AR Music Pad-- Team C1

Author: Mingquan Chen, Xinyu Zhao, Tianhan Hu
Department: Electrical and Computer Engineering, Carnegie Mellon University

Abstract: Nowadays people almost have had the ability to realize everything we want using codes. What could be improved is how are we using it. Our team wants to design an AR music pad so that people can enjoy their interactive experiences whenever they want. It would be portable to any device and very convenient to use, and would include a web application. All you need to do is having a camera, computer and a table.

***Indexed terms*:** Augmented reality, Edge detection, Hand detection, Music

I.Introduction

This would be a computer program implemented in python mostly. When it starts, it will try to find a table through the camera and display buttons on computer screen. When the camera detects a person’s hand on the buttons, the program will play sounds and music correspondingly. The major improvement here is to have the program to be portable so that people do not need to buy and carry that heavy music pad everywhere. Instead, they could use this computer program and interact with AR effects. People can also save a lot of money. We would want this program to be very sensitive to people's hands movements. Program should respond to virtual "press button" within 0.5s to give users good experiences.

II. Design Requirements

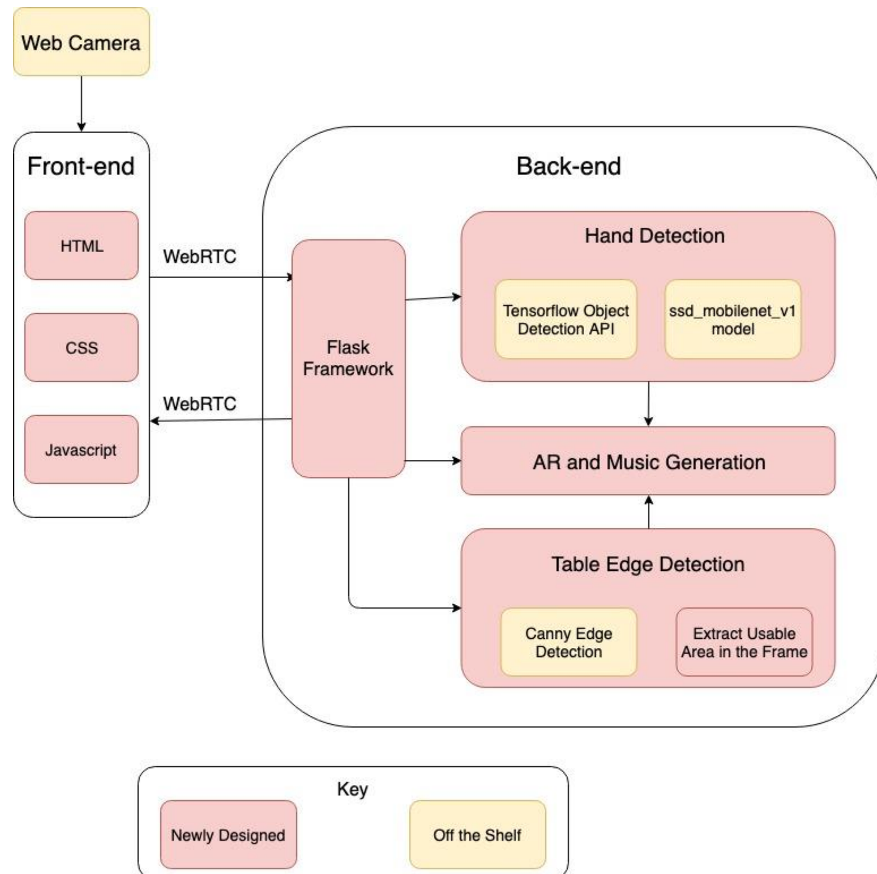
We have broken down our program into three major parts. A table edge detection program is written to detect plain table so we can display AR on it; AR generation program is used to display buttons and interactive changes with users' hands movements; hand detection program is applied to detect users' hands movements. Also there is music generation program which plays sound tracks according to button got pressed.

For table edge detection, we would want the program to be able to detect a plain table. It should, first of all, be correct in detecting edges. The edges' positions as results should be within 10% range of real edge positions. These edge positions in terms of coordinates are used to generate AR button effects. If the table is too large that we could not find the edges, we will not detect it. If the table is too small in the camera such that we do not have enough space for placing buttons, we will not detect it. If the table's positioning is skewed too much such that it would unreasonable to use as a music pad, we will not detect it.

For AR generation, we will display buttons as 2D squares. When people press on it virtually, they should be playing music soundtrack. There should be a default template, but the user is able to customize their interface as well.

For hand movement detection, we would like the program to have relative fast latency, as a music pad should have fast response when users have "instructions". Currently we set the goal to be less than 0.5s response time. We will try our best in improving this number in order to have better user experiences. It will be realized through trying different algorithms for detecting hand movement. Time required to calculate current hand position will be a metric that we measure and compare. Another major goal is being able to correctly detect "press button" action. Since camera only has a 2D view from top down we could not detect the "press" action, but we want to differentiate that from simply flying over buttons. We might realize this function through hand gesture detection i.e. making fist as press action

III. Architecture and/or Principle of Operation



MusicPad is deployed in two versions: the local version and the webapp version. Above diagram is for the webapp version of the deployment. The frontend side would reside at user's browser when user opens up our webapp page. The frontend comprises of HTML, CSS and Javascript code. Its primary responsibility is for collecting video input from user's web camera, sending video frame by frame to backend server for further operations, and collecting frames back and displaying on the browser. It is also responsible for displaying the webpages and collecting other user actions such as button clicks.

