Part 3. Main Report

1. Literature study

1.1 Existing Systems

Similar systems to the proposed system were reviewed. An android-based virtual piano application was proposed by Begum et al. [1]. This system made use of smartphones with high-resolution cameras that were pointed towards a keyboard drawn on a piece of paper. The application processed the image of the paper keyboard in real time and then detected which key was touched by the finger of the user. The application would then produce the corresponding note of the detected key. Sun and Chiang [2] proposed a portable piano tutoring system. Their system required a portable micro-projector, an external webcam, a smartphone and a wireless video streaming device. The micro-projector would be used to project a keyboard layout onto a surface, and the webcam would be connected to the smartphone to capture the movement of the user's hand. Systems that transcribed notes played from a real piano using computer vision were seen in the works of Akbari and Cheng [3], and Goodwin and Green [4]. The system in [3] consisted of a standard webcam being placed directly above the piano at an angle between 30-45 degrees from vertical and could capture the entirety of the length of the piano. Similarly, the system in [4] had the webcam placed directly above the piano, however much closer to the keys and hence only a segment of the keys was captured.

1.2 Key Detection

The proposed system in [1] had an initialization stage where the frame of the captured keyboard was converted from the Red, Green and Blue (RGB) colour space to the Hue, Saturation and Value (HSV) colour space. This conversion was performed since the application was intended to be used with varying light and shadow effects. The value channel was then extracted and binarized. This binarized frame was then used for the key detection stage which was performed by implementing the Moore-Neighbor method to identify the contours of the white keys. The frame was then inverted followed by morphological operations to identify the regions of the black keys. Begum et al. chose this method for its fast performance and reliability. The entire key detection algorithm is implemented in real time. The authors of [2] performed their key detection algorithm in the pre-processing stage and hence this was not computed in real-time. Therefore, the pre-processing stage consisted of detecting contours and identifying the piano keys. Their contour detection was performed by using an iterative method to derive threshold values of the grey-scale frame and convert them to a binarized frame. An initial threshold value of 128 was set and higher and lower pixels were separated into two groups. The averages of both groups were then calculated and a new threshold was obtained by calculating the mean of the two averages. If the difference between the two threshold values was small, the iteration ended, or else the iterations occurred again. The corners of the keyboard were then detected and the detected layout was separated into segments.

The system by [3] considered the keyboard to be a quadrilateral shaped by 4 lines. The Hough transform was applied to obtain these lines and the transformed rectangle that consisted of all the keys was acquired. The Otsu thresholding method was used to distinguish between the black and white keys. A similar approach was performed in [4] where the Hough transform was applied to

detect the lines across the keyboard. This system required the keys to be detected in real-time like that of in [1], and therefore, an accumulator threshold was found such that the lines of the keyboard could be detected even when a hand is placed on the keys.

To detect the keyboard layout, edge detection techniques were explored apart from the methods used in [1] and [2]. A common edge detection technique used in computer vision is the canny edge technique. Khurram et al. [5] implemented a modified canny edge technique to identify two different classes and types of fruit. Their pre-processing stage consisted of converting the RGB image into that of a grayscale image. Their edge detection method consisted of smoothing the input image by computing the Gaussian kernel for the vertical and horizontal gradient of each pixel. The angular direction of each pixel was then computed, followed by a non-maximal suppression method to calculate candidate edge points. A thresholding process was then followed where the upper threshold was considered to be 1.562 times higher than the lower threshold. The final steps of the proposed method consisted of relating stronger edges to weak edges and then implementing zero padding to remove unwanted edges.

Another method that was explored to aid in detecting the keyboard and fingers was template matching. Since template matching generally requires the template to be rectangular, Truong et al. [6] proposed a template matching implementation that matches an arbitrarily shaped template using the idea of coarse-fine and refining interesting regions. Their method's first step was pattern extraction and resolution reduction. The desired region of the template was extracted by applying a binary mask to the template. The resolution of the template was reduced by dividing the template into equal blocks and the desired region was refined. A descriptive vector of the template with low resolution and one with high resolution was then computed. The second step of the method performed matching on low-resolution images. For each pixel of the image, the low-resolution vector is matched with the pixel. The purpose of this step is to identify candidate-matching pixels. The third step of their method performed matching on full-resolution images. Only the candidate pixels obtained from step 2 were used in this step. When comparing their method to that of the FFT method, it was found that the accuracy rate between the two methods was similar however the proposed method was 4.7 times faster than the FFT method. Another implementation of template matching was done by Wu and Toet [7], where they presented a method that speeds up template matching through integral image-based weak classifiers. Their method consists of two phases where the first phase was similar to step 2 of [6] as a set of candidate target locations was determined. This was performed using integral images to compute weak image block binary test patterns and then cascade a set of them. The second phase involved performing template matching on the regions of the candidate target locations. They compared their proposed method to an existing FFT method from the OpenCV library and observed that in some cases, the proposed method was able to correctly detect an object while the OpenCV method failed. Furthermore, the proposed method provided a speed enhancement of no less than one order of magnitude when compared to the FFT-based method implemented by OpenCV.

1.3 Finger Detection

To play the piano, fingers need to be detected. [1] found in studies of Chaves-Gonzales et al. [8] and Khan et al. [9] that the HSV colour space yielded better results in unconstrained environments. [1] then computed the histograms of each channel in the HSV colour space to obtain the threshold values. Using the obtained threshold values, masking was applied to detect

skin pixels within the image. The image then had the borders smoothed and the regions filled. The position of the fingertip was then computed by tracing the boundary of the mask with skin tone pixels and then computing the maxima. Similarly, [2] performed skin detection by testing between the YCbCr colour space and a trained Gaussian Mixture Model (GMM). The YCbCr colour space was chosen due to it requiring less operation time making it ideal for real-time purposes. The skin colour regions of the hands were identified based on quantized skin colour areas. Vishal and Lawrence in [10] also chose the YCbCr colour space for the hand segmentation algorithm in their system. Their method to track the user's finger was to compute the outer contour of the mask and obtain the topmost point. However, this approach has the drawback of only allowing a single finger to be used to play their respective keyboard.

To detect skin pixels, the work of Muhammad and Abu-Bakar in [11] was also explored. These two authors proposed combining the colour spaces of HSV and YCgCr to form a custom colour space of SCgCr. They derived ranges for each channel to identify where skin pixels are detected. They compared their custom colour space to that of the CbCr colour space and concluded that using their custom colour space resulted in a higher detection rate of faces. Fingertip detection was further explored in the work of Wang et al. in [12] where they used the YCrCb colour space, however, split the Cr and Cb channels and then implemented histogram equalization on each channel. The Otsu algorithm was then implemented to obtain a binary image combined with noise. The noise was removed using morphological operations such as opening and closing and at the end of their method, they were able to detect only the hand in their sample image.

Morphological operations were also used in the work of Singh et al. [13], where the authors applied it after a Sobel edge detection procedure to extract the regions of the eyes on a face. The morphological dilation process was used to grow the regions of the edge detected image which was then followed by a conditioned dilation procedure. The face was extracted from the conditioned dilated output by performing geometrical operations. To extract the eyes from the segmented face, the morphological closing operation was performed to separate the background and foreground from the image. Lastly, an erosion operation was performed using a disc structuring element to remove the contour of the face and extract the possible regions of the eyes.

1.4 Template Matching Alongside Skin Detection

The authors of [14] proposed a novel technique to detect faces in colour images using a skin colour model algorithm combined with skin likelihood, skin segmentation, morphological operations and template matching. Their approach consisted of building a skin colour model using the YCrCb colour space and they determined the colour distribution of human skin in the chromatic colour space by using 50 skin samples from 20 colour images. The effect of noise present in the skin samples was reduced by filtering the samples with a low-pass filter. A Gaussian model was then applied to acquire the likelihood of skin for any pixel in an image. Their resultant skin model could show the likelihood of a pixel being recognized as a skin pixel based on the intensity of the pixel in the grayscale space. Skin segmentation was then performed through a thresholding process which was followed by performing the morphological operation of opening on the resultant image. Region labeling was then performed on a clustered group of pixels and a Euler test was performed to distinguish the region as a face. Lastly, template matching was performed using a grayscale template of a standard face. Their face detection model was tested against 50 colour images that contained faces. They found that their model detected 91 faces from a total of 94 faces, which therefore achieved a high detection rate and a low false acceptance

rate.

1.5 Touch Detection

Regarding the detection of the touch of a key, [1] used the method of identifying whether the position of the fingertip fell in the region of the keyboard. The corresponding sound from that key was then played. Similarly, [2] detected the presence of skin-colour regions at validation points of each key. To account for triggers due to noise, a rule was formed that a validation point is only considered to be touched if the skin pixels are detected for more than two frames. Their system also incorporated a tutorial mode where the user was expected to touch a specific key and if the incorrect key was touched, the system would not play the corresponding sound. [10] used shadow analysis to detect whether a key was touched or not. The tip of the shadow of the finger was identified and the Euclidean distance between this and the actual fingertip was calculated. If the distance fell below a specific threshold, the system assumed a touch had occurred. This method also checked whether the finger and the shadow fell within the regions of the keys.

In regards to capturing a surface from a non-orthogonal position, the method of perspective transformation was explored since the proposed system could use a webcam that is placed in such a position. Using a dual-camera system to view a surface requires the use of a perspective transformation to map the points from one plane to another. The authors of [15] used planar homography to correct the camera's perspective when capturing a number plate from a non-orthogonal position. Their method involved obtaining 4 points of the number plate and then computing the homography matrix. With the use of planar homography, the recognition accuracy of the automatic number plate recognition system was improved by three percent.

1.6 Application of Background

The current literature supports the proposed system by providing foundations for various parts of the system.

The methods found in [1], [2], [6] and [10] to detect the keyboard and edges, provide a theoretical understanding of how to detect the keyboard regions in the proposed system. The idea of using template matching to detect the keyboard layout in the proposed system was inspired by the works of [6] and [7]. Various colour spaces were also explored in this literature study and the results acquired from these colour spaces assisted in choosing the specific colour spaces to aid in both the keyboard regions as well as detecting skin pixels for finger detection in the proposed system. The authors of [13] show that morphological operations are powerful tools used to either remove unwanted features or enhance desired regions from an image. These operations were applied in three of the subsystems of the proposed system. The proposed system used the process of erosion to extract the black keys from the keyboard layout, the process of closing combined with dilation was used to enhance skin pixels and increase the accuracy of detecting the hand and fingertips in a webcam frame. The work of [14] inspired the combination of template matching and skin detection to detect the hand and its fingertips in a webcam frame. Therefore, the combination of morphological operations, skin detection and template matching was all used to detect fingertips from the frames of the webcam.

Regarding touch detection, shortcomings exist in the fingertip detection methods of [1] and [2] as the systems only allow for the detection of a single fingertip and hence only a single key can

be touched at a time. A shortcoming was also observed in the work of [10] as their system relied on shadow analysis to detect a touch which requires the light to be positioned in such a way that shadows appear in a specific direction on the paper. The concept of using a perspective transformation within the proposed system was inspired by the work done in [15].

2. Approach

The design of the proposed system involved exploring various computer vision techniques. The proposed system used a dual-webcam setup where one webcam would be mounted overhead and primarily used to detect the regions of the keys and identify if a user was placing their hand above or below the black keys of the layout. A second webcam was placed in front of the layout and it was primarily used to detect the user's fingers and where a touch event occurred.

Detection of the keyboard layout was tackled first using the method of the Canny edge detection to detect the lines of the keyboard layout. This edge detection technique was implemented in first principles and the result was compared to that of the existing library implementation available in OpenCV. However, an issue arose as a contour-finding technique was not successfully implemented at the time which was necessary to trace the edges that resulted from the Canny edge technique.

Therefore, an alternative method to detect the layout was explored which was template matching. Template matching was chosen as it provided a simple yet efficient solution to detect the keyboard layout as it leverages the consistent layout of the keyboard, which remains unchanged regardless of the placement. This allows the keyboard layout to be a constant parameter in the detection process, irrespective of any variable surface textures or background lines. Template matching also offered the advantage of detecting a layout that consisted of more keys without increasing the computational complexity as would occur with the canny edge technique where if the number of keys grows, a larger number of lines would have to be detected. Therefore, template matching was used to detect the region within the frame where the keyboard layout was present. A drawback of using template matching to detect the keyboard layout was that it would require the layout to be of the same size as the template to be detected. This would mean that the camera would have to be placed at a specific position to successfully detect the layout. This was not desired, and therefore, an adaptive template matching algorithm was designed to allow different sizes of the keyboard layout to be detected providing the flexibility to keep the camera at various distances from the layout. The next step in the key detection process was to segment the keys within the layout. Morphological operations were used to separate the black keys from the white keys and a flood-fill technique was implemented to detect the regions of the keys within the frame.

Detection of the fingers was then tackled by exploring an object detection model such as YOLO [16]. Online datasets of fingernails were used to train the YOLOv3 model to detect fingernails in real-time. The model was trained using PyTorch and performed well on a laptop. However, the performance of the model was unable to be tested on the Odroid-N2+ as PyTorch was not successfully installed on the board. Tensorflow was then utilized to build CNNs of different architectures to detect fingertips. A custom dataset of fingertips was collected and labelImg was used to annotate each fingertip in the images. Data augmentation techniques were also explored and utilized to increase the size of the existing dataset. An issue arose as the custom models were unable to accurately detect fingertips in real-time. The exact cause of this issue is unknown, but it's speculated that either the CNN architectures lacked sufficient complexity or the dataset required further refinement. Due to time constraints, further exploration of CNN-based approaches was discontinued.

Therefore, skin detection techniques were explored to detect skin pixels in the frame as the proposed system would require hands to touch the keyboard layout. The concept of combining

skin detection with template matching was then explored. This approach was similar to that performed by [14], however, the template used in the proposed system was of a skin-masked hand instead of a standard greyscale template. This method was chosen as it did not require having to train a model and then passing frames into the trained model to detect hands and fingers. Therefore, the computational complexity of detecting fingertips was reduced. The tradeoff of the skin-based template matching method was that it was dependent on light and would not be as robust as the YOLO model. Another drawback to this method was that clothing in colours similar to human skin could significantly interfere with hand detection, making it difficult to distinguish hands from similarly coloured areas. The proposed system used template matching to detect the hands in the frame, and template matching was performed again to detect the fingertips.

Touch detection was tackled by tracking the fingertips and identifying if a fingertip went past a specific threshold line. A perspective transformation was applied to map the point of a touch from the view of the front webcam to that of the overhead-mounted webcam. An issue arose with the initial touch detection algorithm where the system would detect false double touches decreasing the accuracy of detecting the notes touched. This issue was overcome by tracking the frame that a touch event was triggered and ensuring that another touch event would only be triggered if a certain number of frames had passed. This algorithm also took into account simultaneous key touches and the assumption was made that simultaneous key touches occurred in a single frame. The drawback to this method was that the system was unable to detect up to 10 fingers touching the keys simultaneously as it was difficult to ensure that all the fingers touched keys within the same frame. This drawback occurred due to the limitation of the skin-based template matching method of not guaranteeing the detection of all the fingers in the same frame. An alternative contour-based fingertip detection method was explored to overcome this limitation however it was not reliable in detecting fingertips.

Intensity detection was tackled by first tracking the speed the fingertips moved between two threshold lines. This approach was not reliable since the locations of the detected fingertips change in every frame. Since the detection of the hand was much more stable than the fingertips, the velocity that which the hand moved between the threshold lines was calculated to determine the intensity of a touch event. If the velocity was larger than a specific threshold then the assumption was made that the user desired to play the note loudly.

3. Design and implementation

3.1 Design summary

Table 1 summarises the project design tasks and how they were implemented.

Deliverable or task	**Implementation**	**Completion of deliverable or task, and section in the report**
Design of a key detection algorithm	The key detection algorithm was designed from first principles using methods such as template matching, Otsu's thresholding, erosion and using a flood fill algorithm. The Scipy library was used to obtain the 2D FFT of the image, however, the rest of the template matching code was implemented from first principles.	Completed Section 3.4.1
Design of a finger detection algorithm	The finger detection algorithm was designed from first principles using a combined method of skin pixel detection and template matching. The colour conversions were implemented from first principles. An adaptive skin mask algorithm was designed from first principles by combining skin pixel detection and template matching to obtain optimal values for the skin mask for various lighting conditions. Morphological operations were implemented from first principles and used to refine the detection of the hand and fingers. Masking regions on a frame was applied as well to refine the finger detection algorithm.	Completed Section 3.4.2 Section 3.4.1

Design of a touch detection algorithm.	The touch detection algorithm was implemented from first principles using the method of tracking the fingertips past threshold lines and using a perspective transformation to map the point of a touch event. The detection of skin pixels was also used to detect the touches of black keys. To increase the accuracy of touch events for black keys, a remapping algorithm was designed to remap a point to the nearest black key.	Completed Section 3.4.3 Section 3.4.2
Design of an intensity detection algorithm.	The intensity detection algorithm was designed from first principles using a method of tracking the speed a hand moves between two threshold lines.	Completed Section 3.4.4
Design of a keyboard note generation system	The keyboard note generation system was designed from first principles by generating a waveform for each possible note in the layout. An ADSR envelope was applied to each waveform to produce a natural-sounding tone.	Completed Section 3.4.5

Table 1.
Design Summary.

3.2 System block diagram

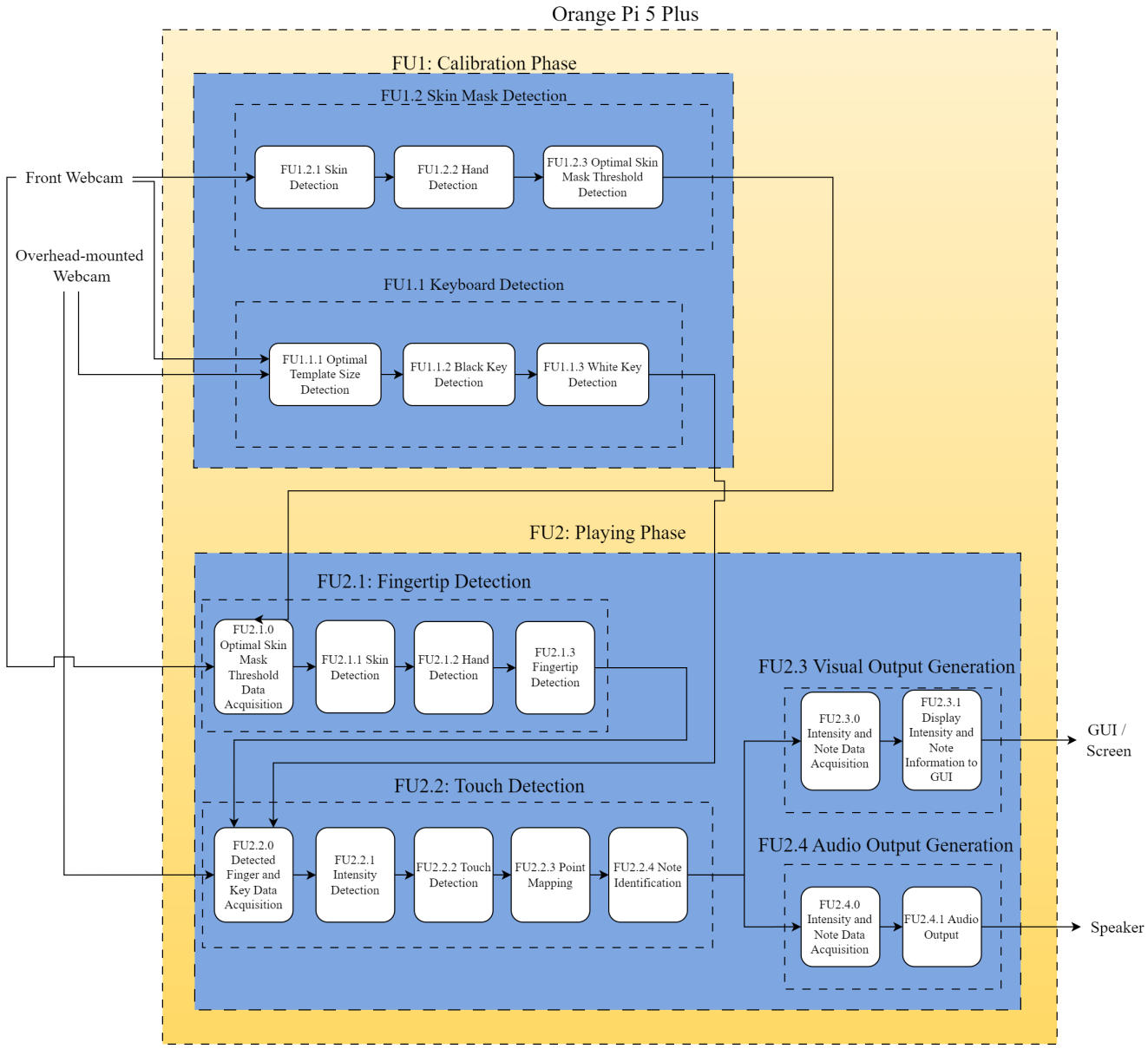


Figure 1.
System block diagram.

