Piano Fingering Detection System

Kane Xie
Dept.Computer Science and Software Engineering
University of Canterbury
Christchurch, New Zealand
kxi12@canterbury.ac.nz

Richard Green
Dept. Computer Science and Software Engineering
University of Canterbury
Christchurch, New Zealand
richard.green@canterbury.ac.nz

***Abstract*—Learning piano fingering is essential for mastering the piano, yet beginners often struggle to identify correct finger placement without guidance. This paper introduces a novel method, leveraging computer vision techniques and a MIDI keyboard, to automatically track piano fingering in real-time. The proposed system employs camera perspective correction, edge detection, and hand landmark detection to accurately identify which fingers play each note. Experimental results demonstrate high accuracy and performance, with recognition rates exceeding 90% for beginner-level piano pieces. Despite promising results, challenges remain, particularly in accurately tracking fingers near the edges of the keyboard and resolving ambiguities when multiple fingers interact with a single key. Nevertheless, the proposed method represents a significant advancement in automated piano fingering detection, offering potential benefits for piano learners and educators alike.**

I. Introduction

The piano is one of the most commonly learnt instruments in the world [1], and many people believe that learning it provides a range of benefits, such as a sense of accomplishment, stress reduction, and increased self-esteem [2]. As such, there is high demand for piano education.

As part of this education, professional pianists agree that fingering is one of the most important skills for beginners to learn [3], but since there are so many areas to concentrate on when learning a piece, beginners can find it a challenge to identify whether their fingerings are correct. In these cases, intervention from a piano tutor is often necessary to correct the problem.

However, an increasing number of musical educators have begun teaching instruments remotely in the last couple of years [4]. Since this makes it harder for instructors to correct their students' fingerings, it would be beneficial for these students to have a method to identify their fingerings automatically. Simply identifying which fingerings were used could help the student become aware of an issue, but such an approach could also be extended to notify students when they play an incorrect fingering.

II. Background

A small number of papers have introduced such methods in the past. Takegawa, Terada, and Nishio [5] introduced a method to track fingerings by placing coloured markers on each fingernail. However, their results had some key limitations. Firstly, their method only recognises one hand. Secondly, the detection fails when the marker is no longer visible, such as when one finger is under another finger or the palm, a common occurrence in piano performances. Finally, the effort involved in putting markers on each finger would likely dissuade students from using such a approach.

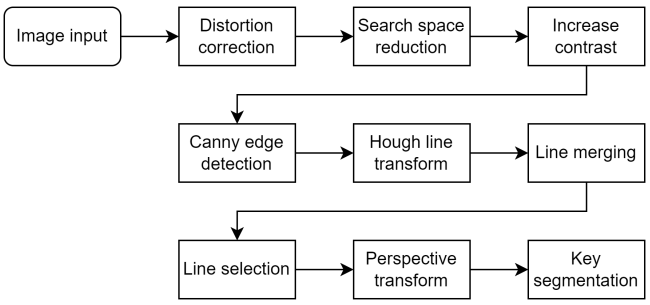

Fig. 1. Pipeline of the setup phase. The appropriate perspective transform is identified, and the image is then split up to segment the area of each key.

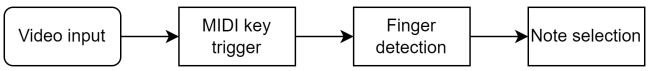

Fig. 2. Pipeline of the real-time phase. In this stage, the user's hand position and key presses are monitored, and the fingering is identified.

Oka and Hashimoto [6] introduced an approach that achieves a fairly high success rate for simple pieces, managing a recognition rate of 91.6% for J.S.Bach's Minuet in G major. However, depth cameras are specialised and expensive equipment, which could be a barrier for students. In addition, the success rate could still be a source of frustration for students, as about 1 in every 10 fingerings would be missing or marked incorrectly.

In this paper, we introduce a novel and low-cost method for fingering detection using a camera aimed at a 49 key (4-octave) MIDI keyboard. The algorithm takes in real-time video and MIDI input of the user playing a piece, and outputs pairs that consist of the triggered note and the finger that played the note.

