AR Piano Playing Using Real-Time Hand Tracking

Ayano Hiranaka
ayanoh@stanford.edu

Eden Grown-Haeberli
edengh@stanford.edu

Kangrui Xue
kangruix@stanford.edu

Abstract

Human hand motions are rich and varied, and tracking them represents a key problem in computer vision. In this project, we investigate hand tracking and interaction via a real-time AR (augmented reality) piano playing app. Using only input rgb frames from a webcam, we utilize Google's MediaPipe Hands API to predict 3D hand keypoint positions in real-time. To calibrate user-piano interaction, we leverage "ArUco" markers to compute homographies between uv piano texture space and xy camera image space. Finally, we implement a simple threshold-based key press detector and play sounds from a library of piano note recordings. Our system runs at 12 frames-per-second on commodity laptops, achieving 81% *accuracy (within an error threshold of 20 pixels) on hand keypoint detection tasks.*

1. Introduction

1.1. Problem Statement

Real-time human pose recognition has a variety of applications, including virtual and augmented reality, medicine, robotics, and many other fields. In particular, hand tracking is a necessary implementation for many human-computer interfaces. Although early hand pose recognition systems tracked human hand motion through data gloves or colored markers, purely vision-based methods have recently become more viable [7].

We have chosen to investigate the musical applications of real-time human hand pose recognition, specifically with respect to augmented reality (AR) piano playing. Augmented reality provides a platform for beginner piano players to learn how to play piano with more targeted teaching approaches, and without the need to purchase an expensive musical instrument. In addition, AR piano playing may allow beginners to practice piano without torturing the ears of the rest of us humans. Piano playing is a straightforward iteration of the hand pose tracking problem, one that can be extended to other, more complex problems; in other words, piano playing is a good entry point into this problem.

1.2. Plan for Approach

To develop an AR piano playing application, three main components are needed: (1) Hand pose recognition, (2) augmented reality calibration, and (3) key press detection. The final system takes as input a live video stream from a webcam. The input is used to determine hand keypoint positions and to instantiate an AR piano, using specialized markers for calibration. The hand coordinates and the keyboard positions are then used in the state estimation step. During state estimation, the hand pose is transformed from camera image space onto the piano texture space and then used to detect piano key press (which finger is pressing a key, and which key is pressed). The end-to-end flow of the system is shown in Figure 1.

2. Background/Related Work

In this section, we describe the relevant background as well as our initial explorations in hand pose recognition, augmented reality, and piano playing interfaces.

2.1. Hand Pose Recognition

Previous works in human hand pose recognition that we explored are Convolutional Pose Machines, OpenPose, and MediaPipe Hands. All of these methods are open source systems which have been trained on large annotated pose datasets, including the MPII Human Pose Dataset [10] and the CMU Panoptic Studio Dataset.

2.1.1 Convolutional Pose Machines

Convolutional Pose Machines (CPM) is an open source machine learning framework produced by The Robotics Institute at Carnegie Mellon University. CPM harnesses GoogLeNet to analyze image sensor data and produce human pose estimates by generating response/belief maps of the key points in each image. The CPM has been shown to have high estimation accuracy, enhanced low level feature extraction, and reduced parameters, making the model efficient and accurate.

Unfortunately, in the process of trying to implement this model, we discovered that the code and dependencies are written in Python 2. Since Python 2 is no longer supported