Figure 1 depicts the system block diagram of the system. The inputs to the system were the two webcams, which played roles in the calibration and playing phase (FU1 and FU2 respectively). The calibration phase used both webcams to detect the optimal template size in their respective frames (FU1.1.1). The frame of the overhead-mounted webcam was then used to determine the regions of the black and white keys in the layout (FU1.1.2 and FU1.1.3 respectively). The frame of the front webcam was used to identify the optimal skin mask threshold values (FU1.2) that would be used in the playing phase (FU2).

The first step in the playing phase was to detect the fingertips (FU2.1). This was performed by using the optimal skin mask threshold values to apply a skin mask to the frame of the front webcam (FU2.1.1) which would isolate the skin pixels from the rest of the frame. Template matching was then performed to detect the hand and performed once again to detect the fingertips within the region of the detected hand (FU2.1.3). The locations of the detected fingertips and the regions of the keys were then passed into the touch detection subsystem (FU2.2). In this subsystem, the desired intensity of the touch was calculated using the speed of the detected hand (FU2.2.1). When a touch event was triggered, a perspective transformation would be applied to the touch-detected point, to map the point from the front webcam to the overhead-mounted webcam (FU2.2.3). The mapped point would then be checked if it fell within the regions of the defined keys (FU2.2.4). If a note was identified, the identity of the note and the desired intensity would be sent to the visual and audio output generation subsystems (FU2.3 and FU2.4 respectively). The audio output generation subsystem would play the note at the desired intensity through the speaker (FU2.4.1). The visual output generation subsystem would display the detected note and the intensity of the touch on the GUI (FU2.3.1).

3.3 Hardware Implementation

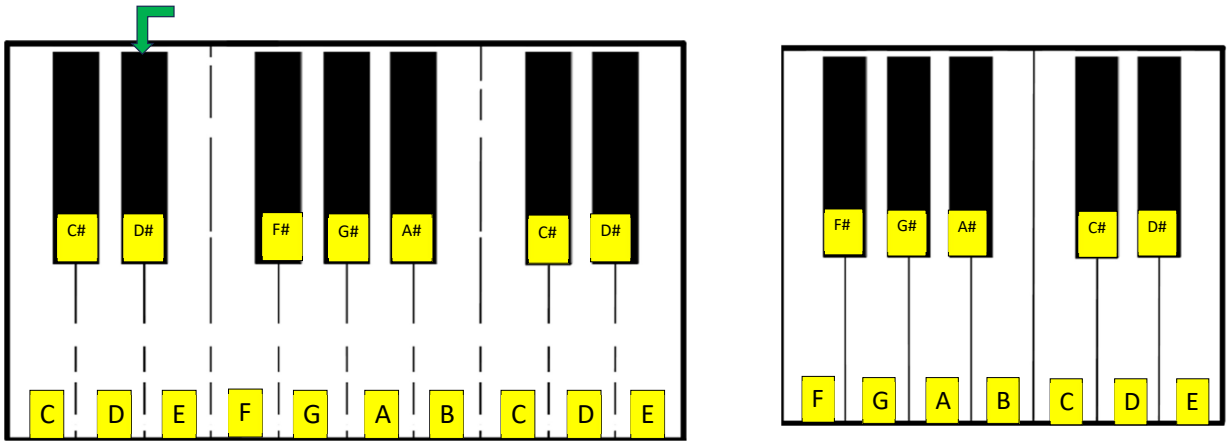
3.3.1 Hardware Components

The single-board computer that was used in the system was the Orange Pi 5 Plus due to its powerful processing capabilities. This board was equipped with an 8-core processor which was beneficial to the system as real-time computer vision and image processing tasks are computationally expensive and require much processing power. A second reason that this SBC was chosen was due to its support of multiple USB ports. This board consisted of separate USB 3.0 and 2.0 sockets which was important as having to use two webcams simultaneously requires the ability to provide sufficient bandwidth to each webcam. The system also used a ring light to provide sufficient light to the webcams to detect the keyboard layout and the user's fingers. The system consisted of two webcams with the overhead webcam being the Hikvision DS-U12, which was selected since it provided 1080p resolution at a frame rate of 30 frames per second. This webcam was fully compatible with the Linux operating system that the Orange Pi 5 Plus was running. Lastly, this webcam was designed to have the ability to be mounted on a tripod, which made it essential for the system as it could be mounted without difficulty on a third-party overhead mount.

3.3.2 Keyboard Layout

A standard keyboard layout that consisted of 17 piano keys was used. This layout consisted of 10 white keys and 7 black keys. The entire keyboard layout was not chosen as it would not fit on a single piece of A4 paper and instead would require multiple pieces of paper. A single webcam would also be unable to capture the entire layout at the distance from the layout required to detect fingers. It would also be impractical to reduce the size of the keyboard layout too much as this would increase the difficulty of touching the desired key which would resultantly reduce the accuracy of the system if fingers easily overlapped keys. Using a compact layout with fewer keys allowed the system to be more accessible to users as it simplified the learning curve by focusing on a smaller set of notes. A paper layout was chosen instead of a projected layout as using a projector would increase the cost and the complexity of the system.

Figure 2a represents the 17 keys layout used in the system. The layout was altered from the standard piano layout by separating the black keys from the edge of the design as represented by the green arrow. This was done to aid in the algorithm that detects the regions of the black keys within the layout. The system was designed to allow the user to play with more notes by adding the expansion layout that is seen in Figure 2b. Therefore, the system allowed the user to create music using 29 notes with the expansion layout.



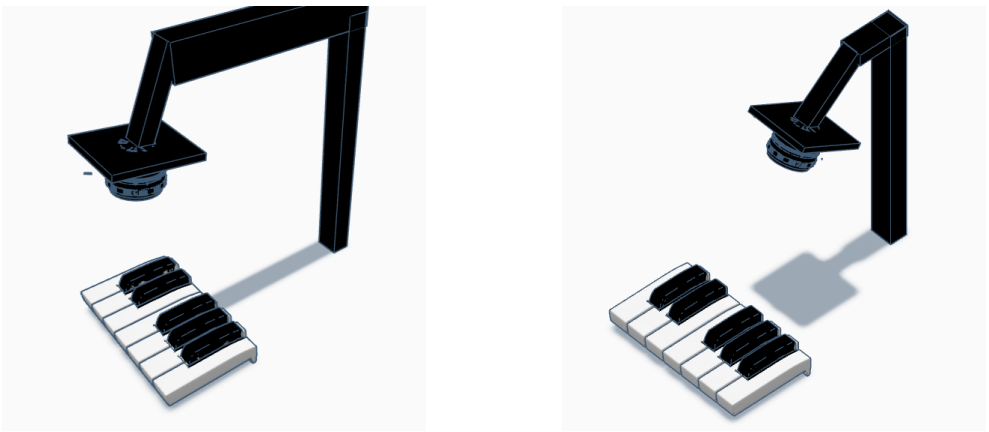
(a)
17 Keys Layout.

(b)
Expansion Layout.

Figure 2.
Keyboard layout used in the system.[1]

3.3.3 Hardware Experimentation

An experiment was performed to identify the ideal location of the overhead-mounted webcam. The primary purpose of this webcam was to provide a frame that allows the system to identify the location of the keyboard.



(a)
Webcam mounted 90° above the layout.

(b)
Webcam mounted < 90° above the layout.

Figure 3.
Overhead webcam positions.

[1]The note labels shown here are for reference purposes only and do not appear on the actual physical keys.

Figure 3 represents a simulation of how the experiment was undertaken to identify the ideal position of the overhead-mounted webcam. The webcam was first mounted directly above the layout at 90° as depicted in Figure 3a, and the performance of the keyboard detection algorithm was recorded. The performance was recorded again when the webcam was placed at positions less than 90° as depicted in Figure 3b. It was found that the system was unable to detect the keys when placing the overhead-mounted webcam lower than 90% due to the non-orthogonal view of the layout.

3.4 Software Design and Implementation

The computer vision approach of the system requires frames to be read from the webcams and processed accordingly for the specific algorithms. The majority of the system was designed in the Python language, due to the benefit of being able to rapidly create prototypes of various subsystems. Python also offers robust interoperability with lower-level languages such as C through libraries like Cython which was used for computationally expensive algorithms of the system. The OpenCV library was used to set the resolution of the webcams and read frames from them. The Scipy library was used to convert the image and template into the frequency domain to perform elementwise multiplication in the frequency domain. This library was also used to write the audio files that were generated in the Note Detection subsystem. The Pygame library was used to load the saved audio files and play out the notes through the speaker. The system made use of the Tkinter library to produce the GUI of the system.

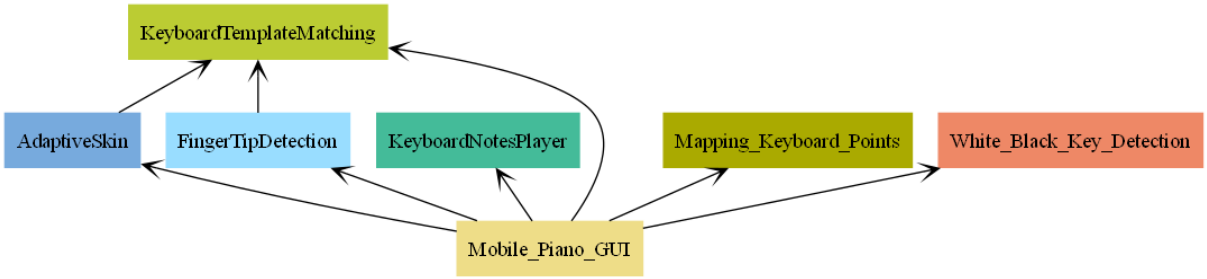


Figure 4.
Packages diagram of the system.

Figure 4 shows how the different modules of the system interact. The Mobile_Piano_GUI module consists of the GUI of the system and calls functions from each of the modules. This module reads the webcam frames and sends them to the respective modules for it to be processed.

3.4.1 Key Detection

The key detection algorithm was developed by first identifying the keyboard's location in the frame of both webcams. The frame of the overhead-mounted webcam was first read and processed. The system detected the keyboard region only in the calibration stage as it was assumed that the keyboard's location was not changed during the system's running. This approach was chosen as performing real-time detection for the keyboard and the user's fingertips would be computationally expensive, slowing down the system's computational speed.

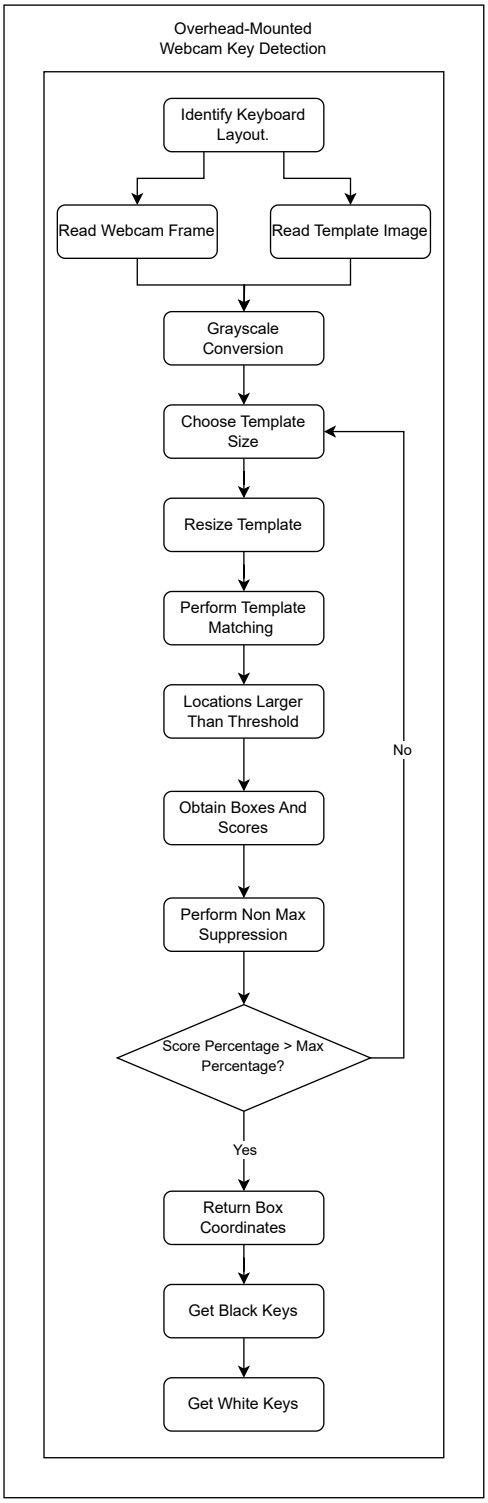


Figure 5.
Software flow of key detection from the overhead-mounted webcam.

Figure 5 depicts the process of the key detection algorithm using the overhead-mounted webcam as the input. The process involved identifying which keyboard layout the user chose and reading the specific template of that layout. The OpenCV library was used to read the template image. The template image and the frame from the webcam were then converted from the BGR colourspace

to the grayscale colour space. This conversion was performed as grayscale images consist of only a single channel, whereas the BGR colourspace consists of three channels and each channel has to be processed individually. Therefore, the conversion to the grayscale colourspace reduced the complexity and processing time. The conversion from BGR to grayscale is

$$Y = 0.2989 \cdot R + 0.5870 \cdot G + 0.1140 \cdot B \tag{1}$$

where Y represents the grayscale channel, while R, G and B represent the red, green and blue channels respectively. The next step in the key detection process was template matching. An adaptive size matching method which combined template matching and bilinear interpolation, was designed to adapt to different sizes of the keyboard layout. Since the overhead-mounted webcam could be moved lower or higher depending on the user's need, the keyboard layout had to be detected at different heights. This allowed the user to not have to set the overhead-mounted webcam at a set height and provided flexibility to use the system in places where space was limited. The method consists of defining a set of different sizes of the template image where

$$[0.55, 0.6, 0.65, 0.7, 0.75, 0.8, 0.85, 0.9] \tag{2}$$

were the resizing scales chosen for the standard keyboard layout and

$$[0.85, 0.88, 0.91, 0.93, 0.95] \tag{3}$$

were for the expanded keyboard layout. For each scale factor depicted in the above arrays, the template would be resized accordingly. The resizing method made use of bilinear interpolation which operates by computing the interpolated value of a point using the four nearest pixels around that point [17]. The first step in the resizing algorithm was to take each pixel in the target image and compute the corresponding position in the original image where the x-coordinate was calculated as

$$x = x' \cdot \text{scale factor} \tag{4}$$

and the y-coordinate as

$$y = y' \cdot \text{scale factor.} \tag{5}$$

x' and y' represent the x and y coordinates of a pixel of the target image. The result of these coordinates may not be an integer, therefore, the four nearest pixel positions were obtained as follows where I represents the original image. The equations to compute the top-left and top-right corners are

$$\text{Top-left corner: } I(x_1, y_1), \quad \text{where } x_1 = \lfloor x \rfloor \text{ and } y_1 = \lfloor y \rfloor \tag{6}$$

and

$$\text{Top-right corner: } I(x_1 + 1, y_1), \quad \text{where } x_1 = \lfloor x \rfloor \text{ and } y_1 = \lfloor y \rfloor \tag{7}$$

respectively, while the equations to compute the bottom-left and bottom-right corners are

$$\text{Bottom-left corner: } I(x_1, y_1 + 1), \quad \text{where } x_1 = \lfloor x \rfloor \text{ and } y_1 = \lfloor y \rfloor \tag{8}$$

and

$$\text{Bottom-right corner: } I(x_1 + 1, y_1 + 1), \quad \text{where } x_1 = \lfloor x \rfloor \text{ and } y_1 = \lfloor y \rfloor. \tag{9}$$

The symbols $\lfloor x \rfloor$ and $\lfloor y \rfloor$ represent the floored x and y values respectively. The distances to these neighbours were then computed followed by the interpolated value. The equations to calculate the x and y of the distances are

$$\text{dx} = x - x_1, \text{where } dx \text{ represents the distance in the x coordinate} \tag{10}$$

and

$$\text{dy} = y - y_1, \text{where } dy \text{ represents the distance in the y coordinate.} \tag{11}$$

The equation to calculate the interpolated value is then

$$\begin{aligned} \text{Interpolated Value} = {} & (1-\text{dx})\cdot(1-\text{dy})\cdot I(x_1,y_1) \\ & + \text{dx}\cdot(1-\text{dy})\cdot I(x_1+1,y_1) \\ & + (1-\text{dx})\cdot\text{dy}\cdot I(x_1,y_1+1) \\ & + \text{dx}\cdot\text{dy}\cdot I(x_1+1,y_1+1). \end{aligned} \tag{12}$$

The resized template was then used in the template matching method. Template matching is a computer vision method used to locate specific areas within an image that resemble a given template. This method uses the NCC metric to measure the similarity between a template and regions within an image. The assumption was made that the orientation of the layout was kept upright, and therefore, the system was unable to detect the layout if this assumption was not met. An approach that can search for template occurrences regardless of their orientation is known as Greyscale-based Template Matching [18]. This approach was not chosen as it would be computationally expensive to test for various orientations as well as adapt to different template sizes. Figure 6 represents the software flow diagram for the FFT-based template matching method.

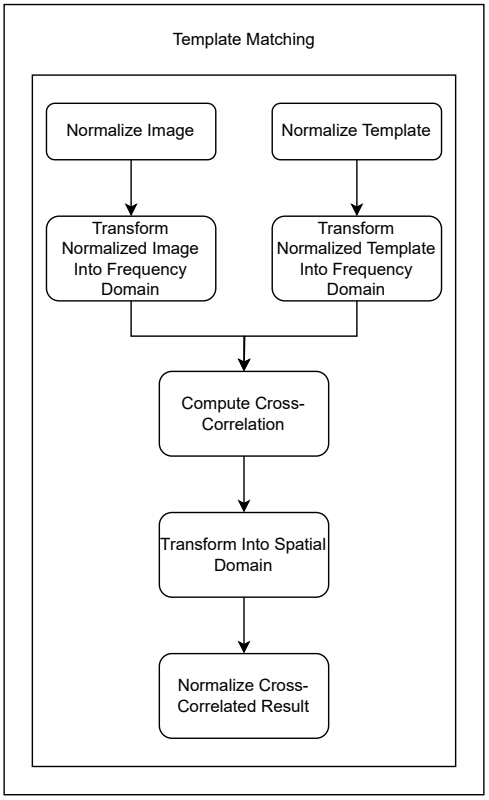


Figure 6.
Software Flow of Template Matching.

The first step in the template matching method was normalization. This was performed to improve the stability and effectiveness of the template matching method. This step made the image and template less sensitive to variations in lighting as pixel values were standardized, and therefore, the matching process focused on relative patterns instead of absolute pixel intensities.