III. Method

A. Distortion Correction

The first step of the method is to account for distortions in the video input, of which there are two main types - radial and

Fig. 3. A comparison between an uncalibrated image (top) and a calibrated image (bottom). In the uncalibrated image, the top of the keyboard does not align precisely with the red guiding line, and the keys are unevenly spaced. In the calibrated image, the bounds of the keyboard sit flush with the guideline.

tangential distortion. An example of this is given in figure 3.

Radial distortion causes straight lines to appear curved in the image, and is more noticeable as lines move further from the centre of the image. This distortion can be modeled with the following equation:

$$\begin{bmatrix} x_{distorted} \\ y_{distorted} \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix} (1 + k_1 r^2 + k_2 r^4 + k_3 r^6) \tag{1}$$

Where:

k_1, k_2, k_3 are the coefficients of the radial distortion;

$r^2 = x^2 + y^2$; and

x, y are the normalised image coordinates of a pixel, without the effect of the radial distortion. Normalised image coordinates are calculated by taking the distance between the pixel coordinate and the optical center, then dividing the resulting value by the focal length.

Tangential distortion is where the lens is not parallel to the imaging plane, which also modifies the resulting image. This can be modelled with:

$$x_{distorted} = x + [2p_1 xy + p_2(r^2 + 2x^2)] \tag{2}$$

$$y_{distorted} = y + [p_1(r^2 + 2y^2) + 2p_2 xy] \tag{3}$$

Using calibration objects that have multiple evenly spaced points, such as a checkerboard, these equations can be solved to find the behaviour of the camera. [7]

After finding these variables, distortion can be corrected so that straight lines in real life also appear straight in the input image. This is crucial for the latter stages of the pipeline.

B. Image Cropping

Because the next stages of the pipeline are computationally expensive, we crop out areas of the image that are not needed. This is done by using ArUco markers, a fiducial marker system [8]. Two markers are attached to the keyboard - one above, and one below. These are then identified with the approach described in [8], and all pixels above the top marker have their colour value set to black. Later on, the positions of these markers will be used to identify the bounds of the keyboard.

C. Image Adjustments

The next step is to adjust the image so that the Canny Edge Detection will be more accurate. Because there are gaps between the keys, the Hough Line Transform often does not identify the edge of the keyboard that faces the user. To adjust for this, a blur is applied to the image so that the missing pixels of the gaps are filled in, making the edge appear continuous. This blur is applied by taking the mean value of all pixels near the target pixel with a normalised box filter. An example 3 pixel by 3 pixel filter would look like this:

$$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \tag{4}$$

The filter used in this pipeline is 20 pixels by 5 pixels. This is applied separately from the blurring done as part of the Canny Edge Detection algorithm, because while the blurring in the algorithm is used to reduce noise [9], its purpose here is to make the keyboard area continuous.

In the next step, since the keyboard is largely white, increasing the contrast makes the edges more defined. The intersection between the black keys and the black keyboard frame was problematic for the border detection, so a white strip of paper has been attached to define the edge.

After this, the image is converted to grayscale to help the Canny Edge Detection, since each pixel now only needs to encode its brightness, not colour.

D. Canny Edge Detection

The Canny Edge Detection algorithm is used to identify edges in the image. There are four key steps in this algorithm: noise reduction, finding the edge gradient, non-maximum suppression, and hysteresis thresholding [10]. Firstly, the image is smoothed out because the algorithm is sensitive to noise. This is done with a 5x5 Gaussian filter. After this, the blurred image is convolved with a Sobel filter to approximate the horizontal and vertical first derivative (gradient) of the image. The Sobel filter is:

$$G_x = \begin{bmatrix} +1 & 0 & -1 \\ +2 & 0 & -2 \\ +1 & 0 & -1 \end{bmatrix} \tag{5}$$

$$G_y = \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} \tag{6}$$

Where:

G_x, G_y are the horizontal and vertical gradients, respectively [11].

From these values, we can easily calculate the magnitude and direction of each gradient, though the gradient is rounded to either be vertical, horizontal, or one of the diagonals for the next step.

