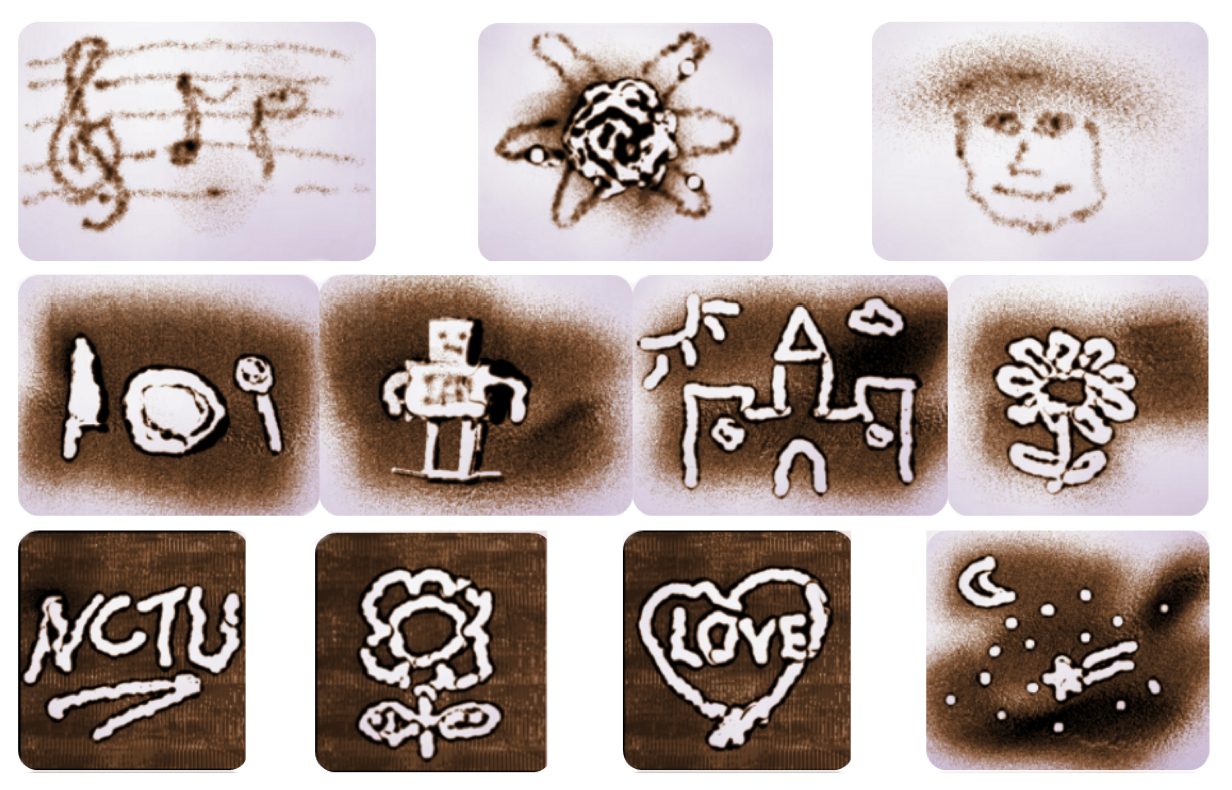
**Project Proposal: Gesture Doodler**

There are many drawing-based iPhone apps available on the App Store, such as KidsDoodle, Sketch, and LINE Brush. The vast majority of these apps involve translating touchscreen movements into brush strokes; apps that enable users to draw using visual gestures such as “ok” signs are much rarer and more difficult to build, especially for the iPhone. Hands-off gesture systems involving the iPhone have been relatively recent. A group of researchers recently presented a new class of interactions, Air+Touch, where the user can interweave touch and in-air (not touching the touchscreen) gestures to create fluid and expressive interactions (Chen, Schwarz, Harrison, Mankoff, & Hudson, 2014). Our team aims to develop an iPhone app that allows users to make basic doodles using a small set of gestures.

**Background:**

In late 2014, a group of Taiwanese researchers successfully developed a sand painting user interface using Microsoft Kinect. Users could perform sand-drawing operations by presenting one of four gestures (closed palm, open palm, single-finger, two-finger) to either the command or sand drawing area. The five users who tested the system were able to produce basic drawings within a few minutes; however, most of the testers agreed that the primary challenge presented by the system was their inability to directly touch the sand that they were drawing on (Wong, Chen, & Chen, 2014). This kind of system limitation is expected, as it is much harder to make detailed and precise drawings using gestures, as opposed to pen and paper. However, the system proposed by Wong et al. worked great as a doodling app.