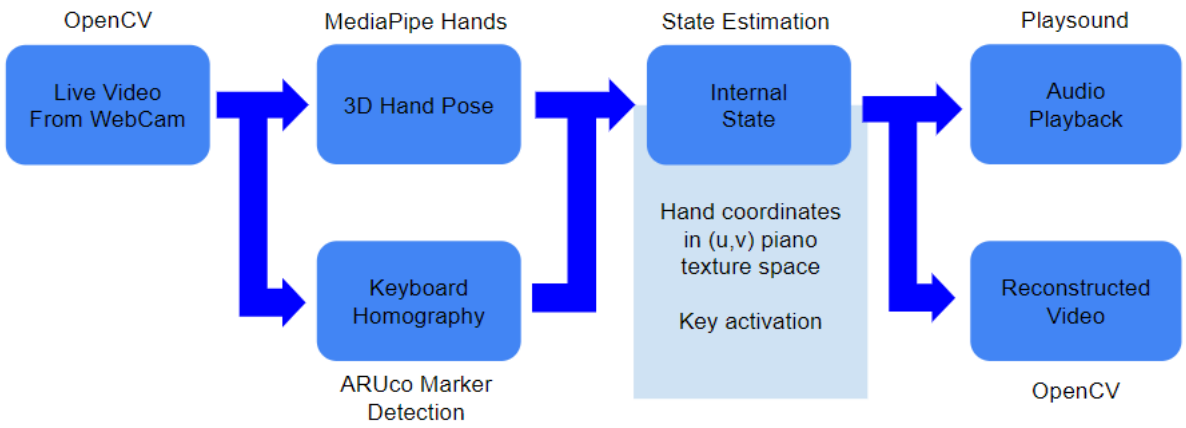

Figure 1. System flowchart

and the versions of this code are therefore not stable, we decided not to train the CPM model.

2.1.2 OpenPose

OpenPose [5] is the first open-source system for multi-person detection of human body, hand, and face keypoints. Given an input 2D RGB image, OpenPose uses a CNN to first predict 2D vector fields encoding the location and orientation of limbs (denoted "Part-Affinity Fields"), before predicting final keypoint confidence maps from this intermediate representation. In many ways, OpenPose represents follow-up work to Convolutional Pose Machines, with both projects having been produced by Carnegie Mellon University and relying on the CMU Panoptic Dataset. Here, we focus specifically on its hand detection capabilities. As part of our initial exploration, we built OpenPose on Mac (CPU only) and ran it on the included dataset images, along with "in-the-wild" images from online.

While OpenPose achieved reasonable (qualitative) keypoint prediction on dataset images (e.g. "Baseball"), it did not successfully generalize to all in-the-wild images (e.g. "Piano"), as shown in Fig. 2. In general, OpenPose pipeline is not well-suited for cases where only part of the body (e.g. arms of a pianist) is visible. Additionally, we found the runtime on CPU to be impractically slow: a single frame of "Baseball" took nearly 30 seconds to process. Overall, though very well-documented, OpenPose is poorly suited for our task of real-time piano playing in AR, as the focus on multi-person detection introduces unnecessary overhead for single-person hand detection.

2.1.3 MediaPipe Hands

MediaPipe [9] is an open source machine learning library provided by Google, specializing in processing live and

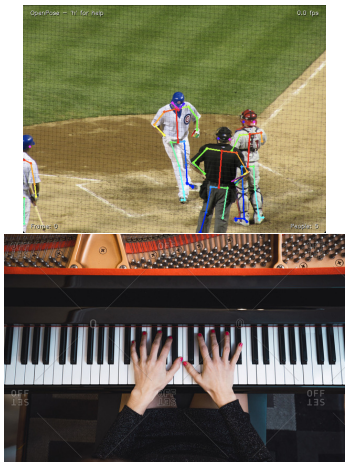

Figure 2. Initial results from OpenPose. Model performs well on included dataset images (top) but fails to generate output when full body is not included (bottom)

streaming media. A part of the MediaPipe library, MediaPipe Hands, achieves real time hand motion recognition (∼30fps on CPU) by using the costly detection process only when needed (e.g. when a new hand appears) and using a less computationally expensive feature tracking algorithm to track the keypoints.

The outputs available from this algorithm are: (1) handedness (right hand or left hand), (2) 3D coordinates of each of the 21 keypoints in the image frame, and (3) estimated 3D coordinates corresponding to each keypoint in the world frame, with the world origin placed at the approximate geometric center of the hand. The 3D world coordinates have high accuracy: there was an error of ∼1cm between the finger length calculated from the 3D keypoints generated by MediaPipe and the finger length measured in the real world.