The backend server is where all the computations take place. It is built and ran with Flask and Aiohttp and deployed on a linux server. The computation comes in three parts: hand detection, table edge detection, and AR and music generation. The hand detection part and the table edge detection part collect the video frame inputs from aiohttp server, and then perform their corresponding computations. After that, these two parts would send the computation results to the AR and Music Generation part, which would perform the hand-button interaction test and draw the AR effect on the frame. Finally, the modified frame would be sent back to the frontend for display, alongside the choice of music to be played if any key is pressed.

All the communications within the back-end server is performed with Python calls, and the communication between the front-end server and the back-end server is carried out with WebRTC.

The local version of MusicPad is very similar to the back-end server of the webapp version with the difference that the web camera input is directly fed to the three working parts, and the modified frame and the music choice is displayed right away.

IV. System Description and Design Trade Studies

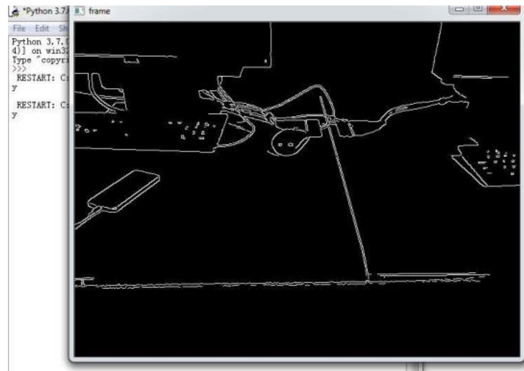
1. Table edge detection

The table edge detection step is the first and a fundamental step of the process. It will be responsible for the providing input data for the next stages, AR projections and hand detection.

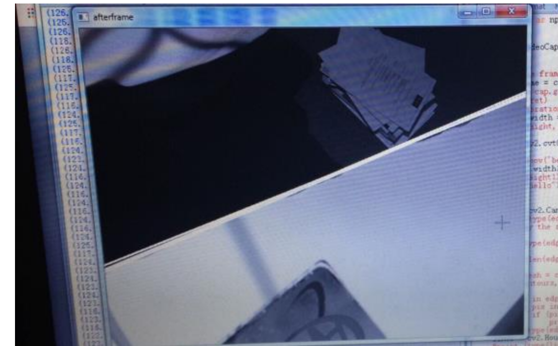
Functionally, at first we thought the camera would scan the surface, if there is no edge to be found, use the entire scope as the data transferred to the AR module, if edges and vertices are found, the coordinates of the edges and vertices within the frame will be recorded. However we then found that it's more practical to assume the camera would find a suitable surface and let the process be instantaneous.

A traversal will be done with the data to distinguish and filter out the desired table's vertices and transfer them to the AR and hand detection part which will know which part of the image is used for effect projection. If the camera is too high from the surface such that the detected surface is too tiny, no data would be transferred.

A lot of different algorithms were attempted in the process. First is the Canny edge detector with the hough line transform, as the numpy nd arrays of grayscale "edges" created by Canny edge detection is required for



the houghlines operations. The lines obtained are mostly definite and clear,



Canny Edge detection(first pic) and Hough line transform

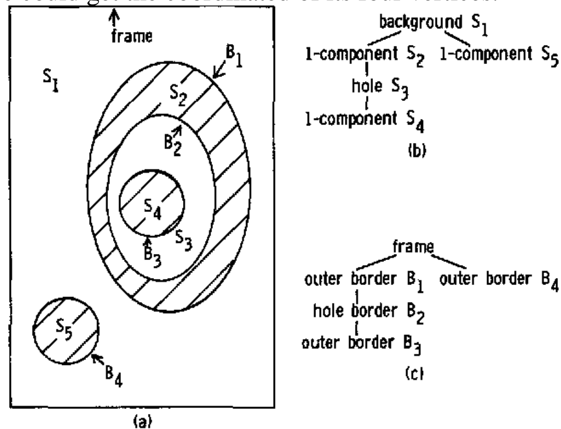
however there was no good way of pinpointing the coordinates from the lines.

Harris Corner detector was also experimented. It's a combined corner and edge detector. In static images where conditions were ideal the algorithm worked well, however when dealing with frames captured with camera in real time many more unnecessary corners were detected most of the time, making it less reliable for our case.