**Figure 1.** Doodles produced by users of the sand drawing interface made by Wong et al. (2014).



Wong et al.’s system was created in three major steps. First, their system detected hands and then computed the hand’s position and gesture. Based on the gesture, the system determined the drawing position and action and sent the information to the sand drawing system. Finally, the drawing system performed the corresponding sand manipulation method.

The task of detecting hands and interpreting hand gestures can in turn be broken down into three major steps, as done by Lee and Lee (2011). First, human skin is detected using background subtracting and thresholding. A scale-invariant angle detection method, based on information about the curvature of the hand, is used to find fingertips. Finally, gestures are recognized by analyzing the position and contours of the detected fingertip (Lee & Lee, 2011).

**Investigation Limits:**

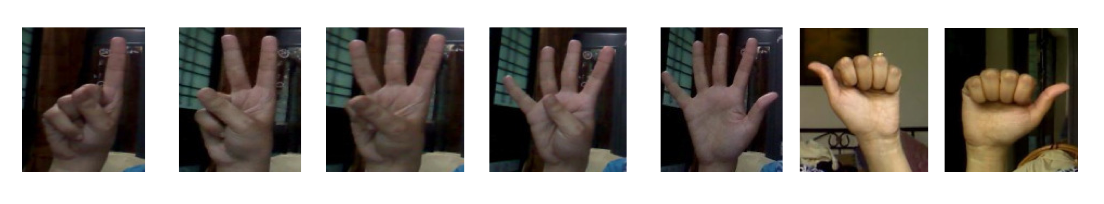
A drawing app as complex as the one made by Wang et al. cannot be implemented in the limited amount of time we have. While Wang et al. was able to detect hands even when the user’s face was visible to the sensor, we will restrict our domain so that the user will be presenting only their drawing hand to the camera. Depending on the sensitivity of the hand detection software to environmental objects, we may impose the additional limit of using a black background. From our experiences with the first assignment, this restriction will most likely be necessary.

Our iPhone app can be used in two ways: with the camera facing the hand of the artist, so that the person who is drawing can see the effects of their gestures, or facing away from the artist, so that the audience but not the artist can see what is being drawn.

**Anticipated Methods and Results:**

For hand detection, we will be working off our assignment 1 projects, which already contain the code for isolating the hand using background subtraction and thresholding (converting from OpenCV with Python to C++ so that it can be used with iOS). From there, we will refine the code so that it is capable of distinguishing a wider array of gestures, beyond just fists and splays.

In the first assignment, we simply needed to distinguish a fist from a splay, and any gesture that wasn’t a either a fist or splay could be classified as ambiguous. Nina’s system was able to identify and distinguish these “intermediate” gestures by counting the number of convex points in a hand to see how many fingers were being held up with a relative degree of accuracy. Since there’s a lot of room for ambiguity, we need the gestures to be as different from each other as possible. As an example, each of the gestures in the set used by Ghotkar and Kharate (2012) featured either a different number of protruding fingers or a different hand orientation. The gestures we anticipate using are fist, splay (five fingers), pointing up (one finger), peace sign (two fingers), and okay sign (or three fingers).



**Figure 2.** Gesture set used by Ghotkar & Kharate (2012).

*Step 1: Domain Engineering*

To get the system to recognize the gestures successfully, the user must have their fingers (if more than one is protruding) widely spaced apart. In this way, our system will be able to discern gestures based on the number of contours between the fingers. The user’s hand needs to be far enough from the camera and from the corners of the image, so that the camera can see their entire hand. If the user breaks this restriction, the system may under-count the number of fingers that are protruding from the fist and misinterpret the user’s gesture. We already have a working system that does this on the webcam, and need to port it to the iOS platform.

*Step 2: Data Reduction*

We will need an method that analyzes the hand images pixel by pixel and segment the different parts of the hand into clusters based on the locations of the “white” and “black” pixels in the image. First, we define the largest un-interrupted area of white as the “fist.” By this definition, the masses of white pixels that “stick out” from the fist area are considered the fingers. To determine whether the fingers are sticking out, we scan the pixels around the “fist” area for alternating white and black regions. For example, in order to detect the thumb, we need to scan for white (fist)-black (gap)-white (thumb) pixel regions on either side of the fist.

Since we are looking for areas of alternating black and white pixels, the orientation of the hand can be vary – tilting the hand should not affect the system’s ability to recognize gestures. Separating the fist from the forearm, however, is another issue that we need to tackle. In a drawing app, at least one of the gestures must be dynamic, so that the user isn’t always drawing in the same part of the screen. Since the fist is an extremity, we need a method to find the center of mass for the fist and essentially chop it off from the wrist so that we are only analyzing the hand, and processing the arm.