The next step is non-maximum suppression, where every gradient that is not a local maximum is discarded (suppressed), and every local maximum is considered for the next stage. For example, if we have a matrix of horizontal gradients

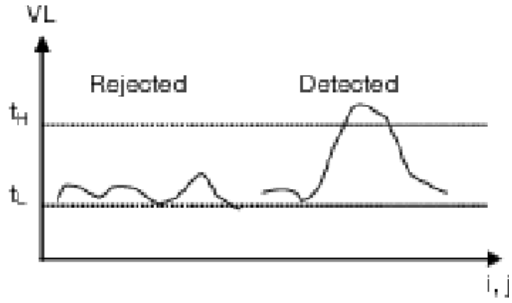

Fig. 4. Example of hysteresis thresholding: no gradient pixel in the left contour has a magnitude greater than the higher threshold, so all pixels in the edge are rejected. On the right, since part of the edge has a gradient magnitude greater than the higher threshold, the curve is accepted. [12]

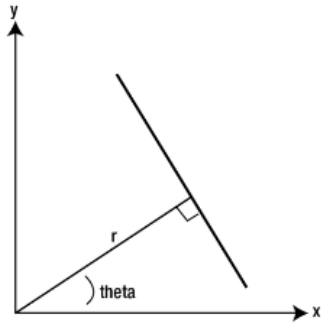

Fig. 5. Representation of a Cartesian line [10].

$\begin{bmatrix}4 & 8 & 2\end{bmatrix}$, we would select the pixel with the gradient of 8, as it is the only local maximum.

The last stage is hysteresis thresholding, the final selection for edges. This stage takes in two threshold inputs, one upper and one lower. A gradient is accepted as an edge pixel if its magnitude is greater than the upper threshold, and rejected if the magnitude is below the lower threshold. Gradients that have a magnitude between the two values are accepted if they are connected to an accepted gradient - that is, if they are adjacent to any accepted edge [10]. A diagram of this is given in Figure 4.

E. Hough Line Transform

After edges are detected, straight lines are tracked with a Hough Line Transform as described in [10]. The algorithm operates on some key principles: firstly, a straight line can be expressed in polar coordinates with a magnitude r and direction θ, as shown in Figure 5. This means that every straight line can be represented as a point (r, θ).

In addition, (x, y) coordinates can be transformed into (r, θ) coordinates with the following relationship:

$$r = xcos(\theta) + ysin(\theta) \tag{7}$$

This means that each point (x_0, y_0) has a family of straight lines, each represented by a (r, θ) pair, that go through that point. Plotting every (r, θ) that goes through a given (x_0, y_0), we get a sinusoidal curve. If two of these sinusoids are plotted

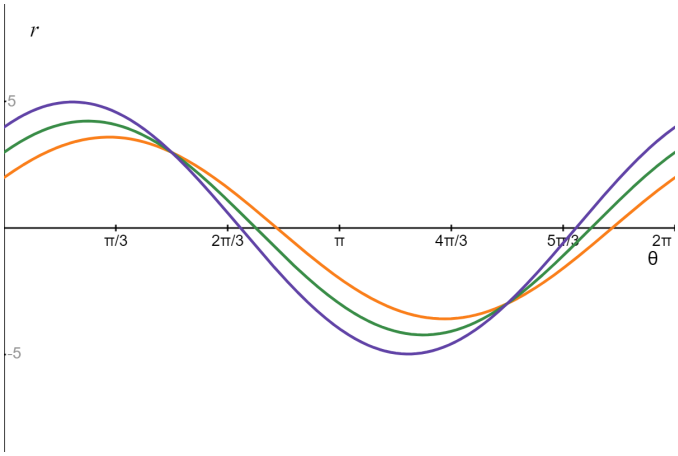

Fig. 6. Polar coordinate representations of lines with constant (x_0, y_0) values $(4, 3)$ (purple), $(3, 3)$ (green), and $(2, 3)$ (orange). The point of intersection, $(\pi/2, 3)$, represents a horizontal line $y = 3$ which intersects all three points.

together, the intersection of these lines represents a straight line that goes through both points, as shown in Figure 6.