Harris Corner Detector results

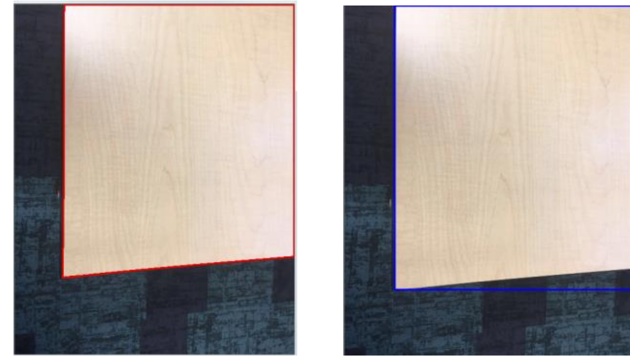
The next attempt was with the contour features in opencv. The most critical part was detecting the contours in a frame. The process was related with the algorithms discussed in an academic paper about topological structural analysis of digitized binary images. As a result we got an array of contours in the frame. Then we have to find the largest one in size within it. After obtaining it there were two paths that were explored: one was using the polygon approximation, which would find the polygon that's the closest in

size and shape encompassing the contour and give out the coordination of the vertices; the other one was finding the minimum external rectangle, after which we could get the coordinated of its four vertices.



Determining the surroundness relation among borders

The polygon approximation was more precise in terms of capturing the edge of the surface exactly, however in many cases when obtaining the coordinates a minority of vertices would be skipped over mainly due to the irregularity of the shape and the occasional instability of the algorithm, which would greatly undermine its ability to transfer data to the next stages; The minimum rectangle approach, would find a rectangle encompassing the target regardless of the target's shape, so it's slightly less precise in this regard, however because it's always a rectangle we can reliably receive the information of all four of its coordinates, making it very reliable to transfer data to the next stages, which is why the minimum rectangle approach was used at the end.



Polygon approximation(red) vs minimum rectangle approach(blue)

2. Hand-Detection Subsystem

This component was originally designed for detecting user's hand movements, and deciding whether the user is trying to activate a button. As we moved forward with the project, we decided to move the decision part to the AR and music generation component. Thus, in the final implementation, this subsystem works solely for the purpose of capturing hand movements, and reporting to the AR generation subsystem.

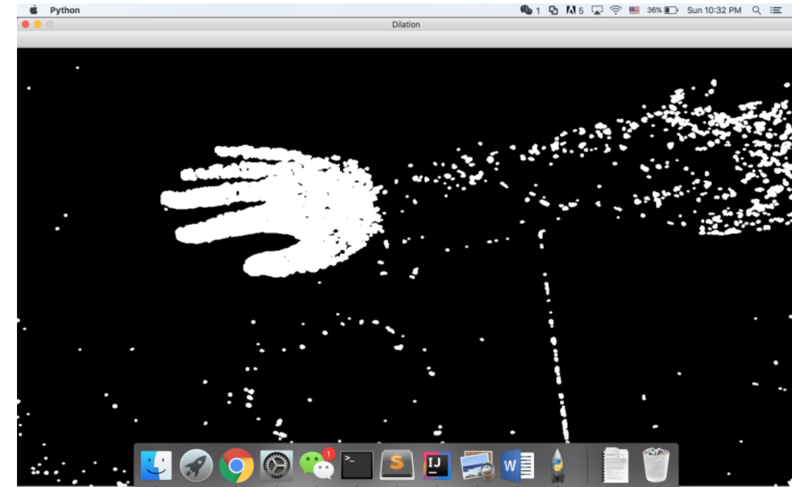
A. First approach: Static thresholding

The first approach we tried for Hand-Button-Interaction Detection was by using static thresholding. In this approach, we tried to differentiate the between user barely moving his/her hand over a button and actually trying to activate the button.

The most intuitive way for user to activate a button on the table would be for him/her to press against the button icon. However, since we plan to display multiple buttons on the table, there would be cases that users needs to move their hands over some other button before pressing on the button that they wish to activate. This is troublesome since all hand detection algorithms that we looked into were not capable of telling the "depth" of the hand or whether the hand contacted with a surface or not. To address this problem, we decided to fallback to a less intuitive but more feasible way of detecting hand-button interaction.

The updated plan is that we would be requiring users to clench their fists while moving towards the target button. Once their hands are on the button, they should release all their fingers to activate the button. With this design, we could then implement hand-gesture detection for telling if the user really wish to activate a button or is just merely moving his/her hand towards the target button.

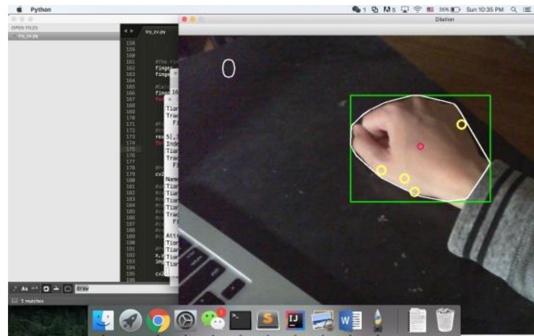
We separated hand-gesture detection implementation into two major steps: hand detection and gesture detection. For hand detection part, we were using skin color detection with static threshold. Basically, through testing we fixed a lower bound and a higher bound for skin color we are looking for, perform dilation and erosion based on that, and threshold binary to extract the hand pixels.