*Step 3: Parsing the Gesture Grammar and Tracking Hand Movement*

Once the gestures have been recognized, we will translate them into system actions using the following mapping. We decided that using the center of mass of the fist as the “paintbrush” for tracking and translating the movement of the fist into brush strokes is easier than attempting to translate the position of the tip of the indicating finger into a brush stroke. This requires real-time object tracking, which Melanie’s implementation of assignment 1 covers, so we will build off of that.

**Fist:** draw on the screen – the movement of the fist is translated into a stroke

**Pointing:** change the color of the brush (changes to a random color)

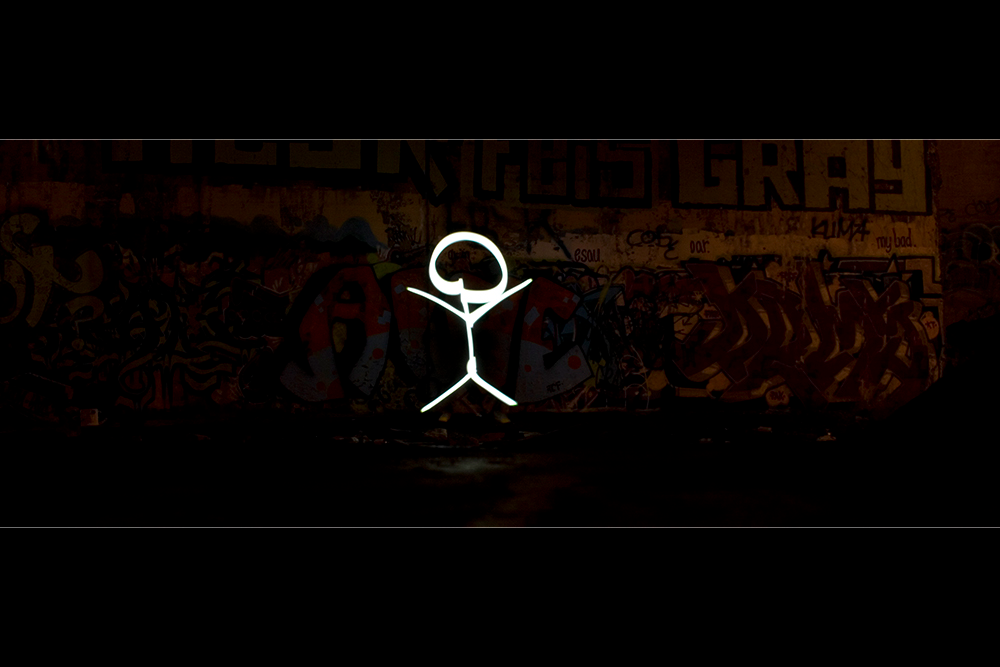
**Peace sign:** pause drawing (use fist to resume)

**Ok sign:** save the image (can just do three fingers)

**Four:** (optional: either undo; or change brush style, as from pencil to crayon)

**Splay:** erase the image

If things go well, we will aim for not only for solid colors, but white lines with traces of other color outlining the white as to follow the aesthetically pleasing appearance of light art/drawing. Examples as follows:

**Figure 3.** Search results as result of looking up “light art” on DeviantArt.com

**Evaluation Metric:**

We will measure program performance by whether it is possible for users to make simple doodles using our system, such as a flower or a house. Since this is a doodling app, not a professional art studio app, we don’t expect to be able to make drawings that contain a great amount of detail. Another measure of performance is whether it works in real-time: in a typical drawing app, the user’s movements on the touchscreen are immediately translated into the appropriate strokes. As a drawing app, we hope to be able to meet this standard as well.

Since coarser gestures such as fist and pointer movements offer far less precision compared to drawing with a pencil on paper, or drawing using Photoshop and a tablet, we anticipate that users will initially have a difficult time producing their desired drawings. For example, in order to draw a smiley face, the user will probably have to draw the entire face in one stroke - the equivalent of completing a drawing without lifting the pencil off the paper. Thus, the quality of drawings produced by our system will be noticeably lower than the quality of drawings made using a typical iPhone drawing app.

Finally, we expect that the gestures will translate to the system action that they are mapped to most of the time. For instance, fists should translate to strokes, rather than resulting in the image getting erased (the expected result of presenting a splay to the screen). To lessen the chances of an unintended image erase, we deliberately left a “gap” in our gestures map – there is no four-fingered gesture in our vocabulary. In the very worst case, only the ok sign should ever be misidentified as a splay. We will consider 95% accuracy as good system performance – 95/100 times, any given gesture should result in the correct system action.