As such, the first step in the algorithm is to map each edge pixel identified in the Canny Edge Detection step to a sinusoid in (r, θ) space. Then, for each intersection between curves, if the number of curves in that intersection is greater than some threshold, we store the point of intersection (which represents a line). We also introduce another threshold, minLineLength, where if the distance between two points is less than the threshold, they are considered in the same line.

After this, all stored lines greater than some length are accepted.

F. Finding Keyboard Bounds

Once the straight lines in the reference image are found, they are labelled as horizontal or vertical. The horizontal lines are merged if they are close together or continuous - that is, if two lines have extremities that are close together, and have similar orientations, we take the two furthest points as a new, merged, line.

We then take the two longest, continuous lines between the fiducial markers used previously to crop the image. These should be the top and bottom edges of the keyboard. Since keyboards are rectangular, we now know the four corners of the keyboard.

G. Aligning Keyboard

With the four corners of the keyboard, we can now perform a perspective transform to effectively get a top-down view of the keyboard. The 3×3 matrix of the transformation is found by taking four points on the source (in this case, the four corners of the keyboard), four points on the destination (the four corners of the output image), and using standard Gaussian elimination to solve the equation

$$\begin{bmatrix} t_i x_i' \\ t_i y_i' \\ t_i \end{bmatrix} = transformation\_matrix \times \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix} \tag{8}$$

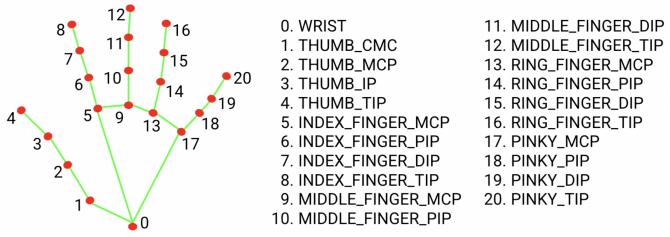

Fig. 7. Modelled hand landmarks from MediaPipe Hands [13].

Where:

t_i is a constant;
(x'_i, y'_i) are the coordinates of the transformed pixel;
(x_i, y_i) are the coordinates of the source pixel; and
$i = 0, 1, 2, 3$.

H. Mapping Keys

Since the layout of the keyboard is known beforehand, the area of each key is drawn onto the transformed image.

I. Hand Detection

After the setup pipeline is complete, the real time phase begins. In this phase, the program listens to the connected MIDI keyboard for any key presses. When a key is triggered, the program runs the MediaPipe Hand Landmark Model pipeline [13] on the current frame to detect the location of 21 key points on each of the user's hands, which are shown in Figure 7. These points are then transformed with the perspective transform solved in the setup phase. If one of the fingertips are within the segmented area of the triggered key, the combination of triggered key and finger is recorded as the recognised fingering. If there are two or more fingers within the key bounds, the finger closer to the performer is chosen, as the performer could be playing a note with one finger under another [14]. Finally, if no fingers are within the recognised area, the fingering is marked as missing.

In this pipeline, the methods for camera calibration, fiducial marker detection, blurring, converting to grayscale, Canny Edge Detection, Hough Line Transform, and perspective transformation were provided by OpenCV, an open source computer vision library [15].

IV. Results

The pipeline was run on a desktop computer, the specifications of which are shown below:

- **OS:** Windows 11 Home
- **Processor:** Intel Core i7-9700F CPU @ 3.00GHz
- **IDE:** PyCharm
- **Language:** Python 3.11.4
- **Camera:** Verbatim 1080p Full HD Webcam 1920x1080, 30fps
- **OpenCV version:** 4.9.0.80

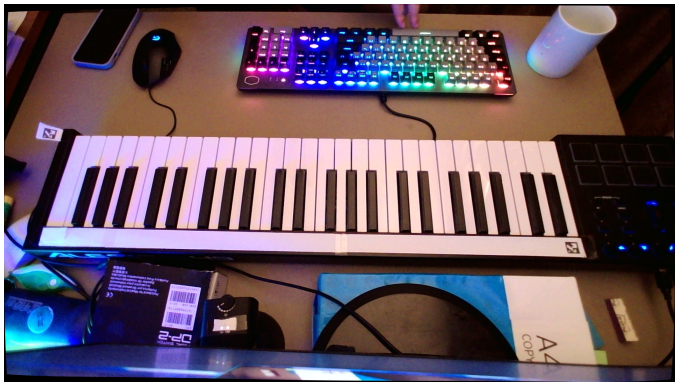

Fig. 8. Reference image for the setup phase.

