Predicting the String Played by a Cellist through Edge Detectors, Hough Transforms, and k-Nearest Neighbors

Kenny Huang and Iroha Shirai

December 14, 2021

1 Motivation/Goal

This project was motivated by our backgrounds in music and our interest in combining this with the concepts that we learned throughout the course. We wanted to explore the topic of object recognition, so we thought about how we could detect the sounds being produced on an instrument just from the visual video of the instrument being played.

We had originally started off with the hope that we could predict the note that a violinist was performing, but found that it was much harder to predict this because of the limitations in how much of the violin's 'face' could be seen with a given camera position – given that the violin is often held facing the ceiling at an angle from the camera. We also found that it was challenging for our purposes because it is much harder to detect the location of where the fingers are placed on the fingerboard, given the size of the instrument and issues with occlusion.

With these challenges, in addition to the short time frame provided to work on this final project, we decided upon our final goal: to predict the string that a cellist is playing on based on only visual data using feature detection and a k-nearest neighbors model.

2 Related work

Some related work to our project topic includes the Visual Music Transcription of Clarinet Video Recordings Trained with Audio-Based Labelled Data paper which focuses on the automatic process for transcribing the notes that a clarinetist is playing based on the visual knowledge provided in the video of a person playing the clarinet [1]. The Sound of Pixels project is also related to our work in that it uses only a visual of instruments being played to disentangle the associated sounds and ultimately determine which sounds are being produced by each individual instrument [2].

Our work is unique from these related works because not only does it focus on a different instrument — a cello— but it also focuses on determining the string that is being played based on the angles of the lines that represent the bow and fingerboard.

3 Approach

Our main approach to reach our goal is to:

1. detect the lines corresponding to the bow and fingerboard of the cello
    (a) optimize our custom Hough transform function
2. calculate the angles of the two lines detected (denoted as theta)
3. train a k-nearest neighbors model on the two theta values
4. use the model to predict the string being played in unseen data

As part of this enumerated approach, we needed to create our own dataset of labeled images so that we could create the necessary data to train the model used in step 3.

4 Design Decisions

The key design decisions that we had to make in order to reach our goals included:

4.1 Framing around the cello

We had originally thought to use the entire video frame as the input of our model. However, we quickly found that this was too ambitious of a goal because the noise in the background often made it impossible to accurately detect the features that we needed. Thus, we decided to limit the input of our model to videos where the majority of the video frame is composed of the cello.

4.2 Hough Transform

We assumed that the white hair of the bow and the strings over the fingerboard would be straight lines so that we could use a Hough transform to find them. Using the Hough transform was an important design decision because it allowed us to easily and accurately find the bow and fingerboard, although runtime concerns did limit the resolution of the video that we could look at.

4.3 k-Nearest Neighbors

There are only four possible predictions when predicting the string being played: A, D, G, or C — the 4 strings on the cello. In addition, we have two values that we are able to acquire: the theta of the bow detection line and the theta of the fingerboard detection line. Given these conditions, we felt that k-Nearest Neighbors was the most efficient classification model to use. We were only considering 2-dimensional data, so we did not have to worry about the vulnerability of the KNN method toward the curse of dimensionality.

5 Implementation

5.1 Dataset

We were not able to find any datasets composed of photos of people playing the cello, let alone ones with each photo labeled with what string is being played. Therefore, we decided to create our own labeled dataset.

In order to do so, we manually collected photos by going to YouTube and looking up keywords including 'cello', 'cellist', and 'cello player'. Our method required us to have a good view of the cello, making sure 1) that most of the cello body was in view, 2) that the cellist did not move around too much during the performance, and 3) that the video was taken with the camera as close to directly facing the cellist as possible. Therefore, the videos that we could use were quite limited. However, we managed to collect 132 photos in total, all of which by taking screenshots of the YouTube videos at certain frames where we could confidently determine what cello string was being played. (The YouTube videos used for this data collection process are listed at the end of this paper.) The screenshots were taken not of the entire video frame, but rather as a rectangle around the cello, including the fingerboard and the bow.