We can also loop in a second user to evaluate the results of a first user to get further evaluation for our metric. If both users see what the first user intended to draw, that would be a true positive result.

|  |  |  |
| --- | --- | --- |
| What User A Wanted To Draw | What User B Sees | What User A Thinks Image Looks Like |
| A | B | C |

true positive: (B = A) ^ (C = A) false negative: (B = !A) ^ (C = A)

false positive: (B = A) ^ (C = !A) true negative: (B = !A) ^ (C = !A)

Since A, B, and C will be English language terms, we may need to use WordNet to find word similarity and/or our own judgment calls, in order to make a decision about how close or whether or not B = A and so on. With such information, we can calculate the precision, recall, and F-1 harmonic mean, or use some other more appropriate metric, to see how our system performs depending on the user’s intentions and the actual results.

**Management Plan:**

We plan to work on alternating tasks, but who is developing which specific metric and algorithm is subject to change depending on our different class schedules (who is more free to work on the project a certain day or weekend). Because the backend and frontend of our application is very intertwined, we cannot split the tasks along those lines. By segregating the tasks in the following way described below while remaining flexible about trading tasks and responsibilities, we are better able to avoid merge conflicts with Git source control, as we know such conflicts are not fun to deal with from past experience working together.

*Step 0: Setting up iOS with OpenCV C++*

Melanie has worked more on capturing and saving images with the camera interface in the iPhone, she can set up the image picker controller class and initial Storyboard for the user interface. We can both look into how to properly integrate OpenCV C++ with iOS, since this is a new area for both of us.

*Step 1: Domain Engineering*

We will make efforts to find/obtain/produce training data together for Step 1 (i.e. both take clear pictures of our hands, to make sure the system can detect our different skin tones).

*Step 2: Data Reduction*

Nina can convert her code logic from Python to C++ for this step. Melanie can handle possible improvements to the old system. We are considering using HSV color values to identify the hand, rather than just black and white binarization, in order to improve her previous system, which required a black/dark background to contrast with the hand. We also hope to remove the previous constraint of requiring long sleeves that match the color of the background to hide the arms, by detecting the center of mass of the hand and chopping off the hand at the wrist. We intend however to keep the constraint of having only the hand and not the face in the picture, which will certainly require the inclusion of the arm in the picture for extended movements and drawing patterns, so it would be very beneficial to the system if users are free to wear any manner of apparel and are not restricted to long sleeves.

*Step 3: Parsing the Gesture Grammar and Tracking Hand Movement*

This step can be divided between implementing the real time tracking (Melanie) and implementing the grammar/drawing actions (Nina). The grammar implementation can further be subdivided into the five options for the five gestures if we both want to work on this task. There exists a body of literature dealing with marker tracking that we can both read and try to implement to have a more robust and improved tracking system.

Another smart way to divide the tasks (if we are both free to work on the project at the same period of time) is for Melanie to work on implementing the drawing grammar with OpenCV and the regular iPhone touch interface in Step 3, while Nina works on Step 2, in order to eliminate the waiting period common to linearly completed projects. In that case, while Melanie works on real time tracking, than Nina can improve on the drawing system, possibly adding that light art look to the drawing lines.

*Step 4: Performance Evaluation*

We will make efforts to find user studies jointly and enter the data into CSV files. Nina can refactor her Python code from Assignment 2 to process the CSV files, as this evaluation step does not need to be built into the mobile app itself and exists solely for development and assessment purposes.

*Step 5: Write the Report*

This we will perform jointly, probably in a Google document. Each of us will focus on the parts that we developed, as we should understand and be able to explain these parts best. If we complete the third step in a reasonable amount of time, we may try to improve the system into a shippable user application, but we understand that steps 4 and 5, especially evaluating the performance of the *visual* (and not user) interface, are most important, and we can always try to improve the user interface after this semester is over.

**References:**

Ghotkar, A.S., & Kharate, G.K. (2012). Hand segmentation techniques to hand gesture recognition for natural human computer interaction. *International Journal of Human Computer Interaction, 3(1)*.

Lee, D., & Lee, S-G. (2011). Vision-based finger action recognition by angle detection and contour analysis. *ETRI Journal*, *33(3)*.

Wong, S-K., Chen, K-M., & Chen, T-Y. (2014). Interactive sand art drawing using RGB-D sensor. *International Journal of Software Engineering and Knowledge Engineering*, 17-46.

Xiang, A.C., Schwarz, J., Harrison, C., Mankoff, J., & Hudson, S.E. Air+Touch: Interweaving touch & in-air gestures. *UIST ’14 Proceedings of the 27th annual ACM symposium on User interface software and technology*, 519-525.

**Alternative Ideas:**

* Using a set of photos, make a simple image mosaic. Basically, we will do a simple version of this: http://www.picturemosaics.com/tech/