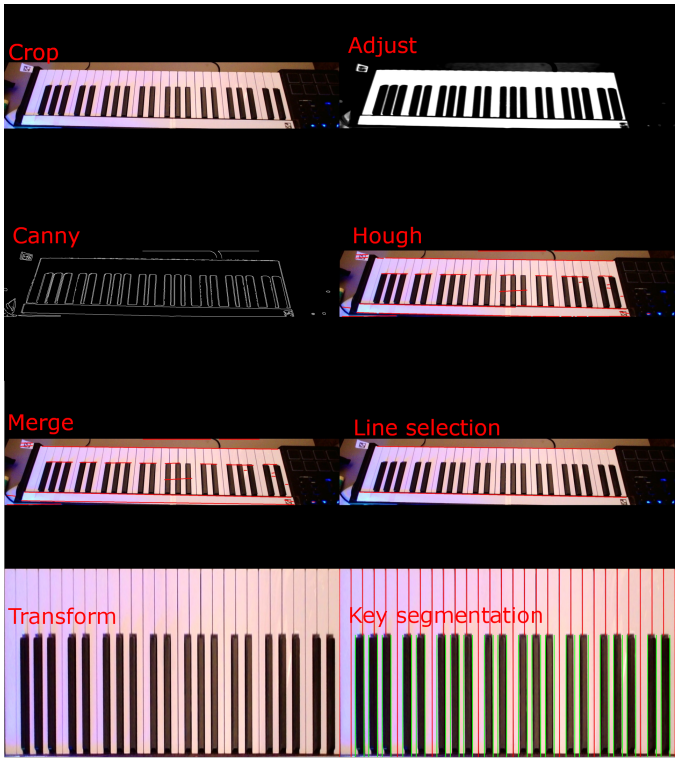

Fig. 9. The different stages of the setup pipeline.

A. Setup

The camera was set up 12cm away from the keyboard and 45cm high, pointing downward at 60 degrees from horizontal. The reference image is shown in Figure 8. The image at each stage of the setup phase is shown in Figure 9. While the result was largely successful, the Canny Edge Detection and Hough Line Transform parameters needed to be changed every time lighting conditions changed, which was highly tedious.

During the real-time phase, markers were added to indicate the position of each fingertip when a key was pressed, as seen in Figure 10.

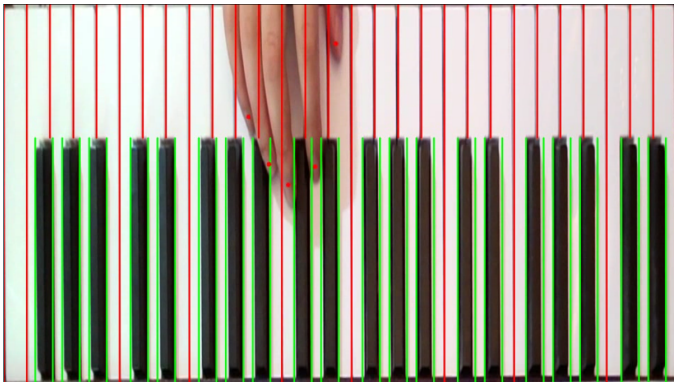

Fig. 10. Pressing a key. The red points represent the location of each fingertip, as returned by MediaPipe Hands.

B. Method Accuracy

The first piece that the proposed method was tested with was the first 16 bars of Bach's Minuet in G major. This piece was chosen for its relative complexity - the piece involves a number of notes being played at the same time, finger crossovers, and some scales. In addition, it means that the results of this paper can be compared against those in [6].

The second extract is the first 11 bars of Bach's Prelude in C minor, BWV 999, played at 100 crotchet beats per minute, or 6.67 notes per second. This piece was chosen because while only one note is triggered at a time, it can be played at a quick pace.

To test, we first play the piece with an exact fingering and record the output from the method. The response is then manually checked for accuracy. There are two metrics we focus on: recognition rate, and precision.