The equation to normalize the resized template is

$$\text{normalized\_image}(x,y) = \frac{\text{image}(x,y) - \mu_{\text{image}}}{\sigma_{\text{image}} + \varepsilon} \tag{13}$$

and the equation to normalize the webcam frame is

$$\text{normalized\_template}(x,y) = \frac{\text{template}(x,y) - \mu_{\text{template}}}{\sigma_{\text{template}} + \varepsilon.} \tag{14}$$

The variables μ_{image} and μ_{template} are the means of the image and template, respectively, while σ_{image} and σ_{template} represent their standard deviations. The small constant, ε, was added to avoid division by zero. This constant only plays a major role when the image is uniform, such as the binarized frame that is normalized during the finger detection phase, as no skin pixels in the image cause it to be uniform. The next step of template matching was to transform both the image and the template into the frequency domain as performing cross-correlation in this domain was simple since it was element-wise multiplication that occurred. The 2D FFT for both the image and the template can be mathematically defined using equations

$$\hat{I}(k,l) = \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} \text{normalized\_image}(x,y) \cdot e^{-j2\pi\left(\frac{kx}{M} + \frac{ly}{N}\right)} \tag{15}$$

and

$$\hat{T}(k,l) = \sum_{x=0}^{P-1} \sum_{y=0}^{Q-1} \text{normalized\_template}(x,y) \cdot e^{-j2\pi\left(\frac{kx}{M} + \frac{ly}{N}\right).} \tag{16}$$

$\hat{I}(k,l)$ and $\hat{T}(k,l)$ are the Fourier Transforms of the image and template, respectively, while M and N represent their width and height. The Scipy library was used to convert both the image and template into the frequency domain using the fft2 function. Cross-correlation was then performed by multiplying the complex conjugate of $\hat{T}(k,l)$ with $\hat{I}(k,l)$ using the equation,

$$\hat{C}(k,l) = \hat{I}(k,l) \cdot \overline{\hat{T}(k,l).} \tag{17}$$

$\hat{C}(k,l)$ represents the cross-correlated result in the frequency domain, while $\overline{\hat{T}(k,l)}$ represents the complex-conjugate of $\hat{T}(k,l)$. The cross-correlated result then needed to be transformed into the spatial domain by applying the inverse fast Fourier Transform to it as follows:

$$C(x,y) = \frac{1}{MN} \sum_{k=0}^{M-1} \sum_{l=0}^{N-1} \hat{C}(k,l) \cdot e^{j2\pi\left(\frac{kx}{M} + \frac{ly}{N}\right).} \tag{18}$$

$C(x,y)$ represents the cross-correlated result in the spatial domain. This result was then normalized to standardize the cross-correlated values using the equation,

$$\text{correlation}(x,y) = \frac{C(x,y) - \mu_{\text{image}}}{N \cdot \sigma_{\text{image}} \cdot \sigma_{\text{template}}}. \tag{19}$$

μ_{image} represents the mean of the raw image, while σ_{image} and σ_{template} represent the standard deviations of the image and template. N denotes the number of pixels in the template. The normalized cross-correlated values could be interpreted as follows:

The output of the template matching result is depicted in Figure 22 within Section 3.4.2. The locations where the normalized cross-correlated values were above a set threshold of 0.6 were found. The bounding boxes and confidence scores corresponding to these locations were then extracted and non-maximum suppression was taken to ensure that only a single box was returned. NMS is a technique commonly used in object detection to discard redundant bounding boxes by

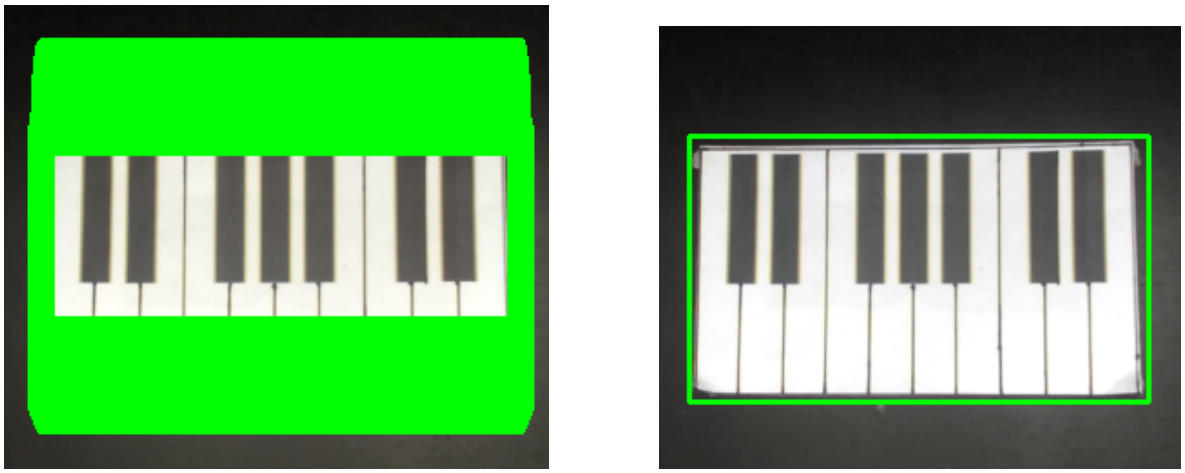
Correlation Value	Interpretation
Near 1	High positive correlation, indicating a strong match between the image and template.
Near 0	No correlation, indicating no similarity between the image and template.
Near -1	High negative correlation, indicating an inverse match or opposite pattern between the image and template.

Table 2.
Interpretation of Normalized Cross-Correlation Values.

retaining the bounding box with the highest confidence score. The first step of NMS was to sort the bounding boxes based on the confidence scores in descending order. The first bounding box was then selected and its IoU with the rest of the boxes was computed. The boxes that have a greater IoU than a threshold of 0.1 was discarded. The equation for computing the IoU between two boxes is as follows:

$$\text{IoU}(B_i, B_j) = \frac{\text{Area}(B_i \cap B_j)}{\text{Area}(B_i) + \text{Area}(B_j) - \text{Area}(B_i \cap B_j)} \tag{20}$$

B_i represents the bounding box of the first box, while B_j represents the bounding box of the second box.



(a)
NMS not applied.

(b)
NMS applied.

Figure 7.
Effect of NMS after template matching.

Figure 7b depicts how NMS discards redundant bounding boxes and returns only a single box. When looking at Figure 5 it can be seen that the process of choosing the template and then performing template matching, followed by NMS was repeated until the template size that provided the highest confidence score was achieved. The final bounding box is obtained and the next step in the key detection algorithm was detecting the regions of the black keys. The image was segmented to contain only the contents of the bounding box and passed into the black key algorithm. The algorithm consisted of converting the BGR frame into the HSV colour space.

This conversion was performed as the value channel captures the intensity of the pixels, allowing it to be less sensitive to changes that occur within the hue or saturation channels. The value channel then went under a binarization technique known as Otsu's thresholding. The purpose of this technique is to find the optimal threshold value that separates the pixels of an image into foreground and background pixels. The first step of Otsu's thresholding was to compute the histogram of pixel intensities for the value channel image. $hist[i]$ will represent the number of pixels with intensity i, where i ranged from 0 to 255 for the 8-bit image. The next step was to calculate the total sum of pixel intensities across the entire value channel,

$$\text{sum\_pixel\_intensities} = \sum_{i=0}^{255} i \times \text{hist}[i] \tag{21}$$

An iterative process was then followed where for each intensity level t, the weights and means for the background and foreground classes were calculated. The assumption was made that pixels below t were considered background pixels, while those above were considered foreground pixels. The weights for the foreground and background classes were calculated

$$\text{weight\_background}(t) = \sum_{i=0}^{t} \text{hist}[i] \tag{22}$$

and

$$\text{weight\_foreground}(t) = \text{total\_pixels} - \text{weight\_background}(t) \tag{23}$$

respectively. The mean for the background class was calculated as

$$\text{mean\_background}(t) = \frac{\sum_{i=0}^{t} i \times \text{hist}[i]}{\text{weight\_background}(t)} \tag{24}$$

and the mean for the foreground class as

$$\text{mean\_foreground}(t) = \frac{\text{sum\_total} - \sum_{i=0}^{t} i \times \text{hist}[i]}{\text{weight\_foreground}(t)} \tag{25}$$

The between-class variance was then calculated which measured the separability of the background and foreground classes,

$$\begin{aligned} \sigma_b^2(t) = {} & \text{weight\_background}(t) \times \text{weight\_foreground}(t) \\ & \times (\text{mean\_background}(t) - \text{mean\_foreground}(t))^2. \end{aligned} \tag{26}$$

The optimal threshold, t was then considered to be the value that maximizes the between-class variance,

$$\text{optimal\_threshold} = \arg\max_t \sigma_b^2(t) \tag{27}$$

This optimal threshold value, t, was then used to threshold the image, where pixel intensities larger or equal to t were set to 255, and those below were set to 0. Figure 8 depicts the Otsu's thresholded image for the detected keyboard layout.

Figure 8.
Otsu's thresholded keyboard layout.

The Otsu's thresholded image was then inverted and a morphological operation of erosion was applied to the inverted result. The image was inverted to aid in the algorithm that detects the regions of the black keys. Erosion was applied to remove the vertical black lines that can be seen in Figure 8. The process of erosion involves using a structural element such as a kernel or binary mask, to examine each pixel in the image. This structural element defines the neighborhood for the erosion process and a 3x3 kernel of ones was chosen to be used over the Otsu's thresholded image. This kernel was placed over each pixel, and if all the pixels under the kernel contained the value 1, the centre pixel would not change, however, if a single pixel under the kernel contained the value 0, the centre pixel would be changed to 0. This process shrunk out the lines that are seen in Figure 8. A kernel size larger than 3x3 was not chosen as this would shrink out the black key regions as well. Erosion can be mathematically expressed as,

$$(I \ominus K)(x,y) = \begin{cases} 1 & \text{if every } I(x+i, y+j) = 1 \text{ for } (i,j) \in K \\ 0 & \text{otherwise} \end{cases} \tag{28}$$

, where I represents the binary image, K represents the structuring element and x and y represent the positions of each pixel.

(a) **Erosion using a 3x3 kernel.**

(b) **Erosion using a 5x5 kernel.**

(c) **Erosion using a 7x7 kernel.**

Figure 9.
Eroded Layouts Using Different Kernel Sizes.

Figure 9 depicts the effect of the kernel size used in the erosion process, where kernel sizes of 5x5 or 7x7 significantly shrunk the black keys. This was not desired as the detected regions of the black keys would be smaller and the accuracy of detecting a touch would be reduced. Figure 9a depicts the desired result where the vertical lines could be removed while ensuring that the

black key regions were not significantly shrunk.

The final step of the black key algorithm was to compute the boundaries of each rectangle present in the eroded image. To do this, the connected-component labelling method had to be implemented. Each pixel in the image was checked to identify if it was part of a rectangle, where pixels with the value of 255, were considered to be part of a rectangle. Adjacent white pixels are grouped by performing a flood-fill where for each pixel at coordinates (x,y), its set of four neighbours is defined as:

$$N_4(x,y) = \{(x+1,y),(x-1,y),(x,y+1),(x,y-1)\} \tag{29}$$

where $N_4(x,y)$ represents the set of neighbours at (x,y). The flood-fill algorithm assigns a unique label L to the pixel and all the neighbours in that set using the following label propagation rule:

$$L(x,y) = \begin{cases} L & \text{if } I(x,y) = 255 \text{ and } L(x,y) = 0 \\ 0 & \text{otherwise} \end{cases} \tag{30}$$

This rule checks if the current pixel has the intensity of a white pixel and has not been labelled yet, if this condition is met, then assign it a label. A label was also assigned to the pixels that were within rectangles, where each rectangle was assigned a different label. The flood fill algorithm made use of a stack to keep track of all the neighbouring pixels where only 4 neighbour pixels were considered. This is known as the 4-connected flood fill algorithm where only 4 directions are examined for each pixel, being up, down, left and right. If diagonal neighbours were considered, it would be known as the 8-connected flood fill algorithm. At the end of the flood fill algorithm, all the pixels in a rectangle would be identified and the next step would be to identify the boundary pixels for each rectangle. A pixel was considered to be a boundary pixel if one of its neighbours contained the value 0. Once all the boundary pixels were identified, the bounding box for the specific rectangle could be obtained.

Figure 10.
Boundaries of black keys.

Figure 10 depicts the boundaries of the black keys that were detected after the 4-connected flood fill algorithm. Algorithm 1 seen below, shows the algorithm used to obtain the regions of the black keys from the frame. It is seen that on line 9, after erosion, a mask operation was applied. This was performed to remove any white pixels that could show up due to the lighting. If these white pixels remained, then the algorithm would not be able to identify the rectangles. Line 10 sorts the rectangles by the x-coordinates to keep them in order of the notes.

Algorithm 1 Identify Black Keys

Require: frame: An image frame in BGR colour space

```
function IDENTIFYBLACKKEYS(frame)
hsvImage ← ConvertToHSV(frame) {Convert to HSV color space}
valueChannel ← ExtractValueChannel(hsvImage) {Extract V channel}
binaryImage ← OtsuThreshold(valueChannel) {Apply Otsu's thresholding}
invertedImage ← Invert(binaryImage) {Invert the binary image}
kernel ← CreateKernel(3) {Create a 3x3 kernel for erosion}
erodedImage ← Erode(invertedImage, kernel) {Apply erosion}
MaskIrrelevantRegions(erodedImage) {Mask out irrelevant regions from the image}
(boundaryImage, rectangleCoords, numRectangles) ← FindRectangles(erodedImage)
if numRectangles = 7 or numRectangles = 12 then
    return SortByXCoordinate(rectangleCoords) {Return sorted coordinates of black keys}
else
    return None {Return None if number of rectangles is invalid}
end if
end function=0
```

The next step was to compute the regions of the white keys. This was performed by separating the layout horizontally into two halves. The base of the first black box was used as the separation line, where the region below this line was considered to be for the white keys, while the region above was considered to be for the black keys. The white key region was then vertically split based on the number of black keys detected in the layout. The standard layout which consisted of 7 black keys, would have 10 white keys, while the expanded layout which consisted of 12 black keys, would have 17 white keys. This method to obtain the white keys was computationally inexpensive, however, the drawback was that regions above the separation line would not be considered as part of white keys, and instead were considered as black keys to aid in the detection of the touch of a black key, which is explained further in section 3.4.3. Figure 11 depicts the result of the key detection algorithm showing where the white and black keys had been detected.

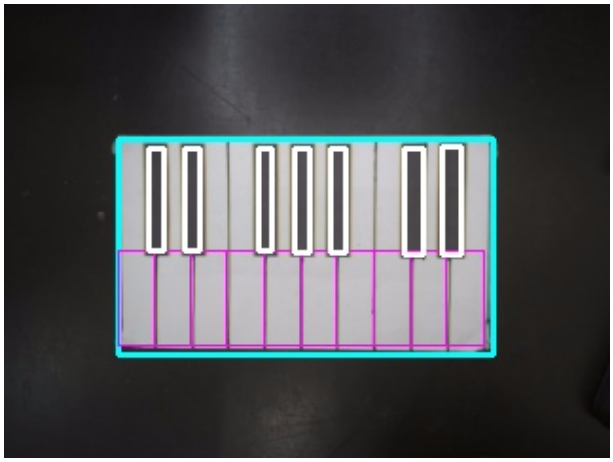
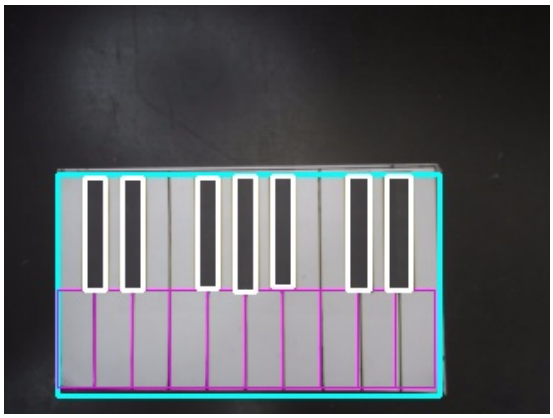


Figure 11.
Detected key regions.

(a)
Optimal template scale of 0.45.



(b)
Optimal template scale of 0.85.

Figure 12.
Adaptive size matching.

Figure 12 depicts the result of the adaptive size matching method designed to detect the keyboard at different heights using the overhead-mounted webcam. The system chose to resize the template to 0.45 of the original size as seen in Figure 12a, while in Figure 12b, the system resized the template to 0.85 of the original size. It can be seen that the system successfully detected the keyboard layout and its keys to varying heights by using the optimal template size for the template matching method.

Once the keys were detected using the overhead-mounted webcam, the system had to detect where the keyboard layout was on the surface from the view of the front webcam. The view from the front webcam differed from that of the overhead-mounted webcam as the keyboard layout now had a trapezoidal shape. Therefore, a trapezoidal template was made for the keyboard layout as seen below in Figure 13. The templates were produced by taking a picture of the layouts and selecting the four corners of the layout seen in the image.

(a)
Standard Layout.

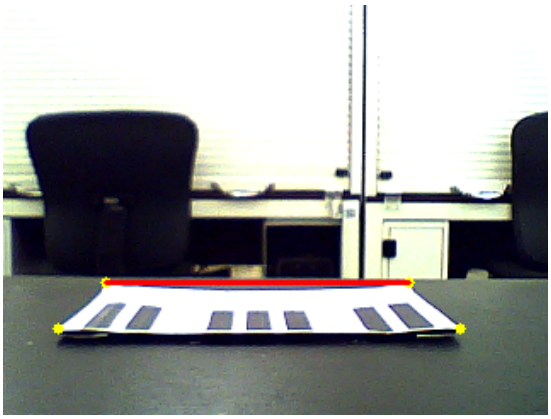
(b)
Expanded Layout.

Figure 13.
Keyboard templates for the front webcam.

Figure 13a depicts the template used for the standard layout seen on the front webcam. Both layouts have a trapezoidal shape surrounded by a black padded background. The process of obtaining the region with the keyboard layout in the front webcam's frame was very similar to that of the overhead-mounted webcam. The flow of the process is depicted below in Figure 15. The adaptive size matching method was also applied to the frame of the front webcam to provide the user with the flexibility of positioning the layout. The method of obtaining the keyboard layout differs from that shown in Figure 5 as the regions of the black and white keys are not

obtained, instead, the corners of the layout are detected from the segmented region found after template matching. The image was then binarized using Otsu's thresholding method as seen below in Figure 14a.

(a)
Otsu's thesholded keyboard segment.



(b)
Keyboard layout corners detected from the front webcam frame.

Figure 14.
Front webcam keyboard layout detection.

The top left corner was detected by taking the very first white pixel found in Otsu's thresholded image, while the top right corner would be the last white pixel in the first row of that image. Similarly, the bottom corners were detected as the first and last white pixels at the bottom. To ensure that the corners were correctly detected, the user would have to position the layout that allows the top corners of the thresholded image to be the first and last pixels for the first row of the image. Figure 14b depicts the four corners recognized by the yellow markings. The detection of the expanded layouts has been depicted in Figures 41 and 42 in Section 6.4.

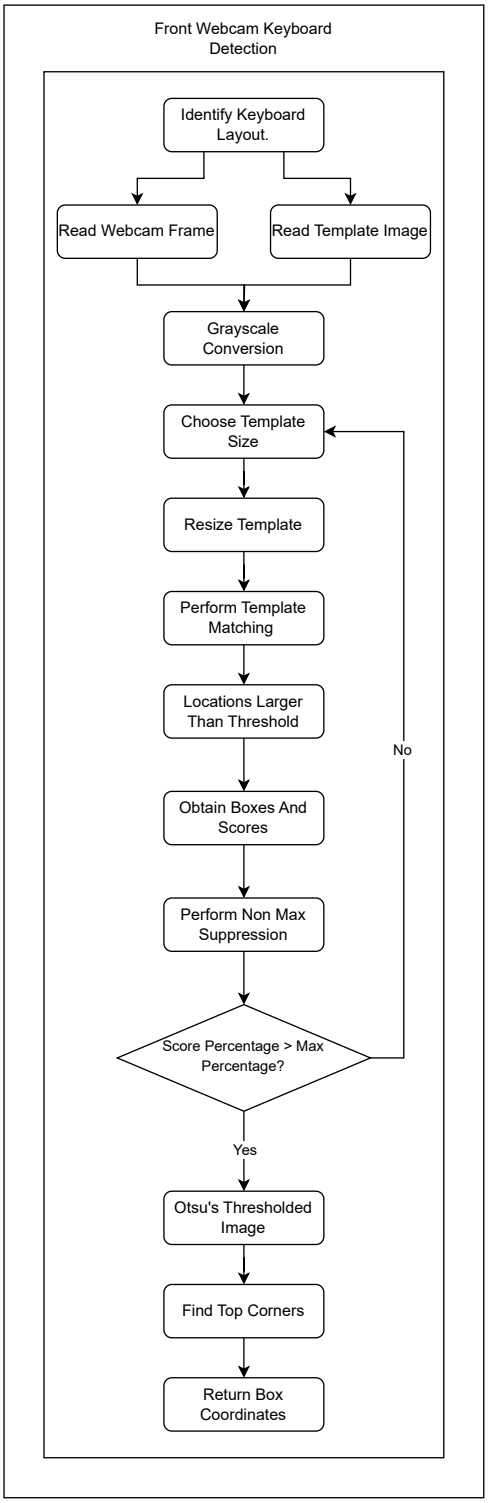


Figure 15.
Software flow of key detection from the front webcam.

3.4.2 Finger Detection

The system had to detect the user's fingers through the front webcam to allow the user to play the keyboard. The method used to detect fingers was a combination of a skin mask and template matching. The purpose of the skin mask was to detect skin pixels in an image and present them as white pixels, while non-skin pixels would be represented as black pixels. Figure 16 depicts the software flow of the finger detection algorithm.