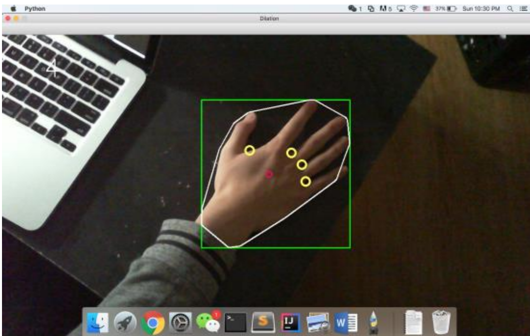


Hand Detection with static thresholding

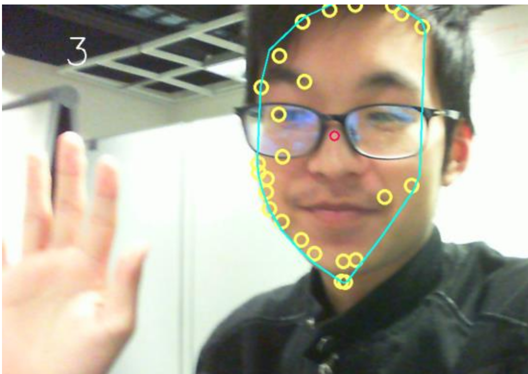
Gesture detection is the next step. We would first find the contour of the hand if it is present in the frame. Based on this information, the program will then calculate the convex hull, center of mass and the convex defect of the hand. The closest four convex defects from the center of mass would be the finger webs, and we know that a finger is pointed if the distance between some pixel on the convex hull and the center mass exceeds the distance between finger web and the center mass by some distance.

The biggest concern, as mentioned by TAs and Professor Marios, is that static thresholding suffers as the lighting condition and table color varies. Moreover, this algorithm is unable to differentiate between hands and arms or even human faces. Thus, having any of these elements in the camera would largely affect the correctness of the result. After several rounds of testing, we decided to give up on this implementation and move on to more adaptive methods.





Hand Gesture Detection with static thresholding



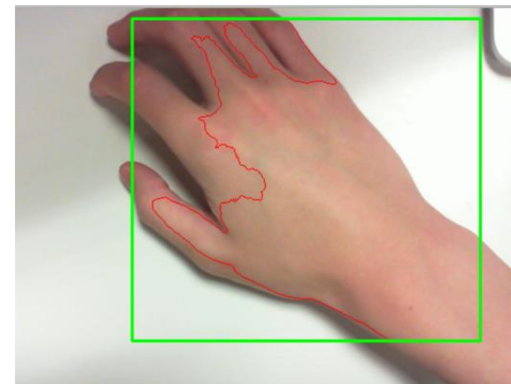
Static thresholding misidentify face as hand

B. Second Approach: Adaptive Thresholding

The second method that we tried was to use a adaptive threshold. Here “adaptive” means that adapting to the background frame. The idea is pretty simple: the background in our use case would be a table and a table’s color would normally be fixed. Therefore, we could get the RGB values of a frame containing only a table before user decides to start the MusicPad. After the user starts play and lays his/her hand in, we would be able to compare the values of the frames with and without hands and be able to recognize the hands in the picture.

In our implementation, we capture the first 50 frames and use their average value as the value for frame without a hand. Then, we allow the user to put his/her hand in. We extract the hand pixels by subtracting the pre-captured frame from the new frame and performing a binary thresholding. The rest of the implementation is exactly like our first approach with static thresholding, with contour finding, convex and convex defects generating, and deciding the hand gesture based on these information.

The result was rather disappointing. When light condition varies on the table (some parts lighter and some parts darker), and the user’s hand cross both areas, our algorithm tend to only recognize part of the hand. Moreover, this method does not solve the problem of misidentifying arms and other human body parts as hands, and would even recognize any other intruding object as a hand. This method also posts more restriction on user’s operations since the user would now have to maintain the camera frames still when reading in the first 50 frames.



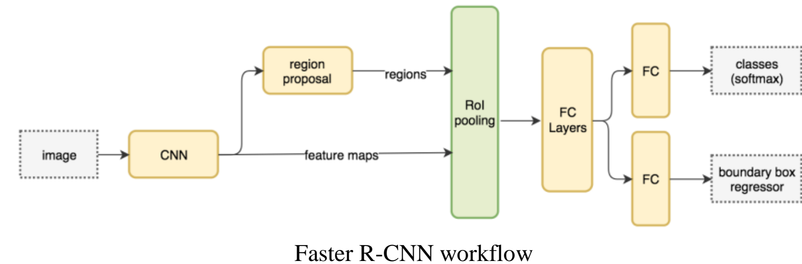
Adaptive thresholding only recognizing part of the hand (the part circled in red line)

C. Final Approach: Machine Learning Based Hand Detection

When we first discussed about how to implement the hand-detection subsystem, we debated between using a machine learning based approach or using only OpenCV. We first settled with using only OpenCV for hand-detection because we were concerned that detecting hands frame by frame by ML would be too slow, and our application would need more a timely detection method. However, when our first two attempts with pur OpenCV methods failed, we decided to give machine learning a try.