Figure3 shows two screenshots from real-time video detection. The keypoint detection accuracy is high when oc-

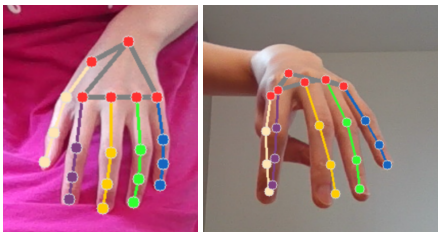

Figure 3. Preliminary results from MediaPipe Hands. Model performs well when there are no occlusions (left) but fails to accurately label the position of the thumb during cross over motion (right)

clusions are small. However, the detection struggles to accurately identify the position of the thumb when there are occlusions. This is an aspect that must be improved because occlusion of the thumb is common in piano playing when finger cross overs occur.

2.2. Augmented Reality

ARKit, ARCore, and OpenCV are among the most widely used AR application development tools. In this section, we discuss the characteristics of each of these tools.

ARKit [3] is a toolkit created by Apple to assist developers in creating AR applications, especially for iOS devices. ARKit's main advantage is it's ease of use. However, ARKit only supports development for iOS applications on relatively new Apple devices unless it is used along with Unity's AR Foundation framework [1]. This is inconvenient for this project because the language officially supported with Unity is C#, while MediaPipe Hands library is only available with Python, C++, and JavaScript.

ARCore is an augmented reality SDK developed by Google [2]. Unlike ARKit, it supports multiple environments including Android, iOS, Unity, and Unreal Engine. As can seen by the supported platforms, ARCore is commonly used in 3D game development, and features unique functionalities like predicting lighting from camera input and applying natural lighting to AR objects generated in the screen.

Another popular tool used for AR is OpenCV. While ARKit and ARCore are SDKs built specifically for AR application development, OpenCV's functionalities are more broad in scope. Consequently, while there are fewer prebuilt, AR-specific functionalities, the major advantage of using OpenCV is its ability to be highly customized and its ease of incorporation into a variety of development environments.

2.3. Piano Key Detection

Piano key detection has been explored in several works, often in the context of end-to-end systems for virtual piano playing. Most notably, Barehanded Music [8] presents a data-driven approach for tracking fingers and detecting presses; conveniently, it also targets AR applications (but uses a depth sensor in addition to standard rgb cameras). The key press detection itself is performed by an SVM classifier that takes, as input, the trajectories of hand joints. To train their classifier, the authors created a custom dataset of 7200 rgb-d images, consisting of the most common finger articulations for piano playing. Their proposed data-driven approach achieves over 90% accuracy on finger press detection tasks (with a small training set), but unfortunately, the authors did not make their dataset available online.

3. Technical Approach

In this section, we discuss the approaches we took to solve each of the three subproblems: hand pose recognition, augmented reality calibration, and piano key press detection.

3.1. Hand Pose Recognition

In our initial explorations, we discovered that OpenPose is too computationally expensive to run real-time on CPU. It also focuses on full body pose recognition, requiring additional training and manipulation to apply to our problem. Although both MediaPipe and OpenPose struggled with occlusions, we selected MediaPipe as our hand pose recognition method due to the availability of a package specific for hand recognition, making additional training unnecessary. MediaPipe's computational efficiency also makes real-time tracking possible, and is thus the optimal method for our piano playing application.

Specifically, MediaPipe hands leverages a convolutional feature extractor (representation learning), following by separate branches to predict 3D locations of keypoints along with presence of hands. The full network (during run-time) is shown in Figure 4. During training, the network takes in rgb images of hands labeled with keypoint locations, generated from both real-world photographs and photo-realistic synthetic renderings in order to improve generalizability.

In practice though, the network details are abstracted away by the MediaPipe API. We call the API to retrieve predicted 3D locations of 21 hand keypoints, which represents an update to our pipeline's internal state. These 3D locations are then processed for the end-goal of piano key press detection, as well as to be displayed on the screen.