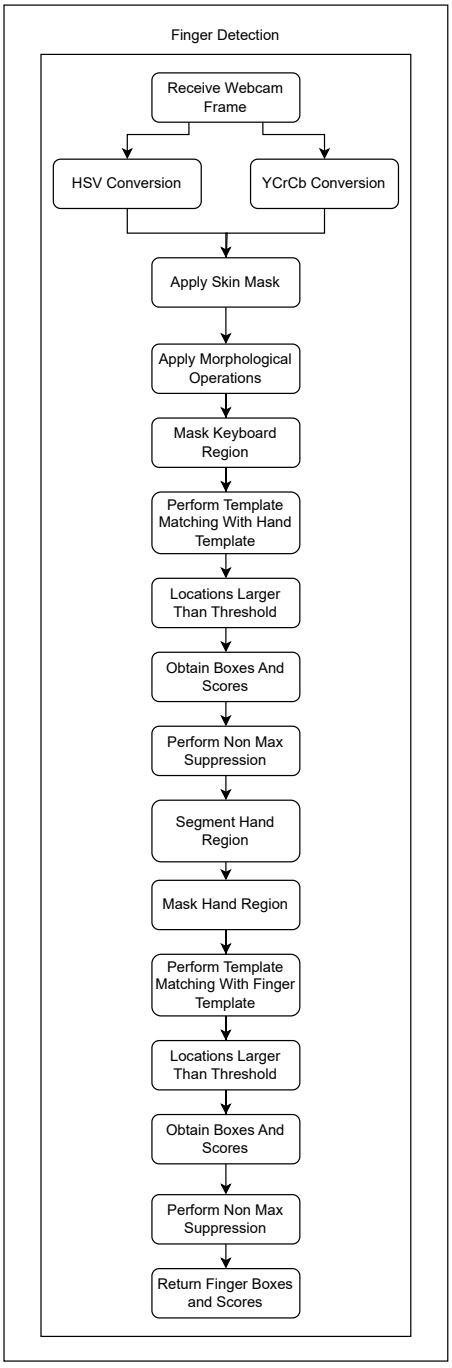


Figure 16.
Software flow of the finger detection algorithm.

The webcam frame was processed by converting the RGB frame into two colour spaces, the HSV colour space, and the YCrCb colour space. These colour spaces were significant as the skin mask combines the saturation channel, S, with the chroma channels, Cr and Cb. For both colour space conversions, the RGB channels had to be normalized using this equation,

$$R' = \frac{R}{255}, \quad G' = \frac{G}{255}, \quad B' = \frac{B}{255}. \tag{31}$$

Regarding the conversion to the HSV colour space, C_{max}, C_{min} and Δ had to be computed using equations 32 to 34. The value channel, V was obtained from C_{max} as $V = C_{max}$. Δ indicates the intensity of the hue as it is known as chroma.

$$C_{max} = \max(R', G', B') \tag{32}$$

$$C_{min} = \min(R', G', B' \tag{33}$$

$$\Delta = C_{max} - C_{min} \tag{34}$$

The hue channel, H, represents the dominant wavelength and was computed using equation 35.

$$H = \begin{cases} 0, & \text{if } \Delta = 0 \\ 60° \times \left(\frac{G'-B'}{\Delta} \mod 6 \right), & \text{if } C_{max} = R' \\ 60° \times \left(\frac{B'-R'}{\Delta} + 2 \right), & \text{if } C_{max} = G' \\ 60° \times \left(\frac{R'-G'}{\Delta} + 4 \right), & \text{if } C_{max} = B' \end{cases} \tag{35}$$

The saturation channel, S, was computed using equation 36.

$$S = \begin{cases} 0, & \text{if } C_{max} = 0 \\ \frac{\Delta}{C_{max}}, & \text{otherwise} \end{cases} \tag{36}$$

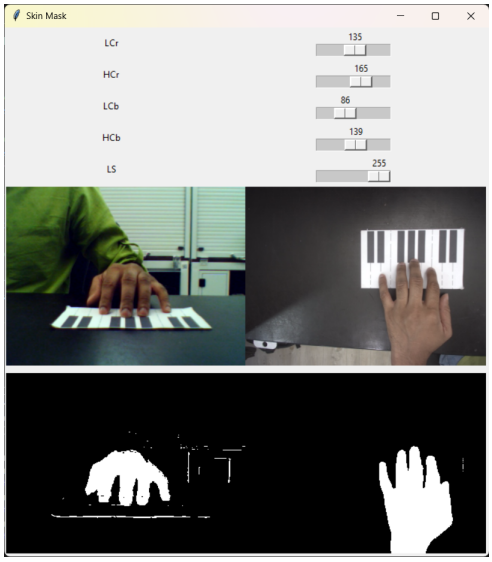
Equation 37 represents the conversion from the RBG colour space to the YCrCb colour space.

$$\begin{bmatrix} Y \\ Cr \\ Cb \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ -0.168736 & -0.331264 & 0.5 \\ 0.5 & -0.418688 & -0.081312 \end{bmatrix} \begin{bmatrix} R' \\ G' \\ B' \end{bmatrix} + \begin{bmatrix} 0 \\ 0.5 \\ 0.5 \end{bmatrix} \tag{37}$$

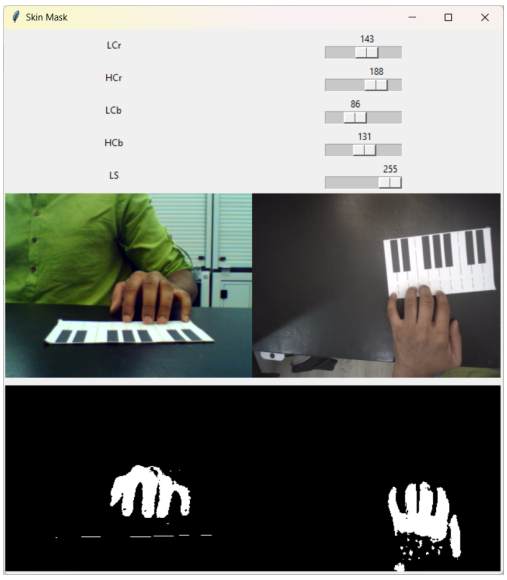
The skin mask was then created by making use of the S, Cb and Cr channels as seen in equation 38,

$$\text{mask} = (V_0 \leq Cr \leq V_1) \wedge (V_2 \leq Cb \leq V_3) \wedge (S \leq V_4) \tag{38}$$

where V_0 and V_1 represent the minimum and maximum threshold of the Cr channel, V_2 and V_3 represent the minimum and maximum threshold of the Cb channel and V_4 represents the maximum threshold of the S channel. These threshold values are heavily dependent on light, and therefore, an adaptive skin mask algorithm was developed to allow the system to be calibrated to use the optimal threshold values based on the current lighting space. For different lighting conditions, these threshold values were obtained by manually changing each upper and lower threshold value until a mask that represented skin pixels most accurately was acquired.



(a)
Laboratory lighting.



(b)
Ring light with laboratory lighting.

Figure 17.
Threshold value selection under various lighting conditions.

Figure 17 depicts how the threshold values were changed under various lighting conditions. The ring light used in the system provided 3 different lighting modes which were used alongside the lighting present in the laboratory to obtain different threshold values. The webcam used could also affect the threshold values, and therefore, a second webcam was tested to provide more threshold values. Table 3 depicts the threshold values used in the adaptive skin mask method.

Table 3.
Threshold Values

Lower Cr	Upper Cr	Lower Cb	Upper Cb	Upper S
142	175	75	155	255
158	194	66	114	255
137	255	0	141	255
134	255	0	149	196
128	147	0	125	255
139	255	0	123	255
139	175	0	123	255
134	161	101	121	255

The software flow of the adaptive skin mask algorithm is depicted below in Figure 18. The algorithm applies different threshold values to the skin mask and then performs template matching to detect the hand in the frame. The threshold values that provide the highest match percentage to the template under the current lighting conditions were then chosen to be the optimal values for the skin mask and would be used as the standard values for the finger detection algorithm.

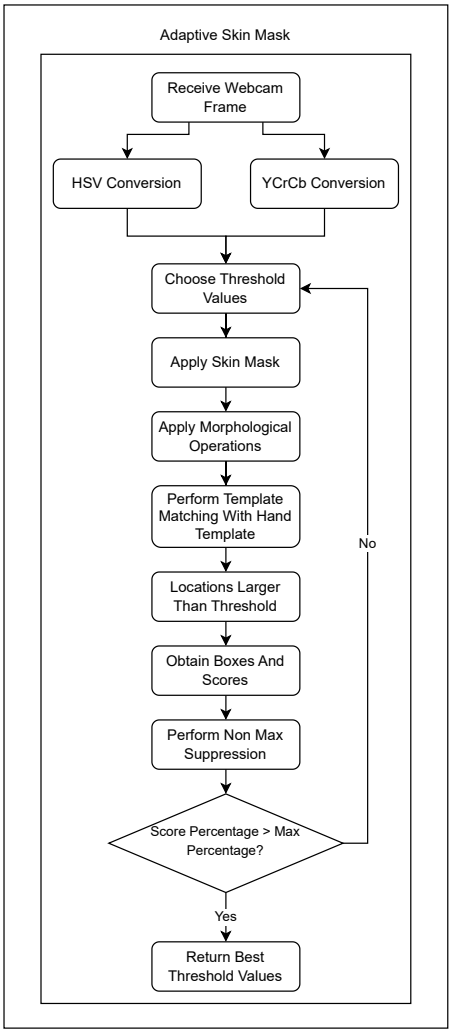


Figure 18.
Software flow of the adaptive skin mask algorithm.

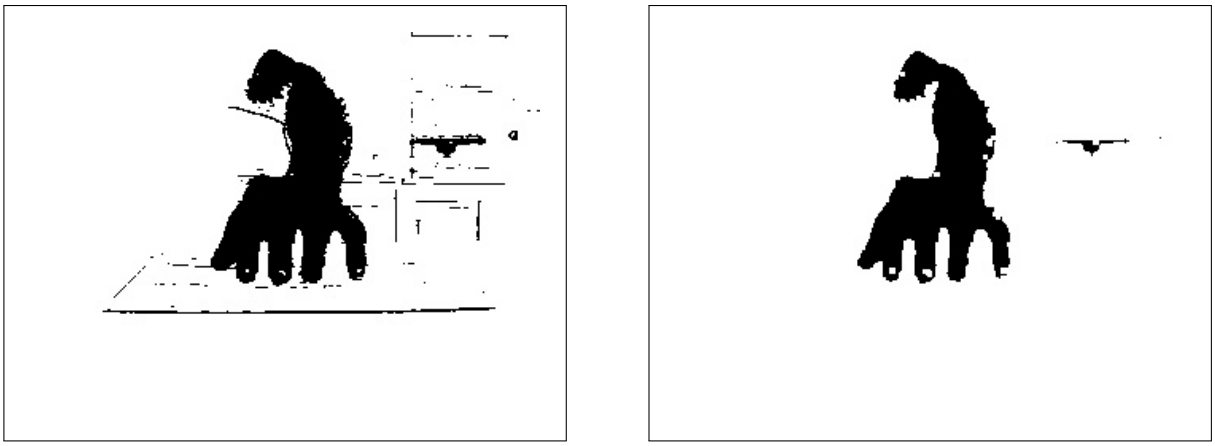
Once the system applied the skin mask using the optimal threshold values, morphological operations were applied to the image, specifically a closing operation followed by a dilation operation. Closing is an image-processing technique used to close small gaps in the foreground of a binary image and is performed by dilating an image, and then eroding it. Dilation expands the foreground in an image by adding pixels to the boundaries of the objects, while erosion removes pixels. Closing can, therefore, be expressed as follows,

$$\text{Closing}(A) = \text{Erosion}(\text{Dilation}(A)) \tag{39}$$

Dilation can be mathematically expressed as follows.

$$(A \oplus B)(x,y) = \max_{(b_x,b_y)\in B} A(x-b_x, y-b_y) \tag{40}$$

A represents the image, while *B* represents the structuring element such as a kernel. Dilation takes the maximum value of the pixels covered by *B*. The purpose of applying a closing operation to the skin mask was to remove the small gaps that could occur within the skin pixels of the hand. The dilation operation performed after the closing operation was to expand the skin-detected region to make the fingertips more prominent. This is depicted in Figure 19. It can be seen that applying morphological operations to the skin-masked image removed most of the background lines that were detected as skin pixels.



(a)
Before morphological operations.

(b)
After morphological operations.

Figure 19.
Effect of morphological operations.

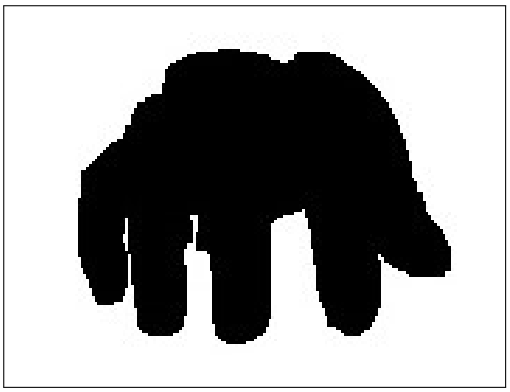
The next step in the finger detection algorithm was to mask the region below the keyboard within the frame as white pixels. This was performed to prevent any reflections of the hand that may be picked up on the surface. Reflections of the fingers could be picked up as possible fingers during the template matching stage and applying the mask prevented such inaccuracies. Figure 20 depicts a green arrow that points to where the white mask was applied. The fingers were placed below the keyboard region resulting in the fingertips being clipped as they overlapped the masked region.

(a)
Webcam frame.

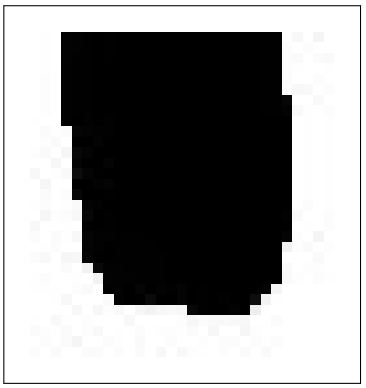
(b)
Masked skin-mask frame.

Figure 20.
Effect of masking.

The finger detection algorithm used template matching to detect the hand and each finger in the frame. The templates used are depicted below in Figure 21.



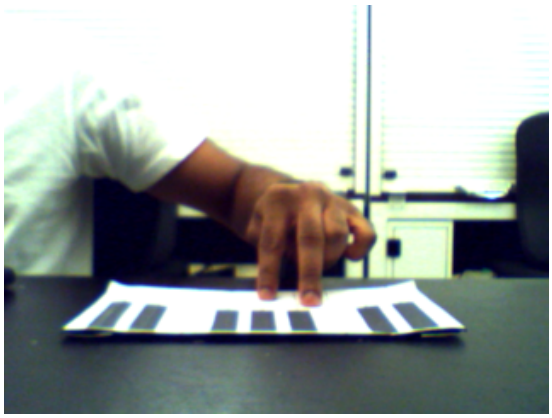
(a)
Hand Template.



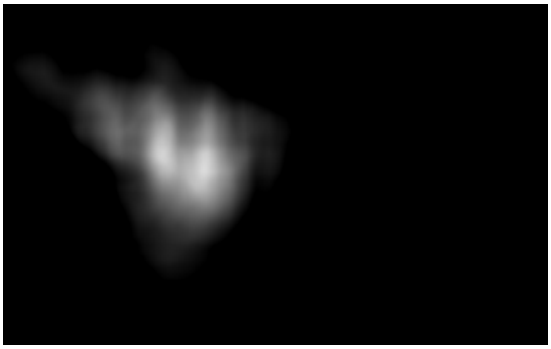
(b)
Finger Template.

Figure 21.
Skin-masked templates.

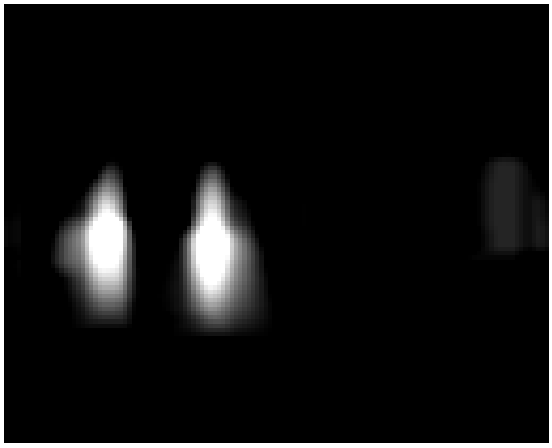
The hand template seen in Figure 21a was used to detect both the right and left hand of the user, and separate templates for each hand were not required since a single template was sufficient to detect two hands in the frame. The finger template was created by cropping a fingertip from the hand template and this template was enough to detect all the fingers on a hand. The finger detection algorithm first detected a hand in the frame and then segmented the region that consisted of the hand. A white mask was then applied to the upper half of the segmented region. This was performed to prevent the template matching algorithm from detecting the top half of the fingers as possible fingertips.



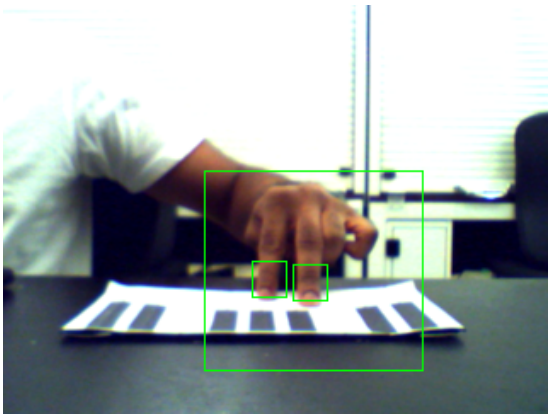
(a)
Original frame without skin detection applied.



(b)
Tempate matching result with the hand template.



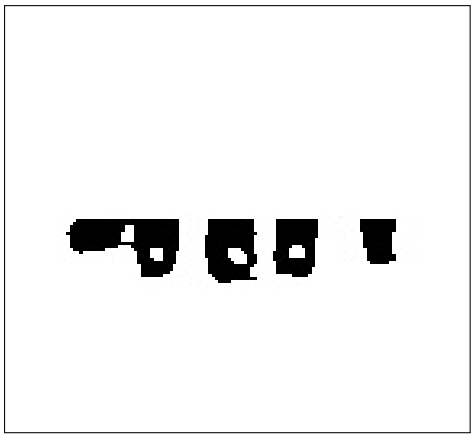
(c)
Template matching result with the finger template.



(d)
Detected fingers after template matching.

Figure 22.
The figure shows the use of template matching to detect fingertips.

Figure 22 depicts how fingertips were detected in a frame using template matching. Figure 22b shows the output of the template matching that occurs between the skin-masked frame and the hand template from Figure 21a. The white regions signify the areas with the highest match, and the outline of the hand can be vaguely seen. Figure 22c depicts how the two fingertips are presented as white regions at the end of template matching. Five fingers could also be detected using this template matching method, however, it required the hand to be positioned in such a way that the thumb was in line with the rest of the fingers.



(a)
Application of white mask.

(b)
Detection of five fingers.

Figure 23.
Positioning of hand to detect five fingers.

Figures 23a and 23b depict how the five fingers are detected. The thumb was the only finger that looked different to the template, however, it was still detected as a finger. The thumb had to be placed in a specific manner to be able to be detected as a finger using the skin-masked template matching method, as generally the thumb is not kept at the same height as the other fingers, and therefore, the white mask may hide the thumb if it was placed higher than the other fingers. The locations of the fingertips were then identified as the centre of the bottom line of each finger box and passed along to the touch detection subsystem.

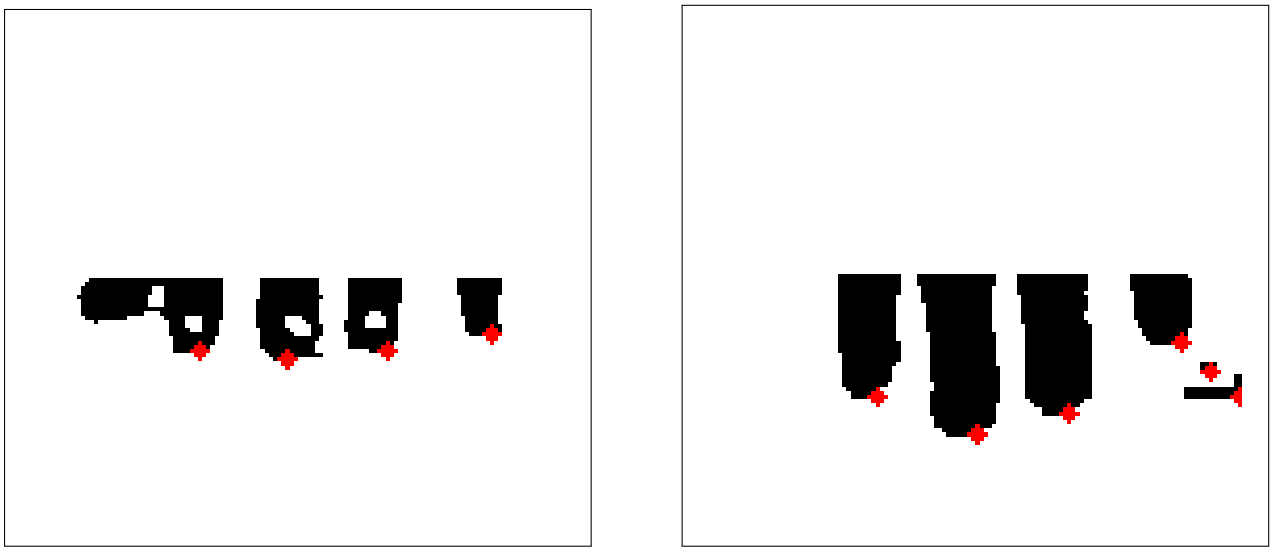
Challenges of the subsystem: A challenge that was encountered was the system's reduced ability to detect fingertips stably when two hands were brought into the frame. It was observed that when the second hand entered the frame, the performance of the skin mask reduced causing the fingertips to not be detected stably. The reason that this phenomenon would occur was that the webcam would automatically slightly alter the white balance of the frame when the second hand was brought into the frame. An attempt was made to correct this by disabling the auto white balance; however, this caused further issues as the frame would have a rich greenish tint which was not able to be used in the system. A second webcam, a Logitech C310, was tested and gave the same issue when disabling the auto white balance feature. Therefore, the issue was not resolved.