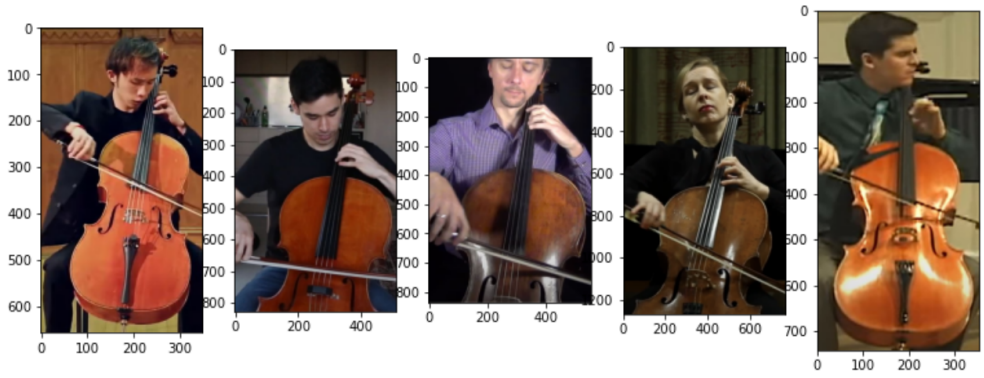

Figure 1: Example images from our test set

5.2 Detect the lines corresponding to the bow and on the fingerboard of the cello

To detect the fingerboard and bow, we first noted that both of these features are straight, extended lines that are roughly vertical and horizontal respectively. We converted the input image to grayscale and then applied a custom modified edge detector to outline the contours of the image from the x-direction. At this point, the brightness of each pixel represented its change from its neighbors on its left and right in the original image.

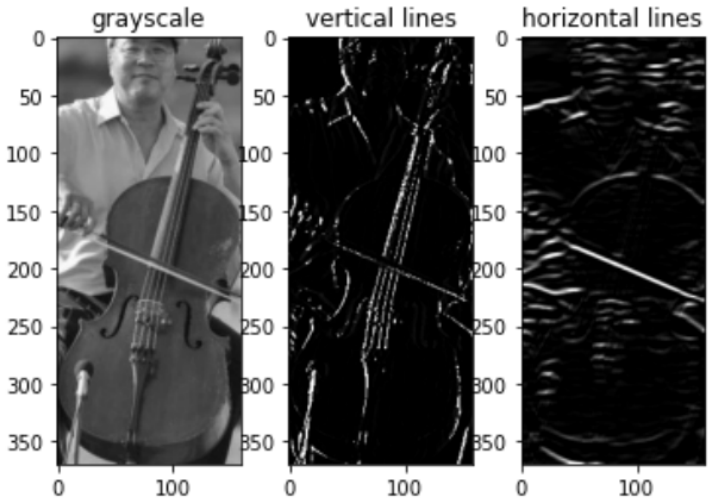

Figure 2: Edge detectors applied to a test image

From here, we assumed that the strings on the fingerboard would be straight lines and used a Hough transform to find the lines that contained the most pixels above a predetermined threshold. In theory, the pixels corresponding to the fingerboard in the original image both differ from their neighbors and lie on a straight line. Therefore, the Hough transform should output a line parallel to the axis of the fingerboard. Similarly, we can apply an edge detector from the y-direction and then use a Hough transform to find the white hair of the bow. Again, the pixels of the bow contrast with the pixels directly above and below and lie on a straight line, so the Hough transform should output a line through the hairs of the bow.

We were able to implement the above functions in part_detection.py, one of our python files. The width and standard deviation parameters of the two edge detectors were then decided through inspection. As we hoped, the model was able to successfully detect the bow and fingerboard of the cello on most of the frames in our test video as well as on 108 photos of the 132 photos collected in our dataset, leading to 82% accuracy on detecting the lines that represent the bow and fingerboard in an image.

