Gaurav Joshi, Akash Kedia, Ash Parasa, Parth Ray

Prof. Ventura

CPE 428

Hand Piano Final Project Report

Introduction

For the course project, we decided to implement a hand piano. A hand piano is a real instrument traditionally known as a Kalimba. It originates from Africa. Our goal was to combine the play style of a Kalimba with the sounds of a real piano to create a virtual hand piano that tracks a person's hands using a camera to play notes specific to certain fingers when they are put down. We wanted to help solve the problem of making music education both more accessible and higher quality. Computer vision represents an avenue that may be used to achieve that, by bringing software-based experiences to students at virtually no cost. This problem is interesting because we want to see how close we can get to mimicking the experience of playing a real piano, short of touching actual keys. There are numerous ways to expand and add features to make the experience more immersive, like adding a 61-key piano overlay to the screen and playing sounds based on which virtual keys are touched. That is not in the scope of the project for this quarter, but it represents something to strive for and makes the project interesting. The project is challenging because there are a lot of objects to detect and keep track of, from the hand to each individual finger. Then we have to monitor the position of the finger to detect a keypress. There are also issues to deal with like occlusion due to fingers crossing or the background being noisy. Our method involves using a pre-made hand detection model from the mediapipe library. This is a high-quality model that gives us all the keypoints we need. Since we are using this model, our job will be to work on the keypress detection to make that as accurate as possible.

Background

One blog post we were inspired by is a post on Towards Data Science called “Air Piano using OpenCV and Python.” This author was inspired by a similar purpose as ours: to help with music education. The author’s sister was struggling to learn piano via Zoom, so the author wanted to create a tool that would be easier to use with the piano teacher. The author used OpenCV to detect the hand pose from the camera and another library called PyAutoGUI to interface with another piano application. Although this approach seemed to work well, we wanted to produce the sounds directly from our application without having to rely on another application. We also wanted to use some more advanced pre-made models rather than start with something rudimentary that would need more time to become a viable product. We took the author’s suggestion to use Mediapipe, a collection of ML solutions for computer vision purposes.

Another source we referenced was a GitHub repository by SeanTKeegan called “OpenCVPiano.” This author also had a much different implementation from what we were aiming for. The author made it so that the user raises their arms into the space of the note, rather than individual fingers. Their approach also results in a 6 key keyboard. We prefer to make it closer to the real experience of playing a piano by using individual fingers. We did like the various features of the author’s piano, like switches to control sustain and pitch with the arms as well. The approach is also very simple, relying mostly on OpenCV to do the detection and performing image subtraction to detect when a particular key or button is “hit” by an arm. This approach would have been feasible for us, but we really wanted to use the more complicated finger keypoint detection. We did like the pitch/sustain features and would want to incorporate those if we have time.

Data and Methods

Our system uses OpenCV, Mediapipe, and playsound. OpenCV is used to read in a feed from a live camera, read a video file, or read an image file. For each image in the video, we flip it and convert it to RGB to prepare it for mediapipe. After mediapipe returns the keypoints, we use OpenCV to draw them on the image.

Mediapipe is an advanced collection of prebuilt ML models. It includes a hand and finger tracking solution. We simply setup the hands object with a number of hands to track and confidence levels. Then we can pass it frames and it will identify the keypoints. We draw the returned keypoints on the image and use them to determine what actions the user is taking. It is up to us to determine when a finger has been bent to indicate that the user is trying to play a note. We do this by tracking the y position of a finger with respect to its last known location and the location of the thumb (which is fixed in the y-direction relative to the rest of the fingers). By relating the position to these two known points, we can adjust the sensitivity of the sound activation. With a low difference, we can play quicker but there may be some false presses detected. With a high difference, there will be fewer false presses but we will have to move the finger further, resulting in slower playing. The thumbs have a special case where we look at the difference in the x position to judge whether its been pressed or not.

Playsound is used to play a sound through the computer's speakers when a particular note is activated. It is a simple library otherwise. A benefit is that a prior sound does not stop playing when another sound is activated, mimicking an actual piano.

Our solution does not require any training data as we are using a pretrained model. We use data only to evaluate the solution. This data consists of live feeds from the system camera and pre-made photos and videos.