We decide to use Tensorflow's Object Detection API for our hand-detection implementation mainly for the reason that there are a great number of pretrained models that could work with this API, and this would make the training process much easier. Also, this API, when detecting objects in an image or a video, could generate a bounding box around the detected object. We could then use this bound box to determine whether or not user is putting his/her hands over a button. The dataset that we are using is the Egohand Dataset[1]. The dataset contains 4800 images of hands in egocentric view and with hand annotations, and we believe would closely resemble our requirement. We spared 500 images for validation, and used the rest 4300 hand images as training set.

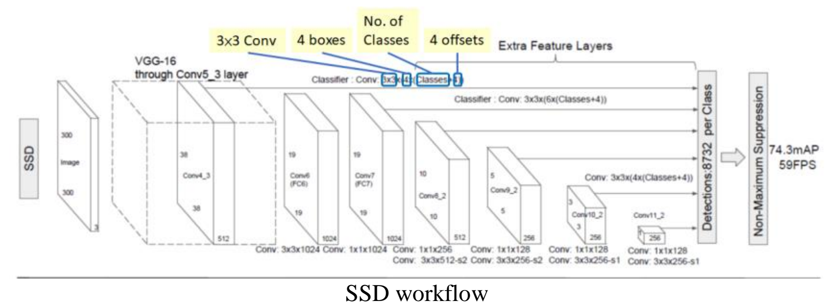
The first model that we tried was Faster R-CNN since we learnt that it would be the most accurate object detection model. It is also faster than conventional Fast R-CNN model by replacing the slow Selective Search with a small convolutional network called Region Proposal network.



Faster R-CNN workflow

We trained using transfer learning on Google Cloud ML Engine with 50000 steps. The obtained hand-detection model was able to detect hands in images with high precision. The problem was that when processing a video, it was too slow for our purpose. The FPS rate fails below 1 (around 0.7), and it was unacceptable for our application which is supposed to be responsive and timely.

To obtain a better detection time, we turned to Single Shot Detection model (SSD). This model runs a convolutional network over each image only once, and therefore would be much faster than Faster R-CNN. On the downside, its accuracy would be compromised compared to R-CNN models. We trained again using the same 4300 images as training set with transfer learning. The result was pleasing. The model was able to reach a 0.89mAP[2], and the frame per second is around 7 when applied to a video.

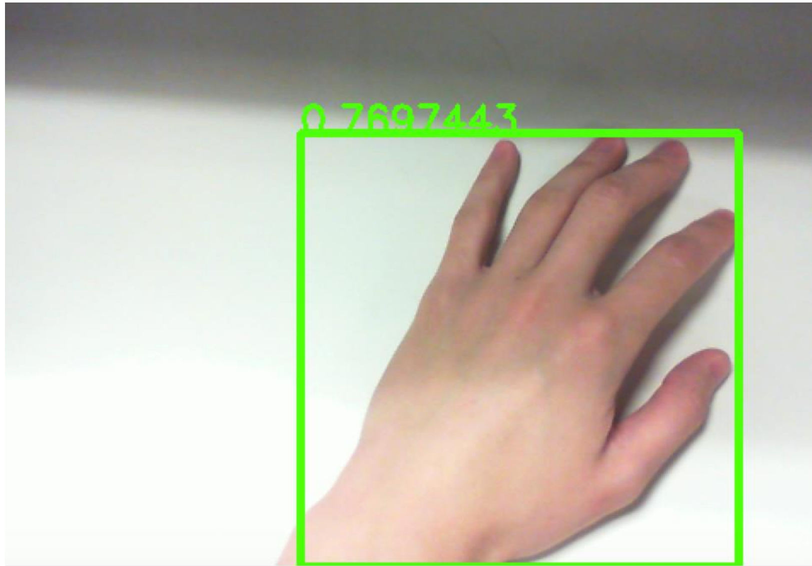


SSD workflow

[1] Egohands Dataset: http://vision.soic.indiana.edu/projects/egohands/

[2] mean Average Precision

This result is pretty satisfactory since now we have a moderate FPS rate and still a very high accuracy. Compared to the previous two OpenCV approaches, the Object Detection based method is able to work under more extreme lighting conditions, and detect hands on tables with very similar colors. Moreover, it makes it possible to differentiate between hands and other parts of human bodies so that other objects such as human arms would no longer trigger any button.



The problem with this approach is that we are not able to further detection hand gestures after finding the user's hands. Therefore, it makes it impossible to tell whether users are just moving their hands over a button, or actually wanting to activate it. After massive research, we found this problem very hard to address. Finally, we have to make the decision to put the burden on the users' side, by instructing them to leave their hands out of the camera frame and only putting their hands in when moved above the desired keys.

3. User Interaction Component

This component comprises everything that program would interact with users' actions. It has two major parts: graphics and music.

In graphics, our program will draw a table and buttons on the screen based on the table we detect. After our program detects the table, it will forward corner positions and our program will draw buttons within the table. The default button number is four. By default relative positions for buttons and table are fixed. When users change their angle of viewing the table by camera, the locations of buttons will change as well. There is another mode inside program called "design mode." When users press button "d" they will enter into the design mode. In design mode users have the ability to draw the buttons wherever they want. They just need to simply drag mouse. Button positions will be remembered relative to table position. All these are done through python opencv package. In the demo template, we use pictures from Zelda to represent buttons. They are overlapped onto the original frame by tuning pixels.

