two cameras not being sufficient to record all the movements of a piano player. Capturing the scene perfectly using more cameras by decreasing the perspective may solve the problem in future works. A digital piano was used to shoot the test video which allowed us to put a camera in front of the hands. However, if an upright piano is used the placement of the cameras should be changed and put somewhere in up and in cross direction to the keys. It is also important to take into consideration that each additional camera would introduce execution overhead to the algorithm because of the need to process and match fingertips from additional visual sources. After a more convenient setup that can capture the fingertip positions at any time without being obscured, the intensity of pressing can also be added to the current system. Piano player's play with their emotions and the nuances reflects them. In future works how fast a fingertip presses a key by tracking it's vertical movement and at which depth the key is being pressed can be analyzed to add the detection the loudness factor.



Bibliography

[1] Caglioti, Vincenzo," IACV - 10- Single View Geometry".

[2] A. Y. Dawod, J. Abdullah and M. J. Alam, "Adaptive skin color model for hand segmentation," *2010 International Conference on Computer Applications and Industrial Electronics*, 2010, pp. 486-489, doi: 10.1109/ICCAIE.2010.5735129. Derfopps, Own work, CC BY 4.0, Available at https://commons.wikimedia.org/w/index.php?curid=74745305

[3] Derfopps, Own work, CC BY 4.0, Available at https://commons.wikimedia.org/w/index.php?curid=74745305

[4] Medioni Gérard and Sing Bing Kang. *Emerging Topics in Computer Vision*. Prentice Hall PTR, 2004.

[5] Lee, Jangwon, et al. "Observing Pianist Accuracy and Form with Computer Vision." *2019 IEEE Winter Conference on Applications of Computer Vision (WACV)*, 2019, https://doi.org/10.1109/wacv.2019.00165.

[6] Jones, M.J., Rehg, J.M. Statistical Color Models with Application to Skin Detection. *International Journal of Computer Vision* **46,** 81–96 (2002). https://doi.org/10.1023/A:1013200319198

[7] Boracchi,Giacomo,"2020_IACV_SingleView_Geometry",Available at https://boracchi.faculty.polimi.it/teaching/IACV.htm.

[8] "Color Conversions." *OpenCV*, https://docs.opencv.org/3.4/de/d25/imgproc_color_conversions.html.

[9] Akbari, Mohammad, and Howard Cheng. "Real-Time Piano Music Transcription Based on Computer Vision." *IEEE Transactions on Multimedia*, vol. 17, no. 12, 2015, pp. 2113–2121., https://doi.org/10.1109/tmm.2015.2473702.

[10] Vezhnevets, Vladimir, et al. "[PDF] a Survey on Pixel-Based Skin Color Detection Techniques: Semantic Scholar." *Undefined*, 1 Jan. 1970, https://www.semanticscholar.org/paper/A-Survey-on-Pixel-Based-Skin-Color-Detection-Vezhnevets-Sazonov/bc1b5ff4fdb70c10a9aa0e9b8f6b260b2e1f4fed.

[11] Yeung, "tutorial1", Available at https://ai.stanford.edu/~syyeung/cvweb/tutorial1.html

[12] Lee, Socret, "Lines Detection with Hough Lines", https://towardsdatascience.com/lines-detection-with-hough-transform-84020b3b1549

[13] Holland, Simon. "Artificial Intelligence in Music Education: A Critical Review." Readings in Music and Artificial Intelligence 20 (2000): 239-274

[14] Suteparuk, Potcharapol. "Detection of piano keys pressed in video." *Dept. of Comput. Sci., Stanford Univ., Stanford, CA, USA, Tech. Rep* (2014).

[15] A. Goodwin and R. Green, "Key detection for a virtual piano teacher," *2013 28th International Conference on Image and Vision Computing New Zealand (IVCNZ 2013)*, 2013, pp. 282-287, doi: 10.1109/IVCNZ.2013.6727030.

[16] Dhawan, Amiraj, and Vipul Honrao. "Implementation of hand detection based techniques for human computer interaction." *arXiv preprint arXiv:1312.7560* (2013).

List of Figures

List of Tables