3.2. Augmented Reality

Although ARKit and ARCore are both attractive options for AR application development, these tools are most beneficial in developing applications that makes extensive use of 3D objects and device motion. Because the only AR object currently required for our project is a stationary, flat keyboard placed onto a tabletop, the advanced features available in ARKit and ARCore are not necessarily needed. Ad-

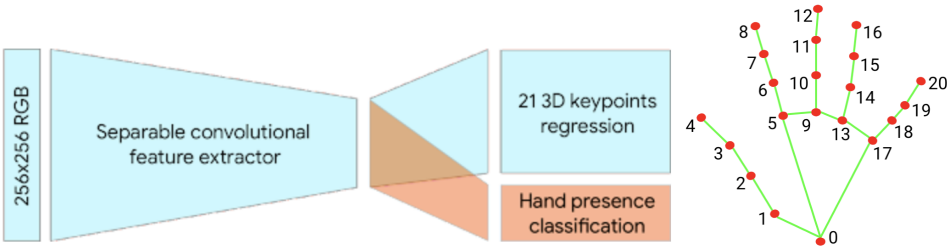

Figure 4. MediaPipe Hands architecture details, (left) the network features a convolutional feature extractor, (right) the 21 hand keypoints regressed by the network.

ditionally, MediaPipe and OpenCV are both supported on Python, making OpenCV an ideal choice for out application. Thus, Python OpenCV's ARUco module is selected as the method to develop the AR component of our system.

The first step is piano initialization. First, the user places a calibration paper on a dark colored table. The paper has four ARUco markers on each corner, and these four markers define a bounding box where the AR piano is placed. Four corners of the bounding box correspond to each of the four corners of the piano image, and these correspondences are used to compute a homography (projective transformation) matrix that transforms the rectangular piano image to fit the bounding box as it appears in the webcam image frame (i.e. the homography matrix H transforms the piano texture frame to the webcam image frame).

$$\begin{bmatrix} t\,x_{webcam} \\ t\,y_{webcam} \\ t \end{bmatrix} = H \begin{bmatrix} u_{piano} \\ v_{piano} \\ 1 \end{bmatrix} \tag{1}$$

where t is the homogeneous coordinate to be divided by after applying the transformation. Using the computed homography, an AR piano appears in the webcam image frame. Once this step is complete, the location of the piano remains stationary, and the calibration paper is removed. The homography matrix is the second input to the piano key press detection algorithm.

In order to overlay the user's hands over the AR piano, we assume that there is enough brightness contrast between the user's hands and the flat surface on which the user places the calibration paper. To identify the user's hands, Otsu's method [4] for binary segmentation is used. This method calculates the foreground and background variances to find a threshold using the following steps: (1) calculate weighted mean for background and foreground pixels, (2) calculate pixel variance and then (3) take the sum of the forward and background variances multiplied by associated weights. These equations are iterated through to find the maximum variance $\sigma^2(t)$. To minimize intra-class variance, Otsu's method maximizes inter-class variance. The threshold values computed by Otsu's method are then used to generate a mask segmenting the user's hands from the rest of the webcam image. Using this mask, along with the webcam frame

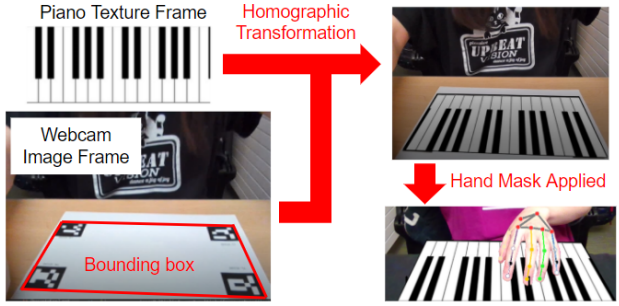

Figure 5. AR framework. Homographic transformation from rectangular piano texture frame to webcam image frame, AR piano initialization, and hand masking

position of the AR piano, the AR piano is only displayed in the locations where the piano position and the hands do not overlap, thus, allowing the user's hands to be displayed over the piano. In this project, we use a black tabletop for testing, but the program can be manipulated to support white tabletops by inverting the mask. The process and results of the AR component are summarized in Figure 5.