Alternative fingertip detection method: Due to the challenge that was presented above, an alternate fingertip detection method was explored. This method followed the same process depicted in Figure 16 up until the fingertip template matching step. The idea of this method was to find the lowest points of the fingertips found in the white masked region of the hand as seen in Figure 23a. This method involved computing the contours of each arch or fingertip and then calculating the lowest point of that arch. The contours were determined using a DFS approach in an 8-connected neighbourhood. This neighbourhood is defined as:

$$N_8(x,y) = \{(x \pm 1, y), (x, y \pm 1), (x \pm 1, y \pm 1)\} \tag{41}$$

The starting pixel was defined to be the first white pixel in the image that had not been visited yet. The DFS algorithm then visits each pixel once and once it's complete for a region, the boundary pixels are traced. The lowest point of that contour would be defined as the point with

the maximum y-coordinate and was then defined as the fingertip.



(a)
Four fingertips detected.

(b)
Incorrect points computed.

Figure 24.
Results of the lowest point finger detection algorithm.

Figure 24 shows the results obtained from the lowest point algorithm where in Figure 24a four fingertips were detected. The thumb was not detected as it was connected to the index finger, whereas in the template matching method, the thumb was detected as a fingertip for this image. Figure 24b depicts another failure case where background skin pixels were detected as fingertips. Due to these results, this method was no longer considered.

3.4.3 Touch Detection

The touch detection algorithm was designed to identify whether a fingertip crossed the base of the keyboard layout. From the viewpoint of the front webcam, the base of the keyboard layout was above the top edge of the layout. Therefore, the touch detection algorithm was designed to identify that a touch event occurred if a fingertip crossed the base of the keyboard. During the key detection phase, once the keyboard layout was detected using the front webcam, the y-coordinate of the base of the layout was identified. The system then calculated a line that was 4 pixels above the y-coordinate of the base, as the threshold line to register a touch event. This line will be referred to as the touch detection threshold line. The 4-pixel offset accounts for scenarios where the fingertip may have touched the base of the keyboard which would be difficult to detect since the front webcam was placed above the surface. Increasing this offset would lead to an undesirable rise in false detections.

The system calculated a rest position line to serve as an intermediate boundary to prevent false touches. This line was calculated to be 10 pixels above the touch detection threshold line. For a touch event to be recognized as valid, the system required the fingertip to first move above this rest position line before crossing the touch detection threshold. This mechanism ensured that

only deliberate movements triggered successive touch events, enhancing the system's precision and reducing false positives. The rest position line was critical in preventing the system from incorrectly registering a second touch if the fingertip remained stationary in subsequent frames. The system calculated two more lines that were used as thresholds for the intensity detection algorithm which is explained further in section 3.4.4.

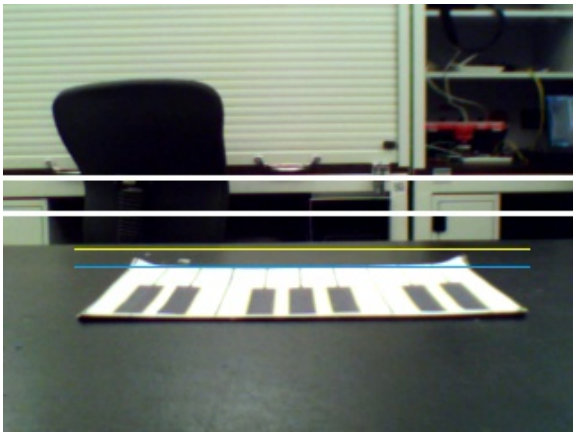


Figure 25.
Threshold lines.

Figure 25 depicts the 4 lines computed by the system used in the touch detection and intensity detection subsystems. The blue line represents the touch detection threshold line and the yellow line represents the rest position line. The two white lines represent the threshold lines used in the intensity detection algorithm. Once a fingertip crossed the touch detection threshold line, the touch event was triggered and the key that was touched needed to be identified. For the key to be identified, the point that the touch event was triggered, had to be mapped to regions of the keys that were detected by the system in the calibration stage. Since the front webcam did not detect the key regions and instead only detected where the layout was present in the frame, a mapping algorithm was developed to map a point on the keyboard from the front webcam to a point on the keyboard detected from the overhead-mounted webcam. This transformation of mapping a point from one view to another is known as a perspective transformation and can be described by a homography matrix which allows the mapping of points from one 2D plane to another [19]. The relationship between corresponding points in the two views is given as follows.

$$p' = H \cdot p \tag{42}$$

where

$$p = \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} \tag{43}$$

and

$$p' = \begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} \tag{44}$$

In regards to one view being the frame from the front webcam, and the second view being that of the overhead-mounted webcam, p, is a point in the front webcam view (in homogeneous coordinates) and p' is the corresponding point in the overhead-mounted webcam view. H is the 3×3 homography matrix that defines the transformation between the two views and can be

written as follows.

$$H = \begin{pmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{pmatrix} \tag{45}$$

Equation 42 can be written as:

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} \tag{46}$$

For a point $p = (x, y, 1)^T$ in the front webcam, the transformed point p' in the overhead-mounted webcam is given by:

$$\begin{pmatrix} x' \\ y' \\ w' \end{pmatrix} = \begin{pmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} \tag{47}$$

where w' is a scaling factor which accounts for perspective. After applying the transformation, the homogeneous coordinates are normalized to get the final pixel coordinates:

$$x' = \frac{h_{11}x + h_{12}y + h_{13}}{h_{31}x + h_{32}y + h_{33}}, \quad y' = \frac{h_{21}x + h_{22}y + h_{23}}{h_{31}x + h_{32}y + h_{33}} \tag{48}$$

To compute the homography matrix H, four pairs of corresponding points from the front webcam and overhead-mounted webcam were required. These points were computed during the calibration stage where template matching was used to identify where in the frame the keyboard layout was present. For each point pair (x, y) in the front webcam and (x', y') in the overhead-mounted webcam, the homography relation gives two equations:

$$x'(h_{31}x + h_{32}y + h_{33}) = h_{11}x + h_{12}y + h_{13} \tag{49}$$

and

$$y'(h_{31}x + h_{32}y + h_{33}) = h_{21}x + h_{22}y + h_{23} \tag{50}$$

These equations can be rearranged into a system of linear equations.

$$xh_{11} + yh_{12} + h_{13} - x'xh_{31} - x'yh_{32} - x'h_{33} = 0 \tag{51}$$

and

$$xh_{21} + yh_{22} + h_{23} - y'xh_{31} - y'yh_{32} - y'h_{33} = 0 \tag{52}$$

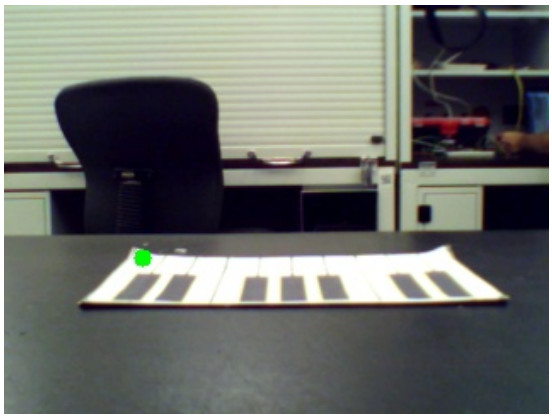
To solve for H, the above system of equations needs to be solved in matrix form, and therefore, written as:

$$\mathbf{Ah} = \mathbf{B} \tag{53}$$

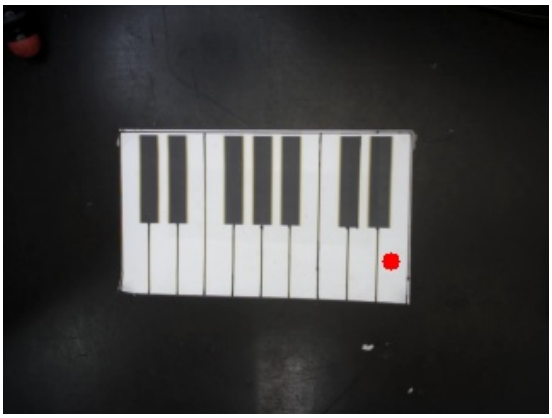
where A is the matrix containing the coefficients from the linear equations based on the front webcam points, h is the vector of homography parameters and B is the vector of the corresponding overhead-mounted webcam coordinates. The system can be solved using the least-squares method. The least-squares solution is:

$$\mathbf{h} = (\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T\mathbf{B} \tag{54}$$

Once H is computed, any point $p = (x, y, 1)^T$ in the front webcam can be mapped to the overhead-mounted webcam by multiplying H.



(a)
Front webcam point.



(b)
Overhead-mounted webcam point.

Figure 26.
Mapping the touch on an E key.

Figure 26 depicts when a touch was detected on the E key from the frame of the front webcam and how it was mapped to a point on the overhead-mounted webcam. Therefore, the touch detection method explained above used the input from a single webcam to successfully detect a touch and map it to a point within the keyboard layout. However, using the input from a single webcam reduced the system's accuracy when detecting a touch of a black key. This was because the method relied only on the frame of the front webcam to see whether the user's fingertip moved past the touch detection threshold line for the touch of a black key. When touching a black key, the system will always detect the fingertip touching a white key before it touches a black key, which happens because of the position that the front webcam is placed as well as the fact that the base of the black keys is always higher than the base of the white keys. The webcam cannot be positioned in line with the surface as the system would not be able to detect the layout of the keyboard. Therefore, an algorithm was designed for the system to detect the touch of a black key without the error of a white key touch.

The algorithm uses a skin mask on the frames of the overhead-mounted webcam to identify whether the hand is placed above or below the black keys. When the user started playing, the system checked to see if there were any white pixels above the black keys, and if there were, the assumption was made that the user was attempting to touch a black key. The point where the touch event was triggered was mapped to a point on the frame of the overhead-mounted webcam and then remapped to the nearest black key. The remapping algorithm iterated through all coordinates stored for the regions of the black keys and computed the centre of each box.

$$\text{center}_x = \frac{\min_x + \max_x}{2} \tag{55}$$

and

$$\text{center}_y = \frac{\min_y + \max_y}{2} \tag{56}$$

Therefore, the coordinates of the center of a box could be calculated as the following.

$$(\text{center}_x, \text{center}_y) = \left(\frac{\min_x + \max_x}{2}, \frac{\min_y + \max_y}{2} \right) \tag{57}$$

The Euclidean distance between the mapped point and the centre of a box was computed as:

$$\text{distance} = \sqrt{(\text{mapped}_x - \text{center}_x)^2 + (\text{mapped}_y - \text{center}_y)^2} \tag{58}$$

This Euclidean distance was computed between every black key box and the mapped point and the box that provided the lowest distance was assumed to be the key that was touched. The algorithm that remapped a point to the nearest black key was designed to also work in favour of using the white regions above the line that separates the black keys and the white keys. If the optimal template size was small, as depicted in Figure 12a, the regions detected for the black keys would be very small resulting in the perspective transformation algorithm possibly mapping a point outside of a black key. Therefore, the remapping algorithm ensures that the touch of a black key was detected if the user's hand was above the separation line. A drawback to the method of assuming that the user is touching a black key is that the system is unable to register the touch of a white key while the user's hand is above the separation line. Therefore, the system is unable to detect the simultaneous touch of a black key and a white key and will assume that only a black key is being touched. Algorithm 2 shows the algorithm followed to implement the nearest black key box method.

Algorithm 2 Find Nearest Black Key Box to a Mapped Point

Require: mapped_point: The point to be checked, black_key_boxes: A dictionary with box coordinates as keys
Ensure: closest_box: The coordinates of the nearest black key box

```
closest_box ← None
closest_distance ← ∞ {Initialize closest distance as infinity}
mapped_x, mapped_y ← coordinates of mapped_point
for all (box_coords, note) in black_key_boxes do
    box_center ← GetBoxCenter(box_coords) {Calculate center of current black key box}
    distance ← EuclideanDistance(mapped_point, box_center) {Compute distance to box center}
    if distance < closest_distance then
        closest_distance ← distance
        closest_box ← box_center {Update to the closest box found}
    end if
end for
return closest_box {Return the coordinates of the nearest black key box} =0
```

Challenges of the subsystem: A challenge encountered with the above implementation of the touch algorithm was the occurrence of false double-key touches when only a single key was touched. This issue occurred as the system detected the same fingertip as a different fingertip within 10 frames from the first key touch. The finger detection algorithm did not consistently assign the same index to the fingertip across frames, as it was possible for certain frames to lack fingertip detection. As a result, if the fingertip's index was updated shortly after a touch event was triggered, the system misinterpreted it as a new fingertip crossing the touch detection threshold. To prevent such false touch detections, three conditions had to be met.

The first condition introduced two frame counters, where the primary frame counter was incremented for every frame that was read from the webcams, while the secondary frame counter,

termed the key counter, was equated to the frame number that a touch event was triggered. The condition was then made that a touch event may only be considered valid if the current frame was more than 10 frames from the last touch event. The second condition tracked which hand was currently being processed, as accidental touch events were often triggered by the same hand. If a touch event occurred when the difference in frames was less than 10 frames and involved the same hand, the touch was considered invalid.

The third condition addressed the limitation introduced by the second: the system would incorrectly detect simultaneous key touches as invalid touches if the same hand was being processed. To enable valid simultaneous key detection, the assumption was made that a simultaneous key touch event would only be triggered if multiple fingertips crossed the touch detection threshold simultaneously within a single frame. These three conditions prevented false touch detections while allowing the possibility of playing multiple keys at once.

3.4.4 Intensity Detection

The intensity detection algorithm was designed to distinguish between the user desiring to play a loud and a soft note. The algorithm consists of calculating the centre of the box that tracks the user's hand and measuring the speed it travels between two points. The two white lines depicted in Figure 25 are used as a rest position threshold line and an intensity threshold line. The current position of the centre point of the box was stored and compared to the centre point in the next frame. If the centre point of the hand moved faster than a predefined threshold, the algorithm interpreted this as a fast touch, and if the fingertip moved lower than the touch threshold line, the respective note would be played at a higher intensity. Conversely, if the centre point of the hand moved slower than the predefined threshold and a touch event was triggered, the respective note would be played at a lower intensity.

Challenges of the subsystem: The initial intensity detection method relied on tracking the speed of each fingertip. This method used the same concept of calculating the speed to move between two lines in the frame. However, due to the unstable fingertip detection method, the calculation of the speed of the movement was not accurate. This occurred due to the indices of the detected fingertips changing continuously. Therefore, the speed of the hand was considered as it was a much more stable and accurate method.

3.4.5 Keyboard Note Generation

The keyboard note-generation algorithm creates the note for each key found on the standard and expanded layouts. Each note corresponds to a particular frequency which ranges from 261.63 Hz to 1318.5 Hz. Table 4 depicts the fundamental frequency for each note the system creates [20].

Note	Frequency (Hz)	Note	Frequency (Hz)
C4	261.63	C#5	554.37
C#4	277.18	D5	587.33
D4	293.66	D#5	622.25
D#4	311.13	E5	659.26
E4	329.63	F5	698.46
F4	349.23	F#5	739.99
F#4	369.99	G5	783.99
G4	392.00	G#5	830.61
G#4	415.30	A5	880.00
A4	440.00	A#5	932.33
A#4	466.16	B5	987.77
B4	493.88	C6	1046.50
C5	523.25	C#6	1108.73
D6	1174.66	D#6	1244.51
E6	1318.51		

Table 4.
Frequencies of Piano Notes.

A waveform was created for each note by summing the fundamental frequencies with harmonics where the harmonics aided in enriching the sound to make it more piano-like. The equation for generating a waveform is as follows:

$$\text{wave}(t) = 0.5 \cdot \sin(2\pi f_0 t) + 0.3 \cdot \sin(2\pi 2 f_0 t) + 0.2 \cdot \sin(2\pi 3 f_0 t) + 0.1 \cdot \sin(2\pi 4 f_0 t). \tag{59}$$

f_0 is the fundamental frequency of the note and t is the time. To make each note sound more natural, an ADSR envelope was applied which shaped the sound's amplitude over time, dividing it into four stages: Attack, Decay, Sustain and Release. Attack was the initial increase in amplitude, while Decay was the gradual reduction of amplitude that occurred after the attack. Sustain was the level at which the sound sustains while the note was held and Release was the decrease in amplitude after the note was released. The ADSR envelope can be described as a function that modulates the waveform's amplitude:

$$\text{envelope}(t) = \begin{cases} \frac{t}{A}, & \text{if } 0 \le t < A \\ 1 - \frac{t-A}{D}(1-S), & \text{if } A \le t < A + D \\ S, & \text{if } A + D \le t < \text{Duration} - R \\ S \cdot \left(1 - \frac{t-(\text{Duration}-R)}{R}\right), & \text{if Duration} - R \le t < \text{Duration} \end{cases} \tag{60}$$

A, D and R represent the times for attack, delay and release respectively, while S represents the sustain level. The duration of the waveform was set to be 500ms to play medium notes, allowing the note to have a clear attack, decay, sustain and release, giving it fullness. The attack time was chosen to be 10ms to make the sound start immediately, while the decay time was set to 100ms as it was fast enough to simulate how the sound of a piano note reduces slightly after being struck. The sustain level was chosen as 70% of the peak volume to simulate the behaviour of a piano. The release time was chosen as 100ms to give the note a natural fade, instead of ending it abruptly. The final waveform combines the harmonics with the ADSR envelope:

$$\text{final\_wave}(t) = \text{wave}(t) \cdot \text{envelope}(t) \tag{61}$$

Figure 27 represents the effect of the ADSR envelope on the middle C note. It can be seen that before applying the envelope, the note abruptly ends, however, applying the envelope allows the note to end gradually and sound more natural. Each note was generated at the initialization stage of the system, converted to 16-bit PCM format and stored as .WAV files. The system then preloaded the .WAV files into memory for efficient playback when touch events were triggered.

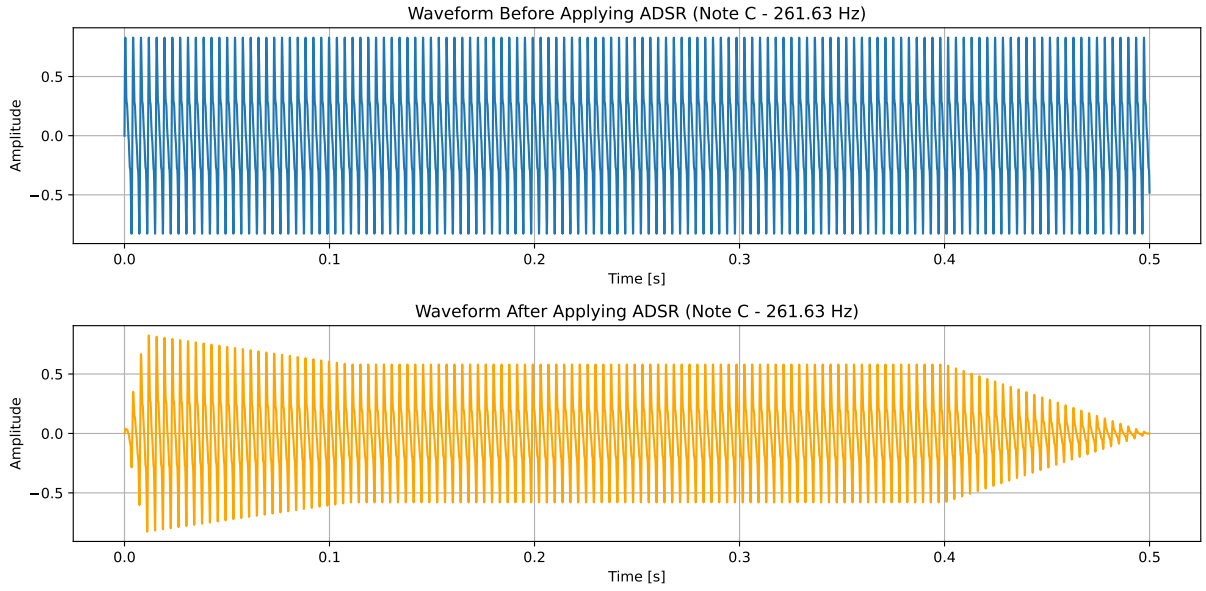


Figure 27.
Effect of ADSR envelope.