Recognition rate is the number of fingerings identified, out of the total number of notes played. Precision is defined as the number of correct labels marked by the proposed approach, divided by the total number of labels the approach managed to mark [16].

The results are shown in Table I. The proposed method had good recognition rates and precision for the Minuet in G minor, but the results were considerably more favourable for the Prelude in C minor.

TABLE I
RECOGNITION RATE AND PRECISION OF THE PROPOSED METHOD ON PIANO MUSIC.

	Minuet in G minor	Prelude in C minor
Number of notes	102	132
Fingerings recognised	92	126
Fingerings correct	86	126
Recognition rate	90.2%	95.5%
Precision	93.5%	100%

C. Speed

On average, the setup phase took 1.44 seconds to complete. In the real-time phase, the time it took for a fingering to be recognised after receiving a trigger from the keyboard was measured across 185 presses. Every key on the keyboard was pressed, and every finger was used at least once. The mean time for fingering recognition was 36.3 milliseconds, which is quick enough for many beginner pieces to be played in real-time.

D. Limitations

There are some key limitations to the method proposed in this paper. Firstly, the accuracy of the proposed method is greatly reduced near the ends of the keyboard. This is because the method of partitioning each key assumes a top-down view of the keyboard. However, towards the ends of the keyboard, this becomes less true. When we project the position of the finger onto a horizontal plane, the height difference between the tip of the finger and the key becomes more apparent, as illustrated in Figure 11. This poses a significant challenge, as while the area of key segmentation could be adjusted to correctly fit a particular camera position, the key segmentation would be wrong if the camera were to move. This is the main reason that the method performed considerably better on the Prelude in C minor compared to the Minuet in G major - while most of the notes in the Prelude were directly below the camera, many notes in the Minuet were near the ends of the keyboard, often causing the approach to fail to pick up the finger that pressed the key.

Another issue is that when two or more fingers are detected within the bounds of a note, the wrong finger is accepted. For example, if the third finger is depressing a note, but the thumb happens to be behind that finger, the proposed method records the thumb as the fingering instead of the third finger. However, this fix is also not trivial, as a performer could choose to play a note with two fingers at the same time. Even if one of the fingers was just resting on the note when another finger played that note, it would be very hard to pick up which finger provided actually triggered the note from the camera's point of view. In the Minuet, this was a big factor in reducing the precision. In the Prelude, a finger never went under another finger, so the precision was 100%.

Finally, the setup phase is sensitive to lighting conditions - every time the lighting changes, the Canny and Hough parameters need to be readjusted. While the results are very accurate when properly tuned, this process can take a considerable amount of time, and would be a major source of frustration for users.

V. CONCLUSION

In conclusion, in this paper we introduced a novel, low cost, and high performance method to detect fingering during piano performances. This method has adequate accuracy on real beginner pieces.

The proposed method has a number of advantages over the previous approaches of [6] and [5]. Firstly, while all methods require a MIDI keyboard, the only other piece of equipment required in the proposed method is an RGB camera. [6] also requires a depth camera in addition to the keyboard, and [5] requires coloured markers to be put on each finger.

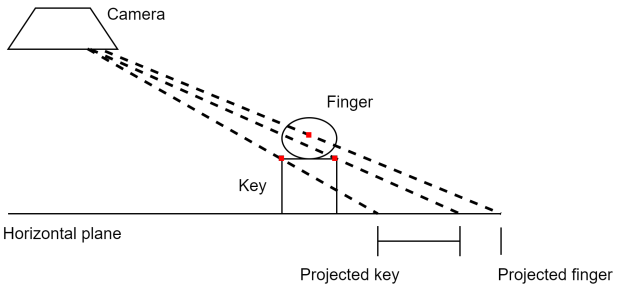

Fig. 11. Effect of height difference between fingertip centre and key. As the key moves further from the camera's normal vector, the assumption that the camera has a top-down perspective is more inaccurate.

Our method has a recognition rate of 90.2% of Bach's Minuet in G major, which is comparable to the 91.6% of [6]. However, our proposed method has a very high recognition rate of 95.5%, and perfect accuracy, on Bach's Prelude in C minor. This is also much higher than the 74% recognition rate of [5].