After drawing buttons, their positions will be remembered. When hand position data came in, the program will compare hand position with button positions. To simplify the model the hand position is represented as a single point by taking the center position of the box detected. If that point is inside any button, the program will react with it by playing a single sound or rhythm. In our demo template we choose to play a single sound.

Audio files are played through simpleaudio package in python. To embrace better user experience, all audio files are adjusted through audacity so that they have same length and similar amplitude. To avoid repetitively playing a single sound, there are logics to decide if the hand is placed on a button already or the hand is moving to a button. The soundtrack is played on a separate thread so that it would not interfere with the main page.

To add more interactions with users, when users press buttons in a certain sequence, they would listen a whole soundtrack instead of single sounds. Also, a special animation will be displayed. There is a global array remembering last few buttons and a function to check if there are any sequence matches the designated one. If so corresponding soundtrack will be played. The soundtrack is produced using Audacity by combining and tuning single notes.

The animation contains two parts: text and image. It will display "bingo!" when you successfully press such sequence character by character. A cartoon character will also be displayed as he is playing an ocarina. Those are set to be finished in a few frames so that it would be animated effects instead of a single picture. In each frame the program will modify the pixels on the main page and as the frames go on an animated effect will be displayed.

To improve user experience further, we add "fix" and "release" buttons onto our screen. They will be triggered by mouse clicks. When "fix" is clicked, our table will be fixed so that no matter how our camera moves, the table position would not change. This tends to help user play with the buttons after they find a sweet spot for their camera. If "release" is clicked, our program will use real-time table edge detection again.

Buttons, soundtracks, and combo sequence are all customizable by users. But for demo purpose we have planned out a template for you already. Users can select their own desired ones and play with it.

4. Webapp Deployment

As the last step of our project, we wish to deploy MusicPad as a webapp, so that people could use our app anywhere, anytime in the world.

We choose Flask as our backend server framework since all our code is written in Python, and compared to django, Flask is more lightweight and easier to deploy. We used HTML and Javascript to build our frontend layer.

The biggest challenge in deploying MusicPad as a webapp is to gather the video frames at users' browsers using their webcam, stream the frames one by one to the backend, and streaming them back for display. For gathering video frames from users' browser, we use WebRTC[3], which is a open sourced project developed by Google, and embedded the project API in Javascript code. The result is pleasing, and we are able to get a smooth video stream from the webcam, and displayed on the browser. For transmitting the frames back and forth between the client-side and the server-side, we first tried using Flask-socketio[4] library. As the library works well with transmitting images, the latency is too high to stream video frames, and thus not applicable for our usage.

As our second try, we used AioRTC[5], which is a Python library built around the WebRTC project. It provides the same functionality as WebRTC of exchanging images and videos between sites, and could be binded to Python code easily. With the help of AioRTC, the latency of sending, manipulating and receiving a frame is largely cut down, and we were able to raise the FPS rate to 2-3. It is still very low compared to our local version of MusicPad, but we believe that it is presentable, and could allow the users to get a bit of taste of MusicPad.

Finally, we deployed our project using a single linux server[6] acquired from Linode. Since it was approaching the deadline for the project, we were not able to acquire SSL for our website, and our service is only deployed using HTTP, but not HTTPS. For this reason, only Firefox, which allows users to open their cameras for HTTP sites could fully support our webapp at the current stage.

[3] WebRTC: https://webrtc.org/
[4] Flask-SocketIO: https://flask-socketio.readthedocs.io/en/latest/
[5] AioRTC: https://github.com/aiortc/aiortc
[6] Our website: http://musicpad-500.com . Enjoy!

Webappp Interface

V. Validation and Metrics

We've done validations mostly for the table edge detection part and the hand detection part. For the edge detection, we calculated the average gap between the actual edge from the computed one, and the area ratio between the two versions, the average accuracies for them are over 90 percent. The response time of the program as calculated by the number of frames per second is 0.03 seconds(around 30 frames per second).

For hand detection, and for the object detection based method with the SSD model, we put in pictures of hands from the testing dataset to test its accuracy, we were able to get a 0.89 mean average precision, and the frame per second is around 7, making the overall response time about 0.14 seconds.