3.5 Optimizations

The system was optimized in various manners since it used Python and C. Regarding the key detection phase, the subsystem was optimized by computing the resizing of the template at the start of the system, which allowed for the system to spend its computational power on performing template matching and obtaining the regions of the keys. Similarly, the note detection subsystem generated the notes' sounds and loaded them in memory at the start of the system, to provide near real-time note playback. The fingertip detection subsystem computed the normalization and standard deviation of the templates at the start of the system as this only needed to be computed once as during the system's running it does not change.

Computationally intensive functions were implemented in C, such as bilinear interpolation, Otsu's thresholding, 4-connected flood fill algorithm, colour space conversions, skin mask and morphological operations. During the compilation of these C files, specific flags were added to the compilation commands to ensure that the functions ran in the most optimal form. The compilation command used looks as follows:

```
gcc -shared -o MorphOperations.so -O3 -march=armv8-a+simd -ffast-math -fPIC
MorphOperations.c
```

The `-shared` flag instructs the code to produce a shared library which allows the compiled

code to be dynamically loaded and shared between different modules reducing memory usage. The `-O3` flag is an aggressive optimization flag. The `-march=armv8-a+simd` flag specifies the ARMv8 architecture that is targeted. SIMD is enabled which allows the compiler to use instructions that operate on multiple data points in parallel. The `-ffast-math` flag allows the compiler to bypass strict math rules that could slow down the execution.

The system was optimized to run on either the Linux or Windows operating systems as it interpreted which operating system it was running on and loaded the required modules for that operating system.

3.6 System Integration

The subsystems were integrated to produce the final system. Figure 28 depicts the implementation of the final system.

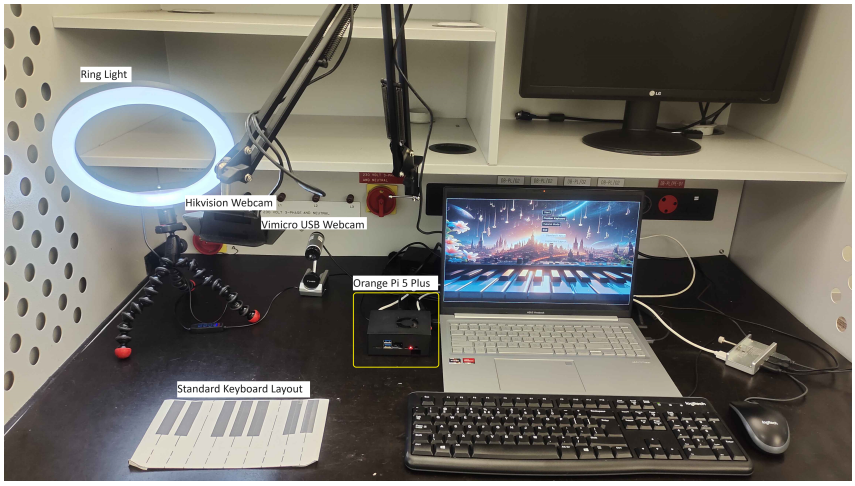


Figure 28.
Mobile Piano Implementation.

Figure 28 shows that the system is running on the Orange Pi 5 Plus board which is providing video output through the laptop screen. The standard keyboard layout is being used and the two webcams are pointed towards the layout.

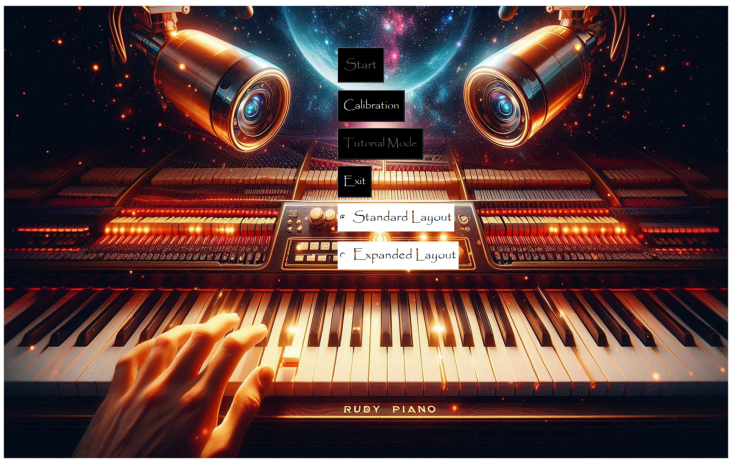


Figure 29.
Start screen of the system.

Figure 29 depicts the start screen of the system where the user was required to perform calibration before they could start playing. The user was expected to choose which layout was being used before calibration with the standard layout set as the default option.

4. Results

4.1 Summary of results achieved

Intended outcome	Actual outcome	Location in report
Core mission requirements and specifications		
The system should be able to detect the order of the notes played with a minimum accuracy of 90%.	The system achieved a note detection accuracy of 96.42%.	Section 4.2.1
The system should process the touch of a key within 150ms.	The system processed the touch of a key in under 10 ms.	Section 4.2.2
The system should be able to detect up to 10 fingers touching the keys.	The system was able to detect up to 8 fingers touching the keys.	Section 4.2.3
The system should be able to distinguish between the two basic dynamic indications in music, a loud touch (forte) and a soft touch (piano) with a minimum accuracy of 90%.	The system achieved an intensity detection accuracy of 88.42%.	Section 4.2.4
Field condition requirements and specifications		
The system should be able to identify the touch of a key when the keyboard layout is kept stationary, and the base of the keyboard layout is parallel to the camera at a position that is no more than 30cm from the base of the camera.	The system was able to identify the touch of a key when the keyboard layout was kept stationary, and the base of the keyboard layout was parallel to the camera at a position that was no more than 30cm from the base of the camera.	Section 4.2.1, Section 4.2.3
The system should be able to identify the touch of a key in a room with normal light having a lux range of 500 - 1000 on a surface where there is no obstruction between the camera and the keyboard layout.	The system was able to identify the touch of a key in a room with normal light having a lux range of 400 - 1000 on a surface where there was no obstruction between the camera and the keyboard layout.	Section 4.2.1, Section 4.2.3
The system should be able to detect fingers when the fingers are kept at a minimum distance of 20cm from the base of the camera.	The system was able to detect fingers when the fingers were kept at a distance of 30cm from the base of the camera.	Section 4.2.3

Table 5.
Result Summary

4.2 Qualification tests

4.2.1

Qualification test 1: System's Note Detection Accuracy

Objectives of the test or experiment: The test aimed to determine the system's accuracy in detecting the order of the notes played in tunes.

Equipment used: The experiment used the standard paper keyboard layout of 12 keys, a ring light to provide light to the system, and two webcams. The overhead-mounted webcam was the Hikvision DS-U12, while the front webcam was the Vimicro USB. The system ran on an Orange Pi 5 Plus board with Ubuntu 24.04. The system appended each detected note into an array and displayed it on the GUI. The final array was also printed on the terminal at the end of each run.

Test setup and experimental parameters: The keyboard layout was placed on a flat dark surface, with the front webcam being positioned 30cm in front of it. The ring light was positioned to shine light onto the area of the keyboard layout however it was not in the frame of the front webcam. Figure 30 depicts a simulation of the test setup.

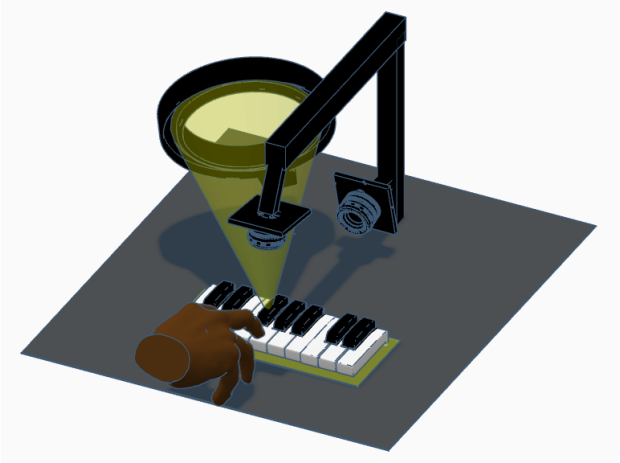


Figure 30.
Simulation of test setup.

The experimental parameters of the test setup involved using a single hand to play the tunes. Each tune was played at a slow pace to ensure that the fingers touched the correct keys. The system's accuracy in detecting each note played in the tunes was being measured.

Steps followed in the test or experiment: The system was restarted for each iteration of the tune to ensure that the optimal skin mask threshold values and optimal template size were achieved in the test. Three tunes were played for five iterations and a confusion matrix was produced for each iteration. An aggregate confusion matrix was then computed. For each iteration, the hand was kept facing the front webcam for each note that was played, as depicted in Figure 30. The three tunes that were chosen for the test were as follows:

- Für Elise by Beethoven: First 35 notes.
- Wavin' Flag by Knaan: First 15 notes.
- Interstellar by Hans Zimmer: 48 notes.

A fourth custom tune was also played that involved playing each note on the standard keyboard layout in ascending order of the notes' frequencies. Each note was touched twice, and therefore, a total of 34 notes were played.

Results or measurements: Figures 31 to 34 represent the aggregate confusion matrices of the four tunes. Figure 35 represents their overall accuracy.

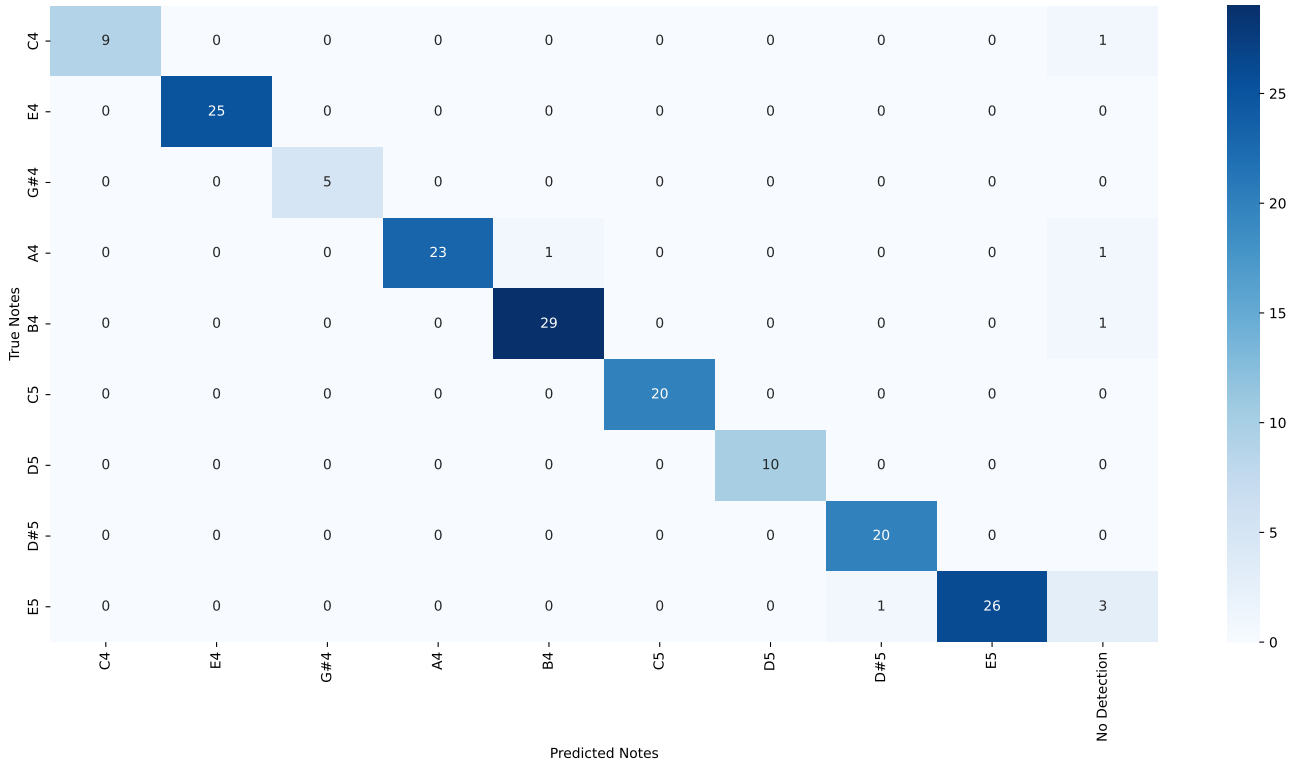


Figure 31.
Aggregate confusion matrix for Für Elise.

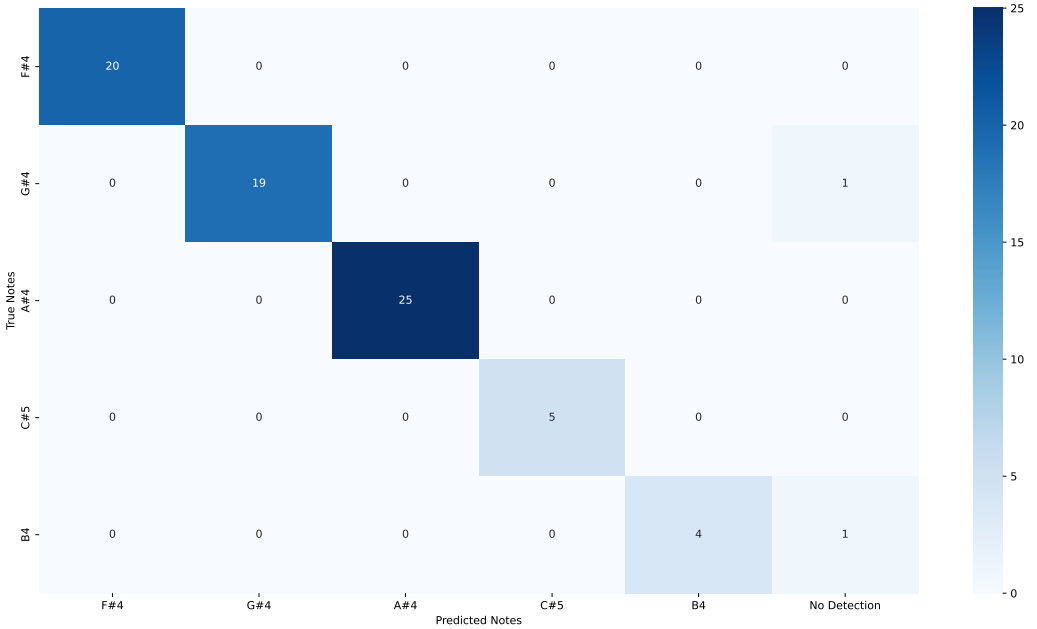


Figure 32.
Aggregate confusion matrix for Wavin' Flag.

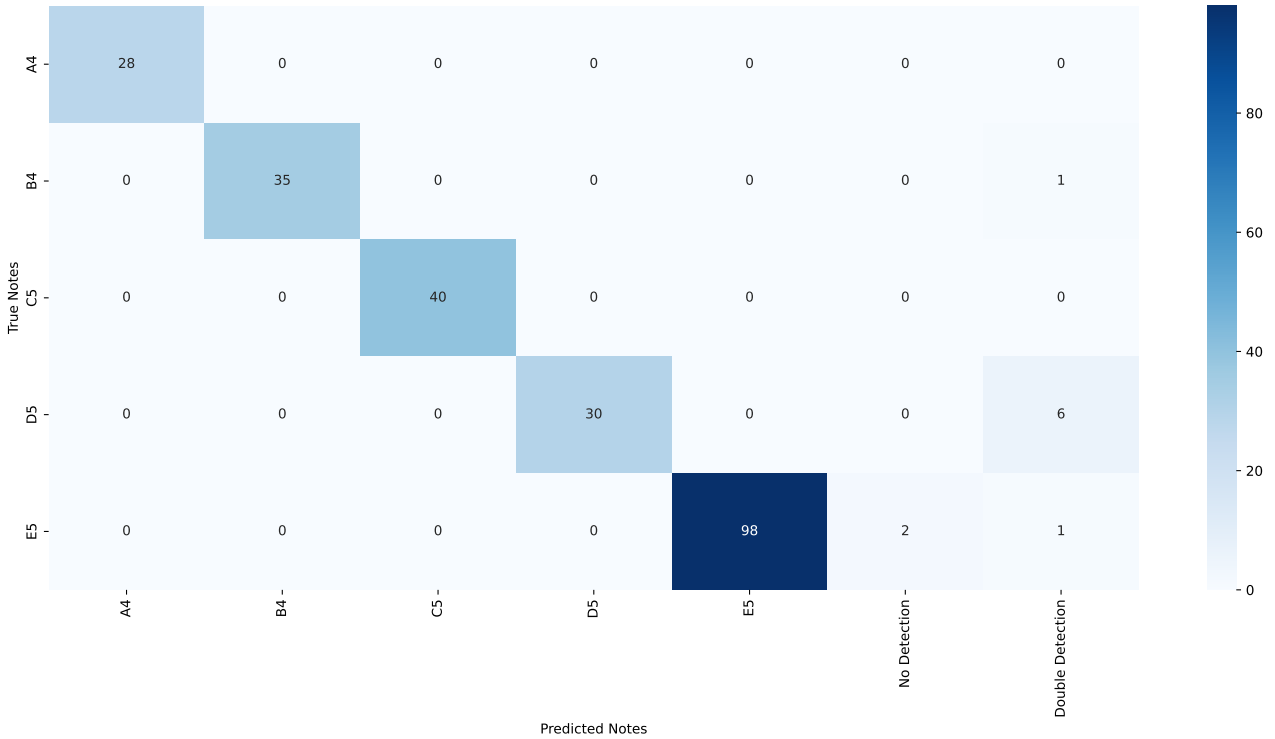


Figure 33.
Aggregate confusion matrix for Interstellar.

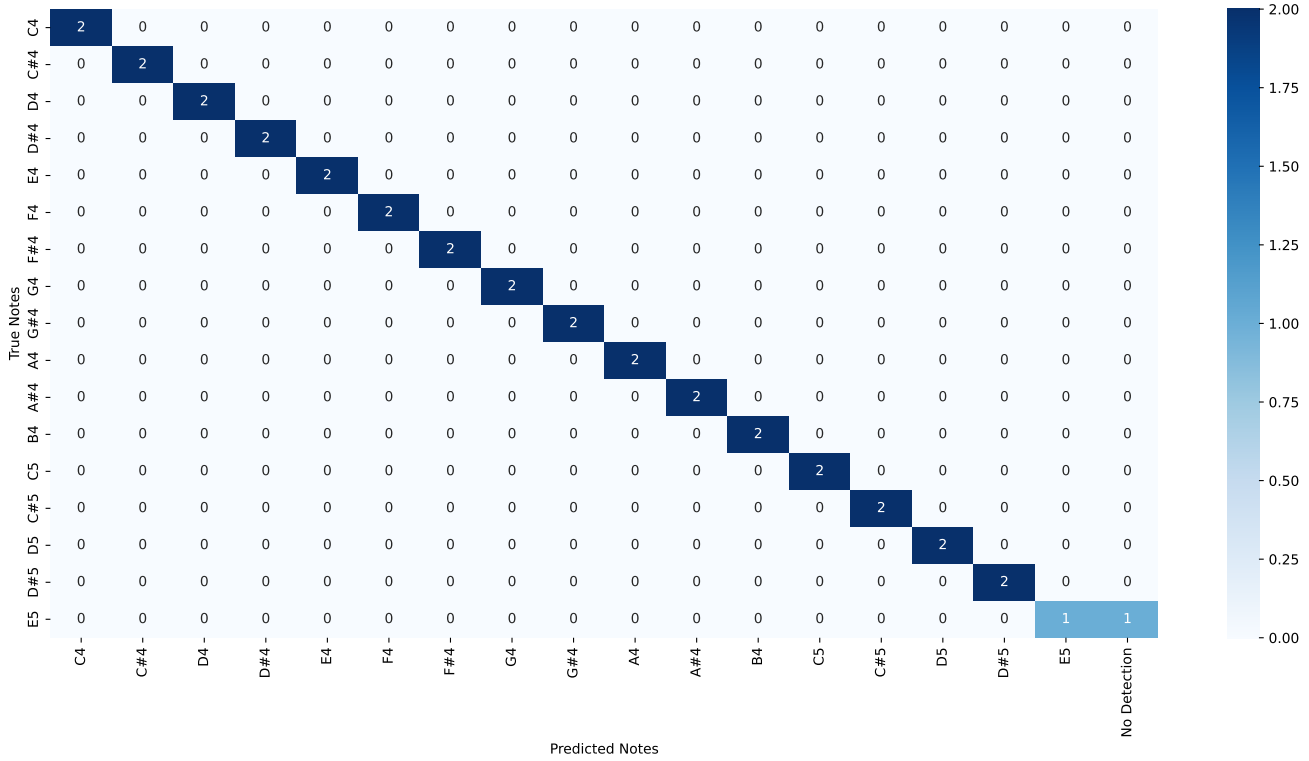


Figure 34.
Aggregate confusion matrix for Custom Tune.