In addition, the method is highly performant, with a processing time of 36.3ms per note, considerably quicker than the 120ms of [6]. Though [5] had a processing time of 20ms per note, their system only tracks a single hand across a range of two octaves.

However, there are still some limitations to this method, such as the proposed method being less accurate near the ends of the MIDI keyboard, picking up the wrong finger when two fingers are on the same note, and having a fragile setup phase.

Future research could look to overcome these limitations, or extend the approach to inform the user when an incorrect fingering has been made.

REFERENCES

[1] P. J. Flowers and J. W. Murphy, "Talking about music: Interviews with older adults about their music education, preferences, activities, and reflections," *Update: Applications of Research in Music Education*, vol. 20, no. 1, pp. 26–32, 2001. [Online]. Available: https://doi.org/10.1177/875512330102000106

[2] P. J. Jutras, "The benefits of adult piano study as self-reported by selected adult piano students," *Journal of Research in Music Education*, vol. 54, no. 2, pp. 97–110, 2006. [Online]. Available: https://doi.org/10.1177/002242940605400202

[3] E. Clarke, R. Parncutt, M. Raekallio, and J. Sloboda, "Talking fingers: An interview study of pianists' views on fingering," *Musicae Scientiae*, vol. 1, no. 1, pp. 87–107, 1997. [Online]. Available: https://doi.org/10.1177/102986499700100106

[4] A. M. Hernández, "Online learning in higher music education: Benefits, challenges and drawbacks of one-to-one videoconference instrumental lessons," *Journal of Music, Technology & Education*, vol. 13, no. 2-3, pp. 181–197, 2020. [Online]. Available: https://intellectdiscover.com/content/journals/10.1386/jmte_00022_1

[5] Y. Takegawa, T. Terada, and S. Nishio, "Design and implementation of a real-time fingering detection system for piano performance," in *ICMC*, 2006.

[6] A. Oka and M. Hashimoto, "Marker-less piano fingering recognition using sequential depth images," in *The 19th Korea-Japan joint workshop on frontiers of Computer vision*. IEEE, 2013, pp. 1–4.

[7] G. Bradski and A. Kaehler, *Learning OpenCV: Computer vision with the OpenCV library*. " O'Reilly Media, Inc.", 2008.

[8] S. Garrido-Jurado, R. Muñoz-Salinas, F. Madrid-Cuevas, and M. Marín-Jiménez, "Automatic generation and detection of highly reliable fiducial markers under occlusion," *Pattern Recognition*, vol. 47, no. 6, pp. 2280–2292, 2014. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0031320314000235

[9] W. Rong, Z. Li, W. Zhang, and L. Sun, "An improved canny edge detection algorithm," in *2014 IEEE international conference on mechatronics and automation*. IEEE, 2014, pp. 577–582.

[10] S. Brahmbhatt, *Practical OpenCV*. Apress, 2013.

[11] N. Kanopoulos, N. Vasanthavada, and R. Baker, "Design of an image edge detection filter using the sobel operator," *IEEE Journal of Solid-State Circuits*, vol. 23, no. 2, pp. 358–367, 1988.

[12] N. Hautière, J.-P. Tarel, and R. Brémond, "Perceptual hysteresis thresholding: Towards driver visibility descriptors," 10 2007, pp. 89 – 96.

[13] F. Zhang, V. Bazarevsky, A. Vakunov, A. Tkachenka, G. Sung, C.-L. Chang, and M. Grundmann, "Mediapipe hands: On-device real-time hand tracking," 2020.

[14] S. Furuya, M. Flanders, and J. F. Soechting, "Hand kinematics of piano playing," *Journal of neurophysiology*, vol. 106, no. 6, pp. 2849–2864, 2011.

[15] G. Bradski, "The opencv library." *Dr. Dobb's Journal: Software Tools for the Professional Programmer*, vol. 25, no. 11, pp. 120–123, 2000.

[16] M. Buckland and F. Gey, "The relationship between recall and precision," *Journal of the American society for information science*, vol. 45, no. 1, pp. 12–19, 1994.