5.3 Optimize our custom Hough transform function

One issue that we encountered was that the Hough transform was incredibly slow when we ran it with the sensitivity that we needed to accurately detect the features. Unfortunately, the premade function in cv2 was not working for us, so with help from the OpenCV documentation [3], we wrote our own custom Hough transform script, which lacked optimizations that the cv2 version may have had. In the first version of our code, a vote was cast for the line corresponding to each possible

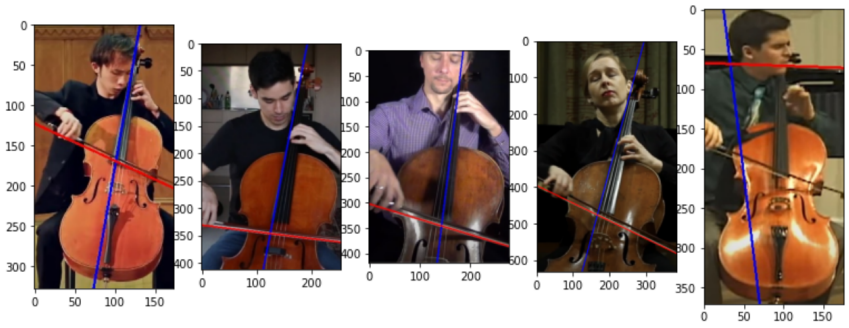

Figure 3: Detected bow and fingerboard lines on test images

angle in $[0, \pi]$ for each pixel in the image. Because we wanted the resolution of our output angle to be one degree, this meant that our function had a runtime of $\sim 180mn$, which took more than 300 seconds for 60 frames (2 seconds of video). As a result, testing became incredibly time consuming, so we sought to optimize our function for our purposes. Creating our own function actually worked to our advantage here by allowing us more flexibility.

Specifically, we had previously made the assumption that our input would be pre-processed such that the majority of the video frame is composed of the cello. This specification allowed us to assume that the bow and fingerboard would only be in certain sections of the image, allowing us to ignore large swaths of pixels. In addition, we also recalled that we chose to analyze the cello because we could assume that between frames, the fingerboard and bow would only change by a negligible amount. Thus, by passing in additional parameters reporting the output values of the previous frame, we were able to drastically reduce the number of possible lines to try. Now, we were able to compile 600 frames' worth of video in less than 120 seconds, or about 5 fps.

One final issue that we ran into at this stage was the inflexibility of the model when it became wrong. Although adding the information about previous frames greatly stabilized our model and reduced the noise of the predictions, it also prevented the model from being able to correct itself back to the ground truth if it did somehow go off course. To remedy this, we allowed it to "reset" every second and compute its outputs without previous information. Although this marginally increased runtime, this made our model more robust to both random noise and catastrophic failure.

5.4 Calculate the theta values related to the lines detected

Once our model was able to detect the two features in the images in our dataset, we sought to use the relative angles between the two lines to make our final string predictions. To do so, we ran our model on our dataset of test images and extracted the angles of the detected lines from the images that we manually verified were detected properly. To address the discontinuity of the bow angles at 0 and π, we transformed it to be centered around $\pi/2$ instead.

5.5 Train a k-nearest neighbors model on the two theta values

Using the resulting dataset composed of only the properly detected images, we generated a k-nearest neighbors model on the two theta values calculated (one for the theta of the fingerboard, and one for the modified theta of the bow). When training, we split up our dataset so that 80% of the dataset was used for training and 20% was used for testing. Using leave-one-out cross-validation, we found that k = 11 yielded the best results, with validation accuracy of 70%.

	validation accuracy		validation accuracy		validation accuracy
1	0.512	6	0.651	11	0.698
2	0.605	7	0.663	12	0.686
3	0.581	8	0.674	13	0.674
4	0.628	9	0.640	14	0.663
5	0.628	10	0.686	15	0.640

Figure 4: Cross validation accuracy plot

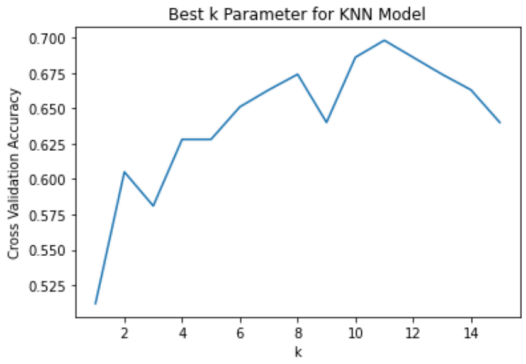

Figure 5: Cross validation accuracy table

Using k = 11, we were able to create a model with 72% test accuracy (albeit on a relatively small sample size).