Evaluation

Our evaluation criteria revolved around the precision and accuracy of the finger-press detection. Evaluating these metrics proved to be quite difficult due to the number of false positives detected by the Mediapipe library depending on the person using the program. For example, someone whose fingers move conjointly would trigger the finger detection when attempting to evaluate discrete finger-presses. As a result, we settled on counting the total number of finger-presses detected by our program, rather than matching a sequence of detections, and comparing these totals to the ground truth predetermined in each of the three test videos. We decided to evaluate these totals using both micro-average precision and mean percent error metrics to see where improvements could be made. Micro-average precision measures the number of fingers where the total number of presses detected equals the ground truth of presses divided by the total number of fingers. Here, higher is better. Mean percent error defines, on average, how far the detected number of presses per finger was from the actual number of presses per finger. Here, lower is better.

Results

Given that our first iteration on this project was quite successful in terms of achieving the goal we set out to reach, our second iteration of the project mostly involved tweaking hyperparameters. In the first iteration of our project, we used a detection and tracking confidence

of 0.90 believing a stricter confidence would prevent false unintentional presses from being detected. After doing some extensive tuning, we found that a 0.50 confidence did a better job at detecting intentional presses than a higher confidence did of preventing unintentional presses. We created a set of three videos, each of which involved finger presses simultaneously or discretely at different tempos. The results of these videos can be found below:

Video 1: Discrete finger presses at a low tempo

First Iteration (0.90 Confidence):

Micro-average precision: 80.0%

Mean Percent Error: 20.0%

Second Iteration (0.50 Confidence):

Micro-average precision: 80.0%

Mean Percent Error: 20.0%

Video 2: Simultaneous finger presses at a slightly faster tempo

First Iteration (0.90 Confidence):

Micro-average precision: 60.0%

Mean Percent Error: 20.0%

Second Iteration (0.50 Confidence):

Micro-average precision: 60.0%

Mean Percent Error: 20.0%

Video 3: Discrete and Simultaneous finger presses at a faster tempo

First Iteration (0.90 Confidence):

Micro-average precision: 30.0%

Mean Percent Error: 26.4%

Second Iteration (0.50 Confidence):

Micro-average precision: 50.0%

Mean Percent Error: 11.4%

Our results were not perfect, but since we were generally satisfied with the way our model performed in real-time, we did not redesign the whole program on our second iteration to account for factors such as conjoint presses and false presses. Given more time, we would have liked to rework our model possibly using a CNN trained on our own data.

Outlook

Our expectations for this project were at minimum to get the hand detection working. We found this prospect slightly daunting as we thought we would have to do everything from scratch. Using Mediapipe to handle that aspect was a strategic choice as it freed us up to focus on the mechanics of fingerpresses. At that point, our expectations became to have a working, playable piano. We achieved those expectations. Our current solution supports playing concurrently using all fingers of the hand.

We learned a few things during this project. We learned about the libraries Mediapipe and playsound. Mediapipe seems to be a great library for pre-made ML solutions to use for computer vision purposes. We find it useful to know about this library for other purposes in the future. We understood that the Mediapipe model is a pipeline of hand detection, palm detection, and finger detection models. Playsound also seems like a nifty library to know when it comes to playing sounds in Python for any reason. These are great tools for prototyping or more advanced purposes. We were also able to explore OpenCV a bit more, as at first we were looking to develop everything from scratch. We had to dive into the concept of feature-matching a bit more,

as hand/finger detection relies on that concept. It is not enough to simply identify something as a finger, because we need to know the position of each specific finger and keep track of it through space. With an understanding of this concept, we could correctly interpret the keypoints that Mediapipe returned.

If we could work further, we would add many things. Some visual aspects include a piano overlay that highlights the notes being played. We have also discussed the idea of being able to interact with the piano overlay but are unsure if it may be an inferior experience. We also want to add the pitch/sustain features. If possible, we want to add an intuitive way to play a full piano, starting with 61 keys.

References

Masud, U. (n.d.). *Air Piano using OpenCV and Python*. Towards Data Science. Retrieved Oct 27, 2021, from https://towardsdatascience.com/air-piano-using-opencv-and-python-298cb22097d9

SeanTKeegan. (n.d.). *OpenCVPiano*. Github. Retrieved Oct 27, 2021, from https://github.com/SeanTKeegan/OpenCVPiano