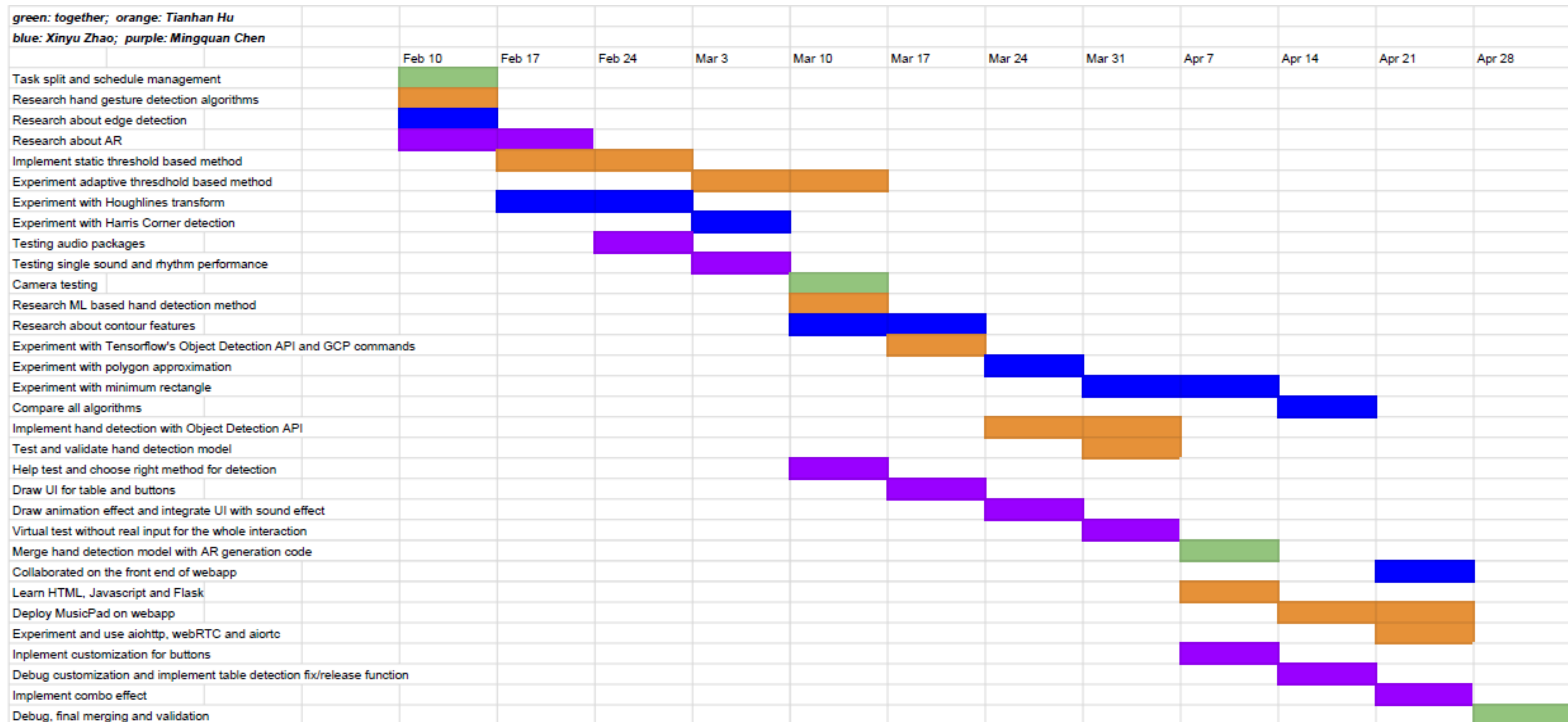
There are also some edge cases that we paid attention to: The program is able detect more than one hand at the same time; because of the nature of contour finding algorithm, when human hand is in the camera it would also get interpreted, which conflicted with our original intention, and that's why we decided to add the frame fixing feature in the interaction part; another one is that the program had trouble dealing with environment that's too bright or too dark.

VI. Project Management

A. Schedule

Due to some changes of plans after the proposal, we made some modification on the schedule, which is attached at the end. More time is to be spent on bettering the color extraction process of hand detection part and the data transferring process of the (table) edge detection part. Since the cameras

just arrived, there would be more testing and data providing afterwards, and we were also reminded that we haven't done things for music generation which is our major output, we will be focusing on that sooner. We also considered the time for code combination and integration. The chart is attached in later pages.



B. Team Member Responsibilities

Tianhan Hu:

My primary responsibility is to build our hand-detection subsystem. During the first 8 weeks, I researched and implemented three different ways of detecting hands from videos, and fixed the Object Detection based method as our final implementation. I then help the team to merge the hand-detection subsystem with the AR and music generation part.

I am also responsible for designing and deploying the webapp version of MusicPad. I collaborated with Xinyu in building the frontend layer. I implemented the backend server using Flask, and the communication layer between the client-side and server-side using AioRTC. Finally, I deployed our webapp on a single linux server on the following site: http://www.musicpad-500.com

Xinyu Zhao:
My main responsibility is implementing the table edge detection part to provide coordinates for the AR effects. I did researches and experimentations on various detection algorithm extensively, with respect to both static images and moving frames, and finally settled at the method with contours. I then helped with the team's resource finding, collaborated with Tianhan on the front-end part of the webapp, and also completed the poster for the final demo.

Mingquan Chen:
My major job is to implement all the interactions with users, including animation and music. I cooperate with my teammates by integrating their data into our program framework. I made all the image and animation effects as well as tuned the audio soundtracks. When our project headed into webapp deployment phase I also helped debugging and testing.

Also, as a team member I helped decide on which strategy we should use and all the factors we should consider. As we said we have tried a lot of different methods, and we discussed together to figure out which one fits our project the best.