Use the model to predict the string being played in unseen data Once we have our detection function and classification function, we are able to actually predict the strings played in our test images. To test this, we generated the predictions for an entire video and overlaid it on top. We were pleased to see that the model was able to determine the string being played throughout the video, even when it changed.

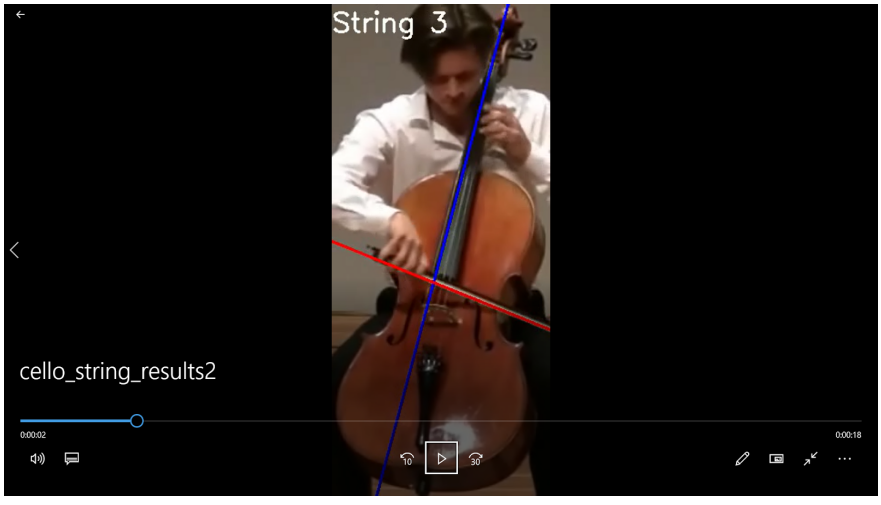

Figure 6: Final outputted video

6 Results

In the end, we were able to complete a pipeline that takes in a visual input and outputs a prediction. The pipeline has two main components: a feature extraction phase using an edge detector filter and a Hough transform, and a k-nearest neighbors classification system. Due to a limited dataset size, we are only able to roughly estimate the error of both phases, from which we can approximate the overall success of our model.

Out of the 132 photos that we added to our dataset, 108 were manually deemed as properly detected, giving us an accuracy of about 82%. Looking at the leave-one-out cross validation accuracy of the KNN model, we can see that using k=11 gives a validation accuracy of 70%, which when applied to the entire training set yields a testing accuracy of 73%. Below, we included a plot of the decision boundaries of the strings [4].