3.3. Piano Key Detection

Initially, we sought to implement the data-driven approach for piano key press detection proposed in [8], but due to inaccessibility of a dataset, we settled on a simple threshold-based approach in the piano texture space. Namely, given the homography matrix H from the ARUco marker calibration, we invert H to compute the inverse projection.

$$\begin{bmatrix} t\,u_{piano} \\ t\,v_{piano} \\ t \end{bmatrix} = H^{-1} \begin{bmatrix} x_{webcam} \\ y_{webcam} \\ 1 \end{bmatrix} \tag{2}$$

where once again, we divide by the homogeneous coordinate t afterwards. We compute the projection of all five predicted fingertip keypoints onto texture space and visualize the result as five colored circles overlaid on the piano texture. An example projection is shown in Figure 6.

For the piano key press detection itself, we qualitatively observe that a 'downwards press' motion translates well to

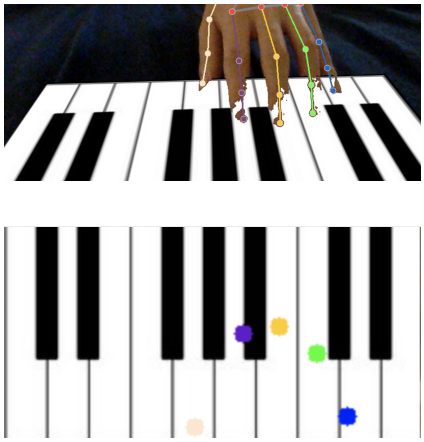

Figure 6. Fingertip keypoint projection from image to texture space. (top) webcam image showing all five fingers laid over AR piano, (bottom) corresponding projected fingertip locations overlaid on piano texture.

Figure 7. Varying threshold lengths of 10, 20, 30, 40, 50, 60, 70, 80, and 100 displayed for scale on CMU Panoptic dataset image. Threshold of 10 has much smaller tolerance for errors.

an increase in the v (vertical) coordinate in uv texture space. Thus, we track sharp increases in the v coordinate to detect whether a 'key press' state event should be triggered. For simplicity, if v rises above a certain threshold value (in practice, we use $v \geq 0.5v_{max}$), we trigger a 'key press'. To prevent the same note from being rapidly and excessively triggered, we require the finger to return back to resting state ($v < 0.5v_{max}$) before that note can be played again.

Since we have a direct map of fingertip keypoint position to note on the piano, once a key press is detected, we simply play the corresponding note. We instantiate a library of piano note recordings (from open source repository freesound.org [6]) to be played during runtime. Additionally, to prevent the audio from lagging video, we leverage separate audio and video processing threads.

4. Experiments

To validate our implementation, we ran quantitative experiments measuring the accuracy of predicted fingertip keypoint locations, along with qualitative demos suggesting how a user might interact with our system. The results are provided in this section.

4.1. Quantitative Results

Hand pose recognition performance was evaluated by calculating what percentage of fingertip keypoints correctly identified the true position of the fingertip. The estimated keypoint is considered correct if the keypoint lies within a certain threshold of the ground truth labels from the CMU Panoptic dataset.

Although the dataset for MediaPipe Hands has not been published, the CMU Panoptic Hands dataset is a labelled dataset corresponding to the outputs of MediaPipe Hands. In order to ensure that the hand recognition algorithm encapsulated our required precision, we ran the MediaPipe algorithm on 15,000 images from the CMU Panoptic dataset. These images encapsulated hands in a variety of conditions, including varied light, poses, and skin color. The CMU Panoptic Hands dataset includes labelled predictions in the form of 21 keypoints for each hand. After running MediaPipe Hands on each image, we extract which hand the outputted keypoints belongs to and calculate the mean absolute error between the predicted keypoints and ground truth values. For each keypoint, we classify the point as either (1) accurate or (2) inaccurate based on whether the calculated mean absolute error lies below a determined threshold and track the number of inaccurately predicted keypoint. We used this number of calculate the accuracy of MediaPipe Hands as a keypoint predictor. A threshold of 30 pixels would indicate the point lying within an error rate of one key, which is the largest tolerable error for our program.

Because the piano that are demoing has eight keys, each key is approximately the length of the 30 pixel threshold. However, webcam fidelity can vary and the end goal for this system would be to model a larger piano on this screen, meaning smaller keys and thus less room for error.