C. Budgets

Items purchased: Genius 120-Degree Ultra Wide Angle Full HD Conference Webcam($52.99), Logitech C930e 1080P HD Video Webcam - 90-Degree Extended View, Microsoft Lync 2013 and Skype Certified($78.15), Executive Office Solutions Portable Adjustable Aluminum Laptop Desk/Stand/Table Vented w/CPU Fans Mouse Pad Side Mount-Notebook-MacBook-Light Weight Ergonomic TV Bed Lap Tray Stand Up/Sitting-Black ($39.9)

We thought we would need to purchase an additional camera based on the performance of the given two, but it's actually not needed.

D. Risk management

We were worrying about testing out so many algorithms might make us miss our schedule. It turned out that we successfully tested out all we want. We worked as a team to support each other by keeping active and having weekly meetings. We kept giving feedbacks to each other and helped to solve the difficulty.

During the last two weeks when we need to deploy our program as a webapp we faced some trouble, as none of us had experiences with webapp. We distributed work properly and helped debugging. Some of us stayed up late for a few days and we successfully managed it.

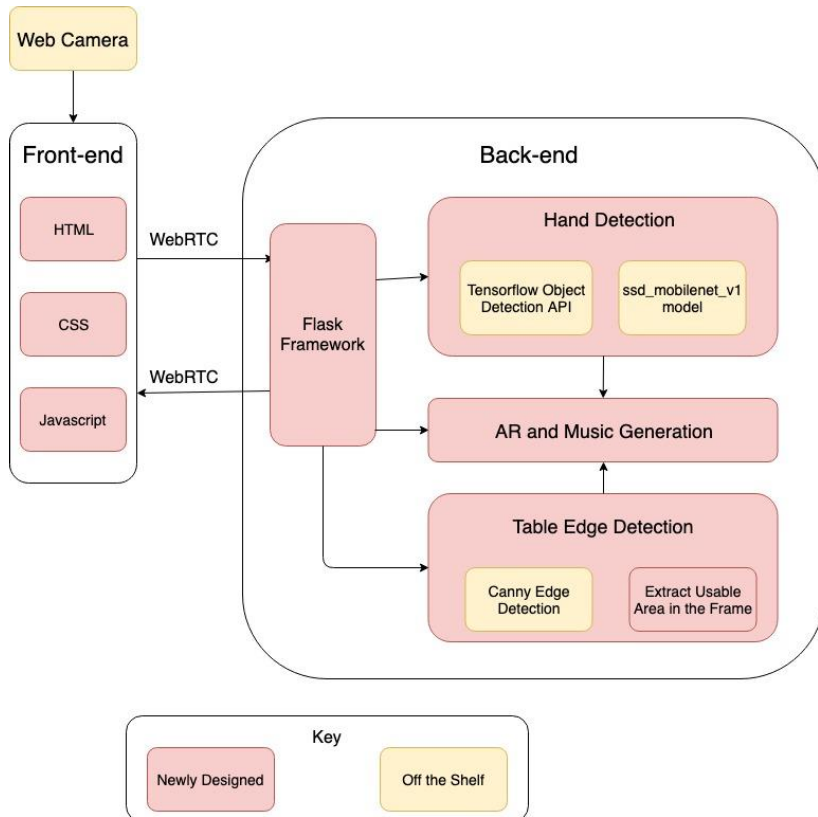
VII. Conclusion and future work

We met our expectation for this project. The major goal was to utilize hand detection to improve life quality and we chose to focus on music applications. Our algorithms at the end worked out well with average of more than 90% of accuracy. The best thing is the program will not recognize anything else, as we have tested, to be human hands. This is very acceptable, as even if it fails to detect hands in the rare cases user just need to hang on for less than one second and the program can recognize the hand. Also the latency by using Single Shot Detector is very low and we can have around 10 fps. This meets our original goal of having less than 0.1s latency.
Our functionality is basically accomplished, as we have interactions both in graphics and music. Nevertheless, there are definitely more things we can do:

If we have more time, we would like to implement following functions:

1. 3D AR buttons. Instead of simply putting 2D squares we will use 3D cubes as buttons. We will also have the pressed version of buttons to make users feel more realistic in using the music pad.
2. Music soundtrack trace. We will have a record function and once users start to record, the button sequences will be remembered and could be played later in the future.

3. More functionality on webapp. Since we do not have experiences with streaming on webapp, we cannot resolve the latency issues. But we would definitely want to see it work with full version.
4. More customizable items on webapp. As we wrote in interaction section, users can customize basically everything: buttons, animation, music, combo, etc. If we deploy full version onto webapp, it would definitely be great to have all these uploaded by users themselves.
5. Using https instead of http. Currently we are using http, so it is limited to certain browser usage. If we can expand to https, people using mobile phones can access our website as well and it would be truly portable.


Web Camera
Front-end
HTML
CSS
Javascript
WebRTC
WebRTC
Back-end
Flask Framework
Hand Detection
Tensorflow Object Detection API
ssd_mobilenet_v1 model
AR and Music Generation
Table Edge Detection
Canny Edge Detection
Extract Usable Area in the Frame
Key
Newly Designed
Off the Shelf