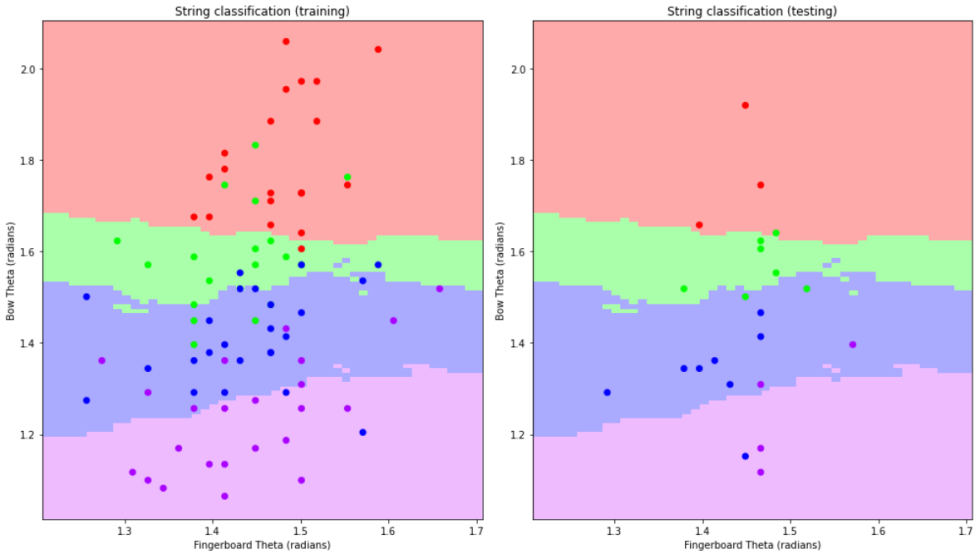

Figure 7: Decision boundary plot of knn model

Assuming that the errors are independent, we can estimate our overall accuracy as about 57%. However, when evaluating the accuracy of the model on continuous video, the common features of the values allows the model to perform better than this baseline expectation. For instance, the fact that the optimized model uses the previous frame's output should improve the accuracy of the first phase from the randomly sampled accuracy of 82%. In future work, expanding the dataset would allow proper testing and a more representative true accuracy of our model.

7 Strengths and Weaknesses of our System

7.1 Strength: Detects the bow and fingerboard accurately:

We were able to use a custom modified edge detector and the Hough Transform to predict the bow and fingerboard to a high level on a wide variety of photos.

7.2 Strength: Optimization:

Though not yet in real time, we were able to optimize our model to run significantly faster by limiting the area where the next line can potentially be detected. As discussed earlier in Section 5.2, this optimization leads to a great deduction in run time.

7.3 Limitation: Video Frame Size:

Our method only works on videos whose video frame is mostly composed of the cello instrument itself. As we discussed previously in this paper, this was a design decision that we had to make due to the nature of our model using Hough transform to detect the bow and fingerboard. This is a weakness because this means that with the current model, we can only run it on videos after the videos are cropped around the cello.

7.4 Limitation: Restriction on Cellist's Movement:

Relating to this limitation of cropping around the cello, our model may also not work well if the cellist frequently moves around within the video. This is because we add an optimization to our method that only looks at certain portions of the video in order to reduce the run time. Therefore, if the cellist moves around a lot, then the model may not be able to find the bow and/or fingerboard within the frame that they expect the items to be in, leading to incorrect predictions or no predictions at all.

This limitation can be solved by setting the optimization parameter within our detect_both() function to False. However, we would like to point out that this will lead to a 25x increase in runtime (from 0.2 seconds per frame to 5 seconds per frame). This is due to the model now going through the entire image, rather than certain subsections that it expects the objects to be at.

7.5 Limitation: Small dataset size:

Due to the limitation in the short amount of time we had to work on this final project, we were unable to create a large dataset. We did our best, but we recognize that this is a limitation on our prediction model and hope that in the future we can retrain our model on a larger dataset size.

8 Conclusion

In conclusion, we were able to create a k-nearest neighbors model that predicted the string being played by a cellist well. We were able to use our model on a 30 second video, and found that it performed satisfactorily according to the eye test. From testing the underlying classification model, we expect the accuracy to be about 70

Through this project, we learned more about the topics that we used in our project — such as object detection and Hough Transforms — because we had to think beyond the concept we studied in class in order to actually implement it in our project. In addition, there were many topics that we

considered but ended up not implementing, including perspective projection and camera geometry. Thinking about these topics was also sufficient in helping us better understand the course material in a new light.

In addition, we found how important it was to define the scope of the project. The project idea that we began with was very broad and ambitious, and we quickly found the difficulty of working with such a broad topic because there were so many paths and steps to consider, let alone the time limit we had with this project being due by mid-December. Thus, this project reinforced the importance of having a clear and feasible topic to work on so that it was easier to begin working with.

We also learned the difficulty in using various open-source APIs and libraries— specifically, OpenPose and MediaPipe. Though we did not end up using it, we encountered various problems with using the libraries while we were considering using them in our project. As simple as the tools' websites made it seem, we encountered countless bugs that prevented us from ever using them.

Lastly, we also learned how time consuming it was to create a dataset. Creating the dataset required finding a YouTube video that fit our criteria, as well as screenshotting and labeling of videos. Even so, the amount of time we spent only amounted to 138 photos. We definitely feel more grateful for the cleaned and labeled datasets available to us now!