With an error threshold of 30, we reach a keypoint accuracy of 84%, meaning that 84% of predicted keypoints lie within 30 pixels of the ground truth pixel, and 16% of predicted keypoints lie outside the 30 pixel radius. Even an error threshold of 20 has an acceptable accuracy percentage of 81%, which leads to the conclusion that in real time this hand pose recognition model will serve our purposes for gesture recognition.

To evaluate the true performance of the model, we calculated the accuracy percentage based on varying thresholds.

Figure 8 shows the accuracy percentages as the threshold changes and Figure 7 visualizes the threshold sizes on

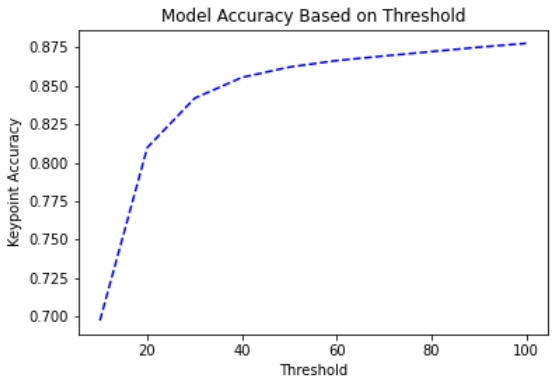

Figure 8. Effect of threshold on MediaPipe keypoint accuracy, which shows how close to ground truth most predicted points lay.

an example test image. According to this data, as long as the threshold is greater than or equal to 20 pixels (with a corresponding accuracy of 81%), this model would likely be a good fit for our piano key detection use.

However, as is demonstrated in Figure 7, the hands used to evaluate this dataset are fairly small, making the model predictions more inaccurate than when used to find the keypoints for hands that take up most of the image space, as is the case in our particular usage case.

4.2. Qualitative Results

Qualitative performance of finger tip tracking is evaluated through an application demo using a realistic device and camera angle. Figure 10 shows excerpts from the demo. The position of the middle finger tip in both the webcam image frame and the piano texture frame (displayed on top left corner of each image) are marked with a red dot. For each of the image shown, hand pose detection correctly determines the position of the middle finger tip in the webcam image frame, and the corresponding transformed coordinates (in piano texture frame) lies on the correct key.

Finally, to demonstrate the end-to-end capabilities of our implementation, we task a user with playing a simple melody. We perform a trial run with audio feedback disabled, such that the user can only rely on their intuition to play the melody. To simplify the task further, we disable all key press activations except for the middle finger. The resulting audio after pitch tracking is shown in Figure 9. Even with audio feedback disabled, the user played 12/13 notes correctly – the only error being a missing note between the 7th and 8th note. In general, the system is able to accurately identify which piano key the finger is hovering over, but the (threshold-based) key press detector itself remains finicky. See the supplementary material for the full video with audio.

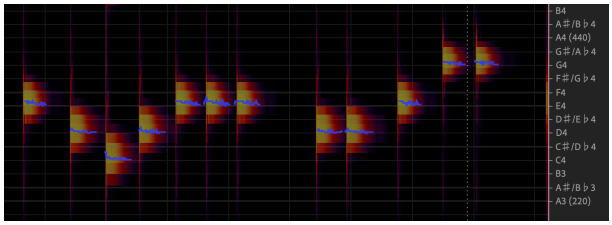

Figure 9. Pitch tracked result from playing a simple melody (Mary Had a Little Lamb) without audio feedback.

5. Conclusion

In this project, we developed a prototype of an AR piano playing application using real-time 3D hand pose tracking with MediaPipe Hands, AR with OpenCV ARUco modules and piano key press detection using a threshold-based detector.

Our AR piano playing prototype achieved 81% accuracy at recognizing each hand keypoint. We have determined that for this specific usage, especially with respect to teaching beginner piano (which requires only a shortened version of the full keyboard), this accuracy is a decent approximation of a hand's real coordinates.