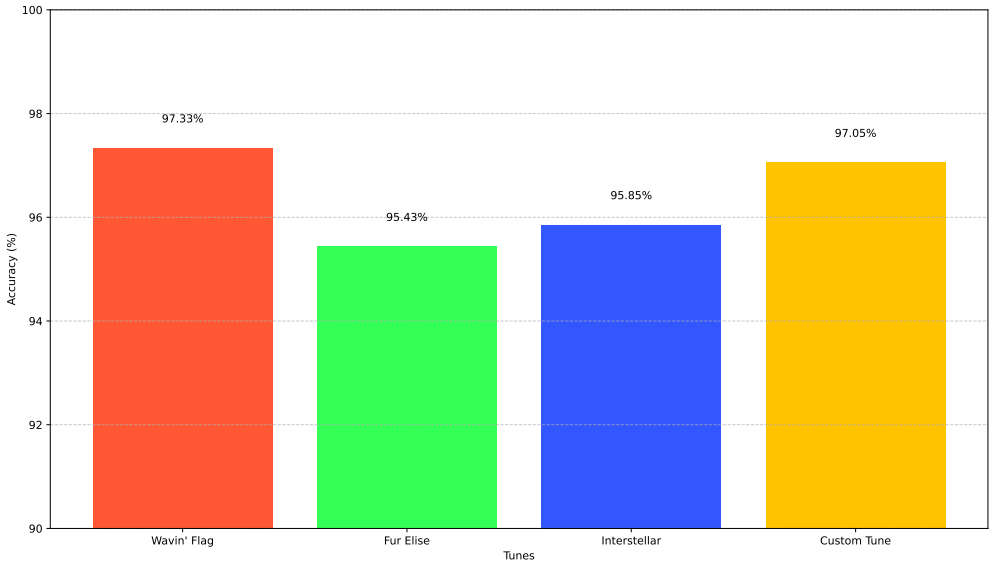


Figure 35.
Graph of overall accuracy for each tune.

Tables 6 to 10 depict the notes that were not detected for each tune and their number of occurrences that happened after 5 iterations.

Tune	Least Detected Note	Occurrences
Wavin' Flag	B4, G#4	1
Für Elise	E5	3
Interstellar	D5	6
Custom Tune	E5	6

Table 6.
Least Detected Notes for Each Tune.

Notes Not Detected	Occurrences
G#4	1
B4	1

Table 7.
Notes Not Detected for Wavin' Flag.

Notes Not Detected	Occurrences
C4	1
A4	1
B4	1
E5	3

Table 8.
Notes Not Detected for Für Elise.

Notes Not Detected	Occurrences
B4	1
D5	6
E5	3

Table 9.
Notes Not Detected for Interstellar.

Notes Not Detected	Occurrences
E5	1

Table 10.
Notes Not Detected for Custom Tune.

Observations: The aggregate confusion matrices in Figures 31 to 34 depict the correctly detected notes in the diagonal. Figure 35 shows that the system achieved a note detection accuracy of 97.33% for Wavin' Flag, 95.43% for Für Elise, 95.85% for Interstellar and 97.05% for the Custom tune. Table 6 shows that the Wavin' Flag tune had two notes that were the least detected. The other three tunes each had one least detected note after 5 iterations.

Statistical Analysis: The precision and recall were derived for each tune from their respective aggregate confusion matrix. Precision measures the ratio of correctly detected instances of a note to all instances where the system detected that note and is calculated as

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}} \tag{62}$$

where True Positives represents the number of correctly detected instances and False Negatives represents the number of incorrectly detected instances. Recall measures the ratio of correctly detected instances of a note to all actual instances of that note and is calculated as:

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}} \tag{63}$$

where False Negatives represents the number of occurrences where the note was not detected. Tables 11 to 13 depict the calculated precision and recall values for each note in each of the three tunes. Table 14 shows the overall average accuracy achieved after playing the four tunes.

Note	Precision	Recall
F#4	1.0	1.0
G#4	1.0	0.95
A#4	1.0	1.0
C#4	1.0	1.0
B4	1.0	0.8

Table 11.
Performance metrics for Wavin' Flag.

Note	Precision	Recall
A4	1.00	1.00
B4	1.00	1.00
C5	1.00	1.00
D5	0.833	1.00
E5	1.00	0.98

Table 12.
Performance metrics for Interstellar.

Note	Precision	Recall
C4	1.00	0.90
E4	1.00	1.00
G#4	1.00	1.00
A4	0.96	0.92
B4	0.97	0.97
C5	1.00	1.00
D5	1.00	1.00
D#5	0.95	1.00
E5	1.00	0.87

Table 13.
Performance metrics for Für Elise.

Tune	Overall Accuracy %
Wavin' Flag	97.33
Für Elise	95.43
Interstellar	95.85
Custom Tune	97.05
Overall Average	**96.42**

Table 14.
Average accuracy for each tune.

4.2.2

Qualification test 2: Measurement of the time taken to detect a note that was touched.

Objectives of the test or experiment: The test aimed to determine the time taken for the system to detect touched notes in a tune.

Equipment used: The experiment used the standard paper keyboard layout of 12 keys, a ring light to provide light to the system, and two webcams. The overhead-mounted webcam was the Hikvision DS-U12, while the front webcam was the Vimicro USB. The system ran on an Orange Pi 5 Plus board with Ubuntu 24.04, and the video output was connected to a laptop using a video capture card. OBS Studio was used to record the system's output and Microsoft Clipchamp was used to analyze the recorded video. The system also displayed on the GUI the time taken for the system to detect a touch after a fingertip crossed the touch detection threshold line and play the respective note.

Test setup and experimental parameters: The keyboard layout was placed on a flat dark surface, with the front webcam being positioned 30cm in front of it. The ring light was positioned to shine light onto the area of the keyboard layout however it was not in the frame of the front webcam. Figure 30 depicts a simulation of the test setup. The experimental parameters of the test setup involved using a single hand to play the tunes. The tunes chosen are the same as those

in the first qualification test, however only the first five notes of each tune were measured. Each tune was played at a slow pace to ensure that the fingers touched the correct keys.

Steps followed in the test or experiment: The system was restarted for each tune to ensure that the optimal skin mask threshold values and optimal template size were achieved in the test. The first five notes of each tune were played where the hand was kept facing the front webcam for each note that was played, as depicted in Figure 30. OBS Studio was used to record each tune and then Microsoft Clipchamp was used to analyze the timestamps of the recorded video. The time for the finger to be above the rest threshold line to below the touch detection threshold line was recorded. The time presented on the GUI of the system was also recorded.

Results or measurements: Tables 15 to 18 depict the detection latency of the first 5 notes for each tune. Table 19 depicts the average detection latency of each tune across the first 5 notes.

Note	**Timestamped latency (ms)**	**System Latency (ms)**
First Note	7	2.51
Second Note	7	2.51
Third Note	7	2.32
Fourth Note	7	2.32
Fifth Note	7	2.38
Average	**7**	**2.41**

Table 15.
Detection latency for Für Elise.

Note	**Timestamped latency (ms)**	**System Latency (ms)**
First Note	5	2.55
Second Note	5	2.40
Third Note	5	2.40
Fourth Note	5	2.38
Fifth Note	5	2.38
Average	**5**	**2.42**

Table 16.
Detection latency for Wavin' Flag.

Note	Timestamped latency (ms)	System Latency (ms)
First Note	12	2.4
Second Note	11	2.43
Third Note	11	2.37
Fourth Note	11	2.37
Fifth Note	11	2.35
Average	**11.2**	**2.38**

Table 17.
Detection latency for Interstellar.

Note	Timestamped latency (ms)	System Latency (ms)
First Note	9	2.55
Second Note	9	2.34
Third Note	9	2.47
Fourth Note	9	2.38
Fifth Note	9	2.45
Average	**9**	**2.44**

Table 18.
Detection latency for Custom Tune.

Tune	Average Timestamped latency (ms)	System Latency (ms)
Für Elise	7	2.41
Wavin' Flag	5	2.42
Interstellar	11.2	2.38
Custom Tune	9	2.44
Overall Average	**8.05**	**2.41**

Table 19.
Average detection latency for the four tunes.

Observations: From Tables 15 to 18 it can be seen that the system achieves a timestamped latency between 5 and 11.2 ms and system latency between 2.38 and 2.44 ms. The system achieved an average timestamped latency of 8.05 ms and a system latency of 2.41 ms as seen in Table 19.

4.2.3

Qualification test 3: Measurement of the number of fingers touching the keys as detected by the system.

Objectives of the test or experiment: The test aimed to determine the system's ability to detect multi-key detection.

Equipment used: The experiment used both paper keyboard layouts. The expanded layout was used to detect over five fingers as space was required between the hands for dual-hand detection. A ring light was used to provide light to the system. The system utilized two webcams where the overhead-mounted webcam was the Hikvision DS-U12, while the front webcam was the Vimicro USB. The system ran on an Orange Pi 5 Plus board with Ubuntu 24.04.

Test setup and experimental parameters: The keyboard layout was placed on a flat dark surface, with the front webcam being positioned 30cm in front of it. The ring light was positioned to shine light onto the area of the keyboard layout however it was not in the frame of the front webcam. Figure 30 depicts a simulation of the test setup. The experimental parameters of the test setup involved using both hands to touch the keys.

Steps followed in the test or experiment: The system was started to obtain the optimal template size and optimal skin mask threshold values. The test was carried out by incrementally increasing the number of simultaneous finger touches on the keys. For each level of simultaneous touch, measurements were taken over 10 separate trials.

Results or measurements: Figure 36 depicts the result of the test in a graphical form.

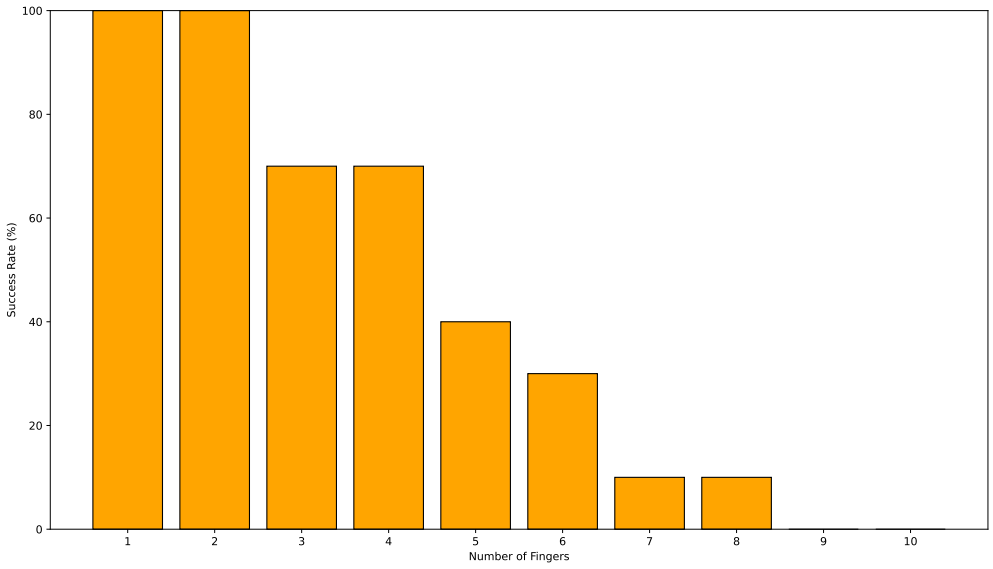


Figure 36.
Graph of detection rate for simultaneous number of keys being touched.

Observations: Figure 36 shows how using up to two fingers resulted in a 100% detection rate. Using three to four fingers resulted in a detection rate of 70% rate and using five and six fingers resulted in detection rates of 40% and 30% respectively. Lastly, using seven to eight fingers achieved a 10% detection rate and no detections were made when nine and ten fingers were touching the keys.

4.2.4

Qualification test 4: Measurement of the system's accuracy to detect the intensity of touch.

Objectives of the test or experiment: The test aimed to determine the system's accuracy in distinguishing if a user wanted to play a soft or a loud note.

Equipment used: The experiment used the standard paper keyboard layout of 12 keys, a ring light to provide light to the system, and two webcams. The overhead-mounted webcam was the Hikvision DS-U12, while the front webcam was the Vimicro USB. The system ran on an Orange Pi 5 Plus board with Ubuntu 24.04, and the video output was connected to a laptop using a video capture card. The system displayed on the GUI whether a soft or loud touch was detected. The system would also play the sound through a speaker at the detected intensity.

Test setup and experimental parameters: The keyboard layout was placed on a flat dark surface, with the front webcam being positioned 30cm in front of it. The ring light was positioned to shine light onto the area of the keyboard layout however it was not in the frame of the front webcam. Figure 37 depicts a simulation of the test setup where the black arrow represents the hand being moved at different speeds as it touches the key. The system's accuracy in detecting the intensity of the touch was being measured.

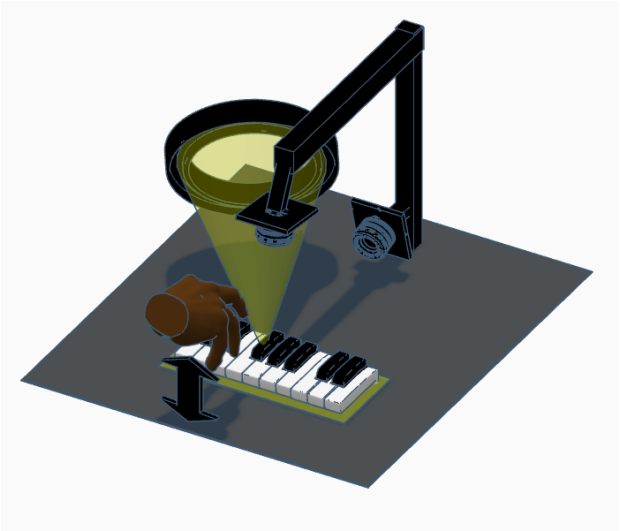


Figure 37.
Simulation of test setup.

Steps followed in the test or experiment: The system was started to obtain the optimal template size and optimal skin mask threshold values. Three tests were used to collect the results. The first test involved pressing a single note 10 times softly and the detected intensity was recorded. The second test involved pressing a single note 10 times loudly and the detected intensity was recorded. The third test involved playing the Wavin' Flag tune with the condition that each E note was meant to be played at a loud intensity while the rest of the notes were played softly. Five iterations of this test were performed; therefore, a total of 75 notes were played.

Results or measurements: Figures 38 and 39 depict the results of the first two tests. Figure 40 and table 20 depict the results of the third test.

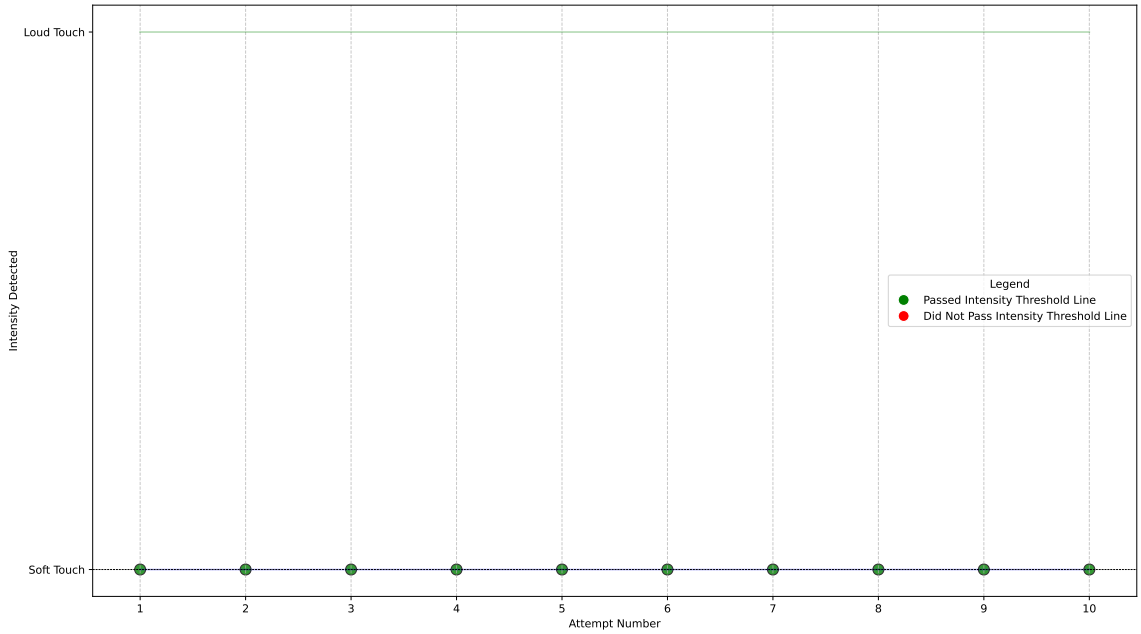


Figure 38.
Dot graph of intensity detection for only soft touches.

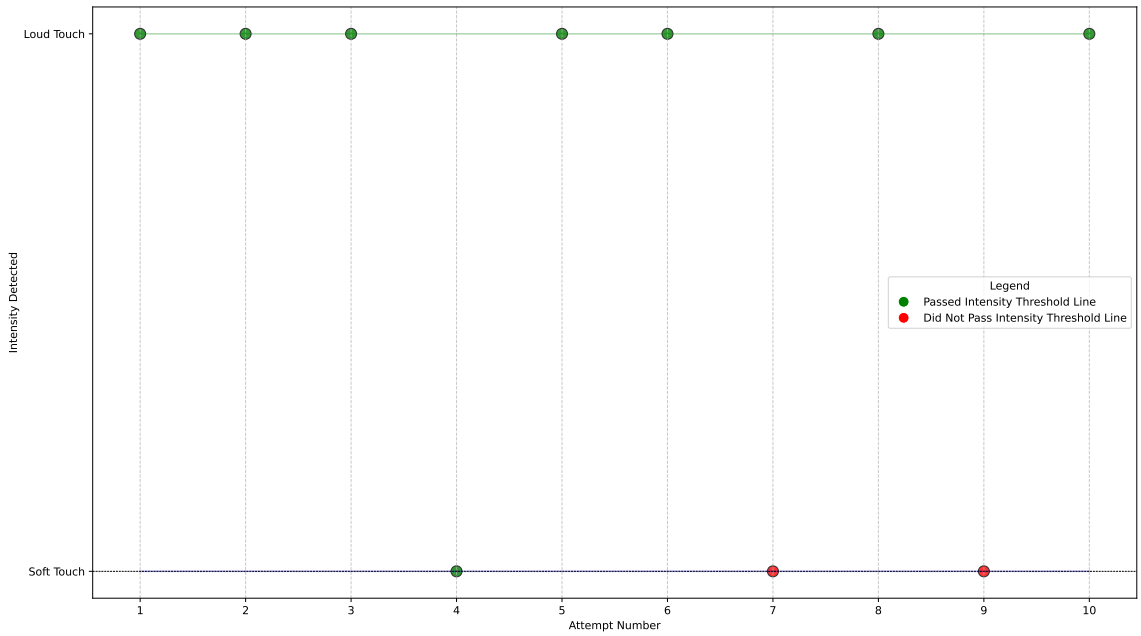


Figure 39.
Dot graph of intensity detection for only loud touches.