9 Future Work

We feel that there are some exciting new paths to consider in the future, given this baseline model that we were able to get working.

9.1 Flexibility in Video Input

As discussed in the strengths/weaknesses section, our current model is limited to videos closely cropped around the cello. Working on a method that detects bows and fingerboards well regardless of cello placement would make this model much more flexible and applicable. One way to do so would be to train a CNN on a sufficiently large labeled dataset of cellos so that it is able to localize the cello on its own, at which time our current model can be applied.

9.2 Running in Real Time

Although we were able to greatly reduce the runtime of our model and compile at a rate of 5 fps, we would have liked it to be able to take in videos at 30 fps and be able to output the string prediction in real time. To do so, we would need to further optimize our Hough transform function or acquire more computational power.

9.3 View from Different Camera Angles

Our current model requires that the cellos be seen from the front of the cello; the video can not be taken from the side. This means that the string being played can not be well detected in videos that have been taken from other angles. Being able to detect the string played from a larger range of angles would greatly increase the applicability of our string detection method.

9.4 Detect Notes Played

This was our original stretch goal, but we didn't have time to fully explore the idea. Assuming that the cellos have been tuned to the correct frequency, it would be possible to detect the note

being played on the cello by looking at where a cellist's finger is placed on the fingerboard. Given that we were able to detect the string being played, a reasonable next step would be to detect the location of where the finger is placed and predicting the note played based on this location.

9.5 Train for Violin

We had originally started this project hoping to be able to detect the notes played on a violin, but as discussed in the motivation/goals sections, we moved our focus to cellos. Therefore, given more time to work on this project, and now with our better understanding after having created a model for cellos, it would be exciting to go back and try to train a model for violins.

10 Acknowledgements

We would like to thank Vivien Nguyen and Sunnie Kim for their guidance and advice throughout this final project, as well as our friend Andrew Chen for suggesting that we consider cellos instead of violins. In addition, thank you to Professor Olga Russakovsky for teaching us so much about computer vision this semester.

References

[1] Gómez, Emilia et al. "Visual Music Transcription of Clarinet Video Recordings Trained with Audio-Based Labelled Data". 2017 IEEE International Conference on Computer Vision Workshops (ICCVW). N.p., 2017. 463–470. Web. https://ieeexplore.ieee.org/document/8265272

[2] Zhao, Hang et al. "The Sound of Pixels". CoRR abs/1804.03160 (2018): n. pag. Web. https://arxiv.org/abs/1804.03160

[3] https://docs.opencv.org/3.4/d9/db0/tutorial_hough_lines.html

[4] https://stackoverflow.com/questions/45075638/graph-k-nn-decision-boundaries-in-matplotlib

10.1 Links Used to Create Our YouTube Cello Dataset

- https://www.youtube.com/watch?v=s92f3CW9IdQ
- https://www.youtube.com/watch?v=KJqNKq-qSWQ
- https://www.youtube.com/watch?v=s0zprgXLpeE
- https://www.youtube.com/watch?v=HkjIXDBz4e8
- https://www.youtube.com/watch?v=PqR36Drhn_k
- https://www.youtube.com/watch?v=wJlBkaIMOlkt=843s
- https://www.youtube.com/watch?v=Zc4pY6dQNFAt=40s
- https://www.youtube.com/watch?v=qPdEw6I86dg
- https://www.youtube.com/watch?v=xOjTsaHjBmUt=294s
- https://www.youtube.com/watch?v=JxKtiyNDgB4
- https://www.youtube.com/watch?v=SPe1MvW7CGs
- https://www.youtube.com/watch?v=wzdpJkrkQ4s

- https://www.youtube.com/watch?v=w-4aicGa9E0
- https://www.youtube.com/watch?v=u-jI1dFKWZ8
- https://www.youtube.com/watch?v=StV_wY2q4FQ
- https://www.youtube.com/watch?v=6I6S_KkYei4
- https://www.youtube.com/watch?v=BDPM3op_ZRs
- https://www.youtube.com/watch?v=roVybvwArbE
- https://www.youtube.com/watch?v=m7Gp18SD0js
- https://www.youtube.com/watch?v=f_xJI-EA0ys