With advancements in motion-controlled devices and technologies, real-time hand pose tracking, especially methods that do not require special equipment like depth sensors, has gained significance. However, most existing works in hand pose recognition that only require RGB information are too computationally expensive for real-time tracking. Although MediaPipe Hands achieves real-time tracking from RBG input, accuracy tends to reduce when hands are at certain angles or when fingers are unnaturally posed, despite being trained specifically on hand poses. Overall, real-time pose recognition in the absence of depth information is a still evolving area of research.

Specifically, MediaPipe and OpenPose have been able to achieve better accuracy due to the large amount of labelled data that each of these models has access to. Finding an unsupervised learning method to improve keypoint prediction in computer vision would go a long way.

Future ideas to improve AR piano playing include: adding support for chord playing (play multiple notes at once), incorporating visual feedback (animation to show which key is depressed) for a more immersive experience, and offering a way to import sheet music into the AR environment.

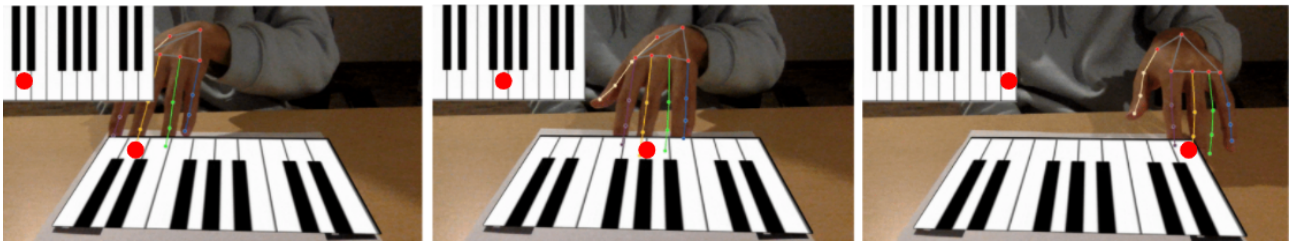

Figure 10. Results of finger tracking: middle finger keypoint location in webcam frame and piano texture frame when finger is placed on keys D (left), G (middle), and C (right)

References

[1] Augmented reality game design software for apps | unity. [Online]. Available at https://unity.com/unity/features/ar. [Accessed: 18-Mar-2022].

[2] Build new augmented reality experiences that seamlessly blend the digital and physical worlds | arcore | google developers. [Online]. Available at https://developers.google.com/ar. [Accessed: 18-Mar-2022].

[3] Dive into the world of augmented reality. [Online]. Available at https://developer.apple.com/augmented-reality/. [Accessed: 18-Mar-2022].

[4] S. Bangare, A. Dubal, P. Bangare, and S. Patil. Reviewing otsu's method for image thresholding. *International Journal of Applied Engineering Research*, 10:21777–21783, 05 2015.

[5] Z. Cao, G. Hidalgo Martinez, T. Simon, S. Wei, and Y. A. Sheikh. Openpose: Realtime multi-person 2d pose estimation using part affinity fields. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2019.

[6] Jobro. Pack: Piano ff (freesound). [Online]. Available at https://freesound.org/people/jobro/packs/2489/. [Accessed: 18-Mar-2022].

[7] H. Liang, J. Wang, Q. Sun, Y.-J. Liu, J. Yuan, J. Luo, and Y. He. Barehanded music: real-time hand interaction for virtual piano. In *I3D Symposium*, 2016.

[8] H. Liang, J. Wang, Q. Sun, Y.-J. Liu, J. Yuan, J. Luo, and Y. He. Barehanded music: Real-time hand interaction for virtual piano. In *Proceedings of the 20th ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, I3D '16, page 87–94, New York, NY, USA, 2016. Association for Computing Machinery.

[9] C. Lugaresi, J. Tang, H. Nash, C. McClanahan, E. Uboweja, M. Hays, F. Zhang, C.-L. Chang, M. G. Yong, J. Lee, W.-T. Chang, W. Hua, M. Georg, and M. Grundmann. Mediapipe: A framework for building perception pipelines, 2019.

[10] L. Pishchulin, E. Insafutdinov, S. Tang, B. Andres, M. Andriluka, P. Gehler, and B. Schiele.