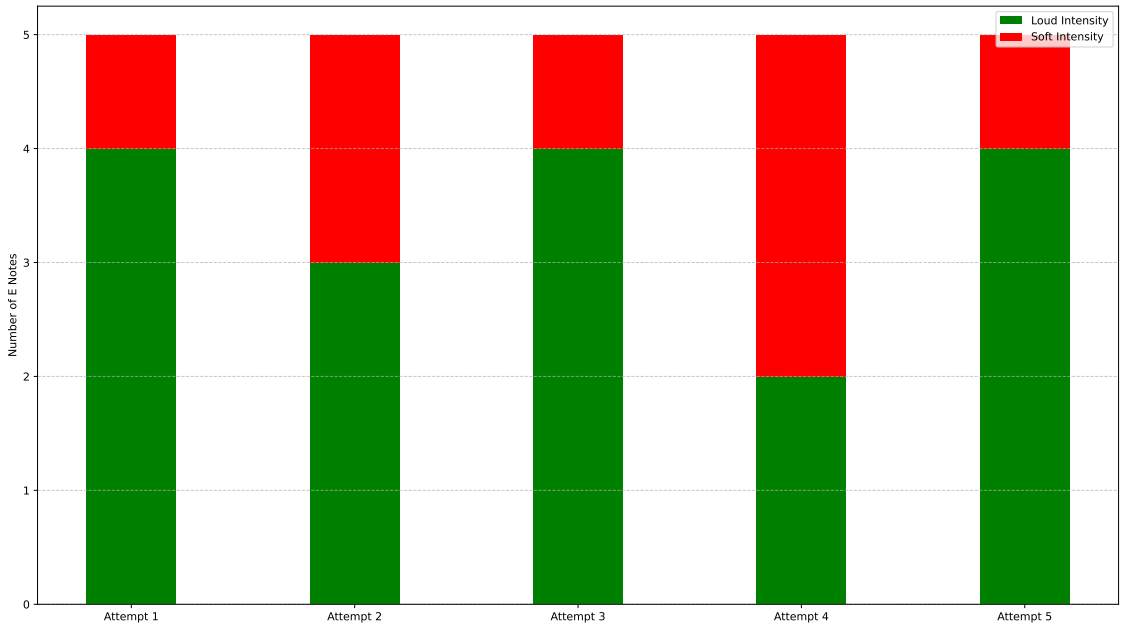


Figure 40.
Stacked bar graph showing the number of loud E-detected notes for Wavin' Flag.

Total Notes Played	Correct Intensity Detected Notes
75	67

Table 20.
Table showing the number of notes that had the correct intensity detected for Wavin' Flag.

Total Notes Played	Correct Intensity Detected Notes	Average %
95	84	88.42

Table 21.
Table showing the number of notes that had the correct intensity detected after all three tests.

Observations: Figure 38 depicts that 10 soft notes were detected for the first test. Figure 39 shows that seven notes were detected at a soft intensity for the second test and 3 notes at a loud intensity. Figure 40 depicts that seven E notes were detected at a soft intensity and 17 were detected at a loud intensity.

5. Discussion

5.1 Critical evaluation of the design

5.1.1 Interpretation of results

The system was evaluated based on the results achieved from the qualification tests that were performed in Section 4.1. The core requirement of the system was to allow the user to play a tune on the paper keyboard layout and produce the corresponding keyboard output. A minimum accuracy of 90% of correct notes detected was targeted.

The results of the first qualification test in Section 4.2.1 measure the system's note detection accuracy. Figure 31 depicts the aggregate confusion matrix that was produced after five iterations of playing the first 35 notes of Für Elise. It is observed that the system detected 167 notes out of 175 notes that were touched, which equates to an accuracy of 95.23%, therefore, the system achieved a higher accuracy than the targeted specification with this tune. However, two misclassifications did occur and a total of 6 notes were not detected at the end of the five iterations. When observing Figures 31 to 34 a trend is seen where the number of non-detections exceeds the number of misclassifications with a total of 11 non-detections as compared to 2 misclassifications.

Table 6 shows which notes of the tunes were least detected where both Für Elise and the Custom Tune detected the E5 note the least. Interstellar had the D5 note detected the least while Wavin' Flag had the B4 and G#4 note detected the least. It is observed that other than the G#4 note, the majority of the notes that were not detected occurred towards the right end of the layout. This is also evident in the Tables 11 to 13 based on the calculated precision and recall metrics of each note. It is speculated that the system tended to detect fewer notes toward the end of the layout as the fingertips may not have been detected as often as they did on the left side. A second possibility is that the system did not capture the frame where the fingertip crossed the touch detection threshold line causing a non-detection to occur. External factors such as lighting could also contribute to non-detections. If the fingertip was placed at an angle where not enough light reached it, it would result in the system not being able to detect the fingertip due to the lack of white pixels in the skin mask. Despite these non-detections, the system managed to achieve a note detection accuracy of over 95% for each tune as presented in Figure 35 and an overall note detection rate of 96.42% which is higher than the targeted 90%.

The results of the second qualification test in Section 4.2.2 measure the time taken for the system to detect a touch. A target of 150ms was set to detect a touch. Table 19 depicts the average latency measured for the first five notes of each tune. The average time-stamped latency recorded using Microsoft Clipchamp was 8.05 ms and the average system-measured latency was 2.41 ms. A discrepancy between the time-stamped and system latency occurs due to having to use external software to see the timestamps before and after the system detected a touch. The system-measured latency measured the time between the system detected a fingertip moving past the touch detection threshold line, to after the system played the respective note.

The achieved latency is lower than the targeted specification which is ideal as it results in a near

real-time response from the system when detecting a touch. It can be deduced that the system was able to map the point the user touched and play the respective note within a very short period. This result was achieved due to the optimizations that were applied explained earlier in Section 3.5. The choice of performing a calibration stage where the keyboard layout and the optimal skin mask threshold values were detected contributed to this low latency as performing these actions in real-time would considerably slow down the system as it would have to continuously test which keyboard layout size and skin mask threshold values are optimal.

The results of the third qualification test in Section 4.2.3 measure the number of fingers touching the keys as detected by the system. A target of up to 10 fingers touching the keys was set. Figure 36 depicts the detection rate for a simultaneous number of keys being touched. It is observed that the system managed to detect up to 8 fingers touching the keys simultaneously which is short by two fingers from the targeted specification. The system managed to achieve a 100% detection rate in detecting up to two fingers touching the keys. This accuracy drops to 70% when detecting up to 4 fingers. It can be deduced that the system provided the user with sufficient ability to play tri-ads and four-note chords. However, the accuracy drops below the desired rate when detecting five simultaneous notes and more. The system was unable to detect up to nine and ten fingers touching the keys. However, it is uncommon to play more than six notes at once.

Upon completing the test, it was observed that when using two hands, the stability of detecting the fingertips was reduced. This has been explained earlier in the challenges subsection of Section 3.4.3. This affected the results as the system was unable to detect all the fingertips present in both hands. A second reason that caused the reduction in accuracy was that the system only recognized simultaneous touches when they happened in the same frame. Therefore, if the system did not detect a fingertip in the frame, and a simultaneous touch was intended, the system would not recognize that specific fingertip touching the key. Therefore, due to the use of the skin mask, the webcam and the lighting played external roles in reducing the accuracy of simultaneous touch detection.

The results of the fourth qualification test in Section 4.2.4 measure the system's accuracy in detecting the intensity of a touch. A target of 90% intensity detection rate was set. Figure 38 depicts the result of a test that performed only soft touches which achieved a 100% intensity detection rate. Figure 39 depicts the result of a test when only loud touches were performed which achieved a 70% intensity detection rate. The system's default sound intensity level was set to soft, and would only set it to high if the user moved their hand at a faster speed than the set threshold. Therefore, since only soft touches were performed, the system was easily able to achieve a 100% intensity detection rate in the first test. When observing the second test, it is seen that in two out of the three misclassified soft touches, the system did not detect the hand passing the intensity threshold line. This occurs if either the hand was moved too fast which prevented a frame being captured where the intensity detection threshold line was passed, or if the intensity point on the hand never passed the intensity threshold line during the touch. However, on the fourth attempt, the system did capture the intensity point moving past the intensity detection threshold line, but it did not compute the speed to be higher than the set threshold.

Figure 40 depicts the result of a test where every E note in the Wavin' Flag tune was meant to be played at a loud intensity. In three of the five attempts, the system achieved an 80% intensity detection rate. The lowest detection rate was found in attempt four, where only a 40% intensity detection rate was achieved. Table 20 shows how of the 75 notes played from the Wavin' Flag tune, 67 notes had the intensity detected correctly which is a rate of 89.33%. When combining the three tests, it is observed that the system correctly detected the intensity of 84 notes out of 95

notes played. This results in an overall accuracy of 88.42% which is slightly lower than the target specification, however, it is an excellent result as it proves that the system is very capable of detecting the desired intensity the user would like the note to be played. This method of intensity detection was more reliable than the approach of trying to track the speed of the fingertip as described in Section 2.

5.1.2 Critical evaluation

Upon evaluation of the system, it is observed that various subsystems contributed to achieving a note detection accuracy rate of 96.42%. The implementation of template matching within the key detection subsystem proved to be an effective choice as it accurately detected the region in the frame where the keyboard layout was, on various surfaces. Regarding fingertip detection, the combination of a skin mask and template matching proved to be a strong approach to tracking the hand and worked decently well in detecting the user's fingertips. However, it was lacking in effectively detecting more than 5 fingertips as it did not detect all fingertips stably. To detect all 10 fingertips, the user had to position their hand in a certain manner for the system to detect the thumbs. This instability in the finger detection resulted in the system not being able to detect up to 10 fingers touching the keys.

The touch detection subsystem was implemented well however introduced limitations in the detection of multi-touch events such as requiring all fingertips to be detected and ahead of the touch detection threshold in the same frame to be considered as a simultaneous touch event. However, this approach did help prevent the occurrence of false detections which would have decreased the system's note detection accuracy greatly. Another drawback to the touch detection subsystem was that it would not detect when fingers were touching both a black and a white key. This was performed to remove false white key detections when a black key touch event was meant to be triggered.

The use of template matching proved to be an effective choice as it provided a fast solution to track a hand. Segmenting the hand frame and performing template matching to detect the fingertips proved to be a more computationally efficient method than attempting to search for fingertips within the entire frame. The choice of implementing specific computer vision algorithms in C, such as the morphological operations, colour space conversions and the flood-fill algorithm, proved to be computationally efficient resulting in near real-time performance in the detection of a touch. Lastly, the note detection subsystem was implemented well as the notes were produced at the start of the system and did not take up sufficient processing time during the playing of a tune.

5.1.3 Unsolved problems

The unsolved problems of the system include not being able to detect the simultaneous touches of a black and white key. Secondly, the system is unable to detect up to ten fingers touching the keys. The touch detection algorithm could be improved upon with more funding as RGBD cameras could be used to utilize the depth information that is provided to make a more robust touch detection algorithm. If more time was available, the fingertip detection algorithm could be improved by leveraging a technique such as a CNN.

5.1.4 Strong points of the design

Various aspects of the proposed system worked very well. Two of these aspects are the designed adaptive size matching and adaptive skin matching algorithms. The adaptive size matching algorithm provides the user with the flexibility to use different keyboard template sizes that meet their needs. This method proved to be a more effective method than the edge detection method that relied on detecting specific edges of the layout. The adaptive skin matching mask matching algorithm provided the user the flexibility to run the system under different lighting conditions. The finger and touch detection subsystems were designed well enough to achieve a note detection accuracy of over 90% when playing tunes with a single hand. The intensity detection algorithm was designed well as it accurately detected when the user desired to play a note loudly or softly.

5.1.5 Expected failure conditions

The system is expected to not detect the keyboard layout if it is positioned in a way that is not upright to the webcam frame. This is due to the system expecting the keyboard template to be in a specific orientation. The system is expected to fail if a user is wearing clothing that resembles skin colour as the system will not be able to distinguish the difference between the clothing and the hands since the clothing will be picked up as skin pixels. This can result in the hand not being detected when attempting to play a tune. The system is expected to not perform well if the user places their hand in such a way that the fingertips cannot be distinguished from the palm. This can occur if the user rests their palm on the surface. Therefore, if the user does not place their hand as seen in Figures 22d or 23b, the system is expected to not capture the fingertips correctly. The system is also expected to fail when attempting to use more than 2 hands on the layout as currently the system only detects a maximum of two hands. Any more than two hands can cause the system to not detect the desired hand when playing. The system is also expected to detect false touches if a finger is placed on the key for a long time and the finger detection algorithm updates the index of the finger. This occurs due to the instability of tracking every fingertip.

5.2 Considerations in the design

5.2.1 Ergonomics

The system was designed to be modular requiring very little setup. The user is just required to use an overhead-mounted webcam and a webcam that is kept on the surface. A ring light or any sort of desk lamp is required to provide light to the keyboard layout. The system is designed to be operated on a flat surface, and therefore, the user can choose to either stand or sit. For further flexibility, the user is allowed to position the overhead-mounted webcam at various heights and the system will detect the keyboard layout at predefined heights. The user can choose their own webcam to be kept in front of the layout and the system will obtain the optimal skin mask for it using the adaptive skin mask algorithm. A user-friendly GUI was designed that provides instructions for each step in the calibration phase.

5.2.2 Health and safety

The system uses a mount for the webcam that should be securely mounted to prevent it from falling and causing possible injuries. The ring light used should be positioned to provide light on the surface and not be shown directly in the user's eye.

5.2.3 Environmental impact

The webcams, ring light and SBC should be disposed of properly at the end of their life cycle. The paper keyboard layout, which is intended to provide a tactile interface, should be recycled at the end of its use.

5.2.4 Social and legal impact

The system democratizes music education through an approach that employs computer vision, together with a paper keyboard layout, where the user is not required to purchase a piano or digital keyboard. The system can further facilitate musical learning and creativity. If marketed, it would provide a potential source of employment through software development and manufacturing roles. The system adheres to the POPI act as it does not store any information about the user and video data is only processed locally.

5.2.5 Ethics clearance

Ethical clearance was not required for this system.

6. Conclusion

6.1 Summary of the work completed

This report describes the design and implementation of a system developed to transform a surface into a playable piano using computer vision techniques and a paper keyboard layout. The motivation to build such a system was to provide users with an inexpensive method to play the piano without the use of a physical piano.

A literature study was completed on systems that performed the similar function of identifying a keyboard layout and detecting the keys that a user pressed. Furthermore, literature on the combination of skin detection and template matching was studied. The knowledge gained from these studies was applied to create a system that could identify a keyboard layout within a frame, detect the fingertips of a user and identify where the user touched their fingers on the layout. The system was implemented in Python and used C code for computationally extensive algorithms. Several field tests were completed to analyze the performance of the system and identify the scenarios where the system could fail. The core test was to measure the system's accuracy in detecting the notes touched by a user and its result is depicted in Figure 35.

6.2 Summary of the observations and findings

Based on the results of the qualification tests that were performed, the system achieved a note detection accuracy of 96.42% proving that the system was very capable of identifying the notes that the user wanted to play. The system processed each touch in under 10 ms, proving that it could process a touch in near real-time allowing for tunes to be recognized when being played on the system. The system achieved an intensity detection accuracy of 88.42% proving that the system was very capable of recognizing the intensity that the user wanted to play the notes at. Furthermore, the system was able to detect up to eight fingers touching the keys allowing the user to play tunes that consist of tri-ads and four-note chords comfortably. A discovery was made that the system was unable to detect up to 10 fingers touching the keys due to the fingertip detection algorithm not being stable enough to meet this specification.

6.3 Contribution

Knowledge of computer vision techniques was gained in the applications of image processing, object detection, image filtering and edge detection. Experience in using OpenCV to interact with webcams was acquired. The understanding of machine learning concepts was expanded by attempting to build a hand and fingertip detection model. Advice was received from the study leader to improve the robustness of the system.

6.4 Future work

The system includes a tutorial mode that is currently under development which aims to teach the user to play specific tunes that are stored in the system. Further improvements could be made to the system such as allowing the simultaneous detection of white and black keys being touched. The system could include an option for the user to change the sound of the instrument that is being played, similar to what is already available on digital keyboards. The system could also be expanded to detect more keys and hands.

7. References

[1] H. Begum, S. Shaheen, M. Moetesum, and I. Siddiqi, "Digital beethoven — an android based virtual piano," in *Proc. 2017 13th International Conference on Emerging Technologies (ICET)*, 2017, pp. 1–5.

[2] C.-H. Sun and P.-Y. Chiang, "Mr. piano: A portable piano tutoring system," in *Proc. 2018 IEEE XXV International Conference on Electronics, Electrical Engineering and Computing (INTERCON)*, 2018, pp. 1–4.

[3] M. Akbari and H. Cheng, "Real-time piano music transcription based on computer vision," *IEEE Transactions on Multimedia*, vol. 17, no. 12, pp. 2113–2121, Dec. 2015.

[4] A. Goodwin and R. Green, "Key detection for a virtual piano teacher," in *Proc. 2013 28th International Conference on Image and Vision Computing New Zealand (IVCNZ 2013)*, 2013, pp. 282–287.

[5] M. K. M. Rabby, B. Chowdhury, and J. H. Kim, "A modified canny edge detection algorithm for fruit detection classification," *IEEE Xplore*, Dec 2018, (accessed Jul. 15, 2020). [Online]. Available: https://ieeexplore.ieee.org/document/8636811

[6] D.-D. Truong, V.-T. Nguyen, A.-D. Duong, C.-S. N. Ngoc, and M.-T. Tran, "Realtime arbitrary-shaped template matching process," in *Proc. 2012 12th International Conference on Control, Automation, Robotics and Vision (ICARCV)*, 2012.

[7] T. Wu and A. Toet, "Speed-up template matching through integral image based weak classifiers," no. 1, p. 1, Jan. 2014.

[8] J. M. Chaves-Gonzalez, M. A. Vega-Rodríguez, J. A. Gomez-Pulido, and J. M. Sanchez-Perez, "Detecting skin in face recognition systems: A colour spaces study," *Digital Signal Processing*, vol. 20, no. 3, pp. 806–823, 2010.

[9] R. Khan, A. Hanbury, J. Stottinger, and A. Bais, "Color based skin classification," *Pattern Recognition Letters*, vol. 33, no. 2, pp. 157–163, 2012.

[10] B. Vishal and K. D. Lawrence, "Paper piano — shadow analysis based touch interaction," in *Proc. 2017 2nd International Conference on Man and Machine Interfacing (MAMI)*, 2017, pp. 1–6.

[11] B. Muhammad and S. A. R. Abu-Bakar, "A hybrid skin color detection using hsv and ycgcr color space for face detection," in *Proc. 2015 IEEE International Conference on Signal and Image Processing Applications (ICSIPA)*, 2015, pp. 95–98.

[12] X. Wang, R. Wang, and F. Zhou, "Fingertips detection and hand tracking based on curve fitting," in *Proc. 2014 7th International Congress on Image and Signal Processing*, 2014, pp. 99–103.

[13] A. Singh, M. Singh, and B. Singh, "Face detection and eyes extraction using sobel edge detection and morphological operations," in *Proc. 2016 IEEE International Conference on Communication and Signal Processing (ICCSP)*, Jun. 2016.

[14] D. N. Chandrappa, M. Ravishankar, and D. R. RameshBabu, "Face detection in color images using skin color model algorithm based on skin color information," *IEEE Xplore*, Apr. 2011, available: https://ieeexplore.ieee.org/document/5941600. Accessed: Nov. 16, 2020.

[15] D. Sihombing, H. A. Nugroho, and S. Wibirama, "Perspective rectification in vehicle number plate recognition using 2d-2d transformation of planar homography," Oct. 2015.

[16] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 779–788.

[17] B. Gooch and A. Wilkie, *Non-Photorealistic Rendering*. San Francisco, CA: Morgan Kaufmann, 2001.

[18] Adaptive Vision, "Template matching," https://docs.adaptive-vision.com/4.7/studio/machine_vision_guide/TemplateMatching.html, 2017, accessed: 2024-08-01.

[19] M. Maulion, "Homography transform: Image processing," https://mattmaulion.medium.com/homography-transform-image-processing-eddbcb8e4ff7, 2021, accessed: Aug. 20, 2024.

[20] R. Sengpiel, "Calculator for note names," https://sengpielaudio.com/calculator-notenames.htm, 2024, accessed: Nov. 1, 2024.

Appendix

Expanded Layout Detection

Figures 41 and 42 depict the system's ability to detect the expanded keyboard layout.

Figure 41.
Key detection of the expanded layout.

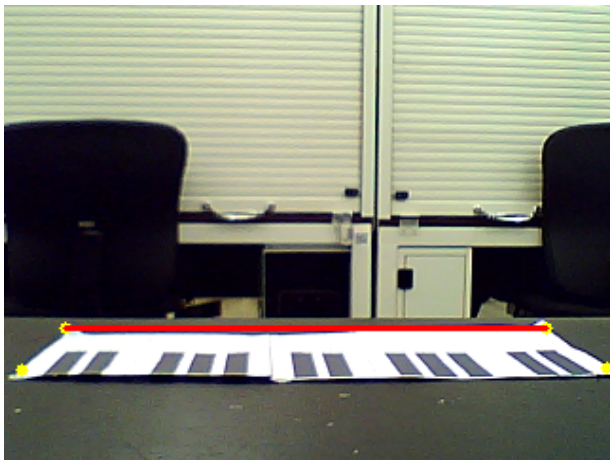


Figure 42.
Keyboard layout detection of the expanded layout.