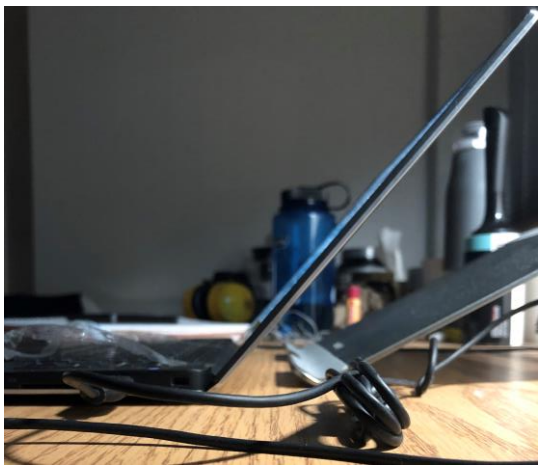
Visual Interfaces – Assignment 1 Write-Up – Omer Baddour

My goal was to create a program which could respond to gestures formed with the hands of a person using a computer. I wanted to do this because it allows a computer user to interface with any computer application using their hands – all they have to do is assign application-specific semantics to sequences of gestures. For example, if a person wishes to use their hands to cause a music playing program to play, pause and skip to the next song, or to manipulate their favorite browser, they simply have to cause the detected gestures to trigger the appropriate commands.

To capture images of gestures, I placed my phone camera on the bottom of my laptop screen above my keyboard, and took pictures from the center of my screen angled upwards. Taking pictures in this manner worked well for a number of reasons. The bulk of the background of captured images consists of ceiling, which tend to be plain in color and decoration, thus yielding fewer contributors to noise than if the background captured more wall. This noise could otherwise result in more false positives – gestures being identified which were not issued by the user – which would inhibit the success of the program. Angling the camera upwards minimizes the presence of other people – only those that walk close and are tall are captured. This further reduces the likelihood of false positives. Devices on desks and laps tend to be lower than eye level, resulting in screens being angled upwards. Inbuilt laptop cameras tend to be underneath the screen, and consequently angled upward. Thus, my setup mimics what is common, making it easy for people to use live input from this camera to facilitate real-time gesturing. Lastly, in the case that the user's visual input is not below their screen, the upwards angle of screens means that small cameras can be balanced on screens and point in angle my program is used to receiving data in. In either case, minimal modification to the user's likely normal setup is required, yielding convenience.

a.



b.

Setup from front and side

There are some disadvantages to reading visual input from below the user's computer screen. My gesture language uses height to distinguish gestures. Having a low camera angled upwards means that a user must exert more effort to issue high gestures than if the camera was placed higher and angled more downwards. This increases the effects of fatigue on accuracy of gesture classification. Another potential issue is that people who are not tech-savvy tend to associate the interfacing medium with their screens. The distance between the camera and the center of the screen could result in some misinterpretation of gestures for such users.

My program creates binary images of skin regions in the get_skin_binaries() function. The function casts all input images to their equivalent HSV representations, and then maps pixels within certain bounds of Hue, Saturation, and Value to 1, and all other pixels outside of these bounds to 0. I found the lower and upper bounds of Hue, Saturation and Value for skin detection by experimenting with different values over a large range of images in my is_skin() function. In the end, I found that optimal values are: 0 <= Hue <= 30, 25 <= Saturation <= 90, 178 <= Value <= 255. I chose to use this color space because it only uses one channel to describe color, namely Hue. Matching a range of Hue values with my skin tone and finding a range of tolerance values for fluctuations in light and shadows via Saturation and Value was easy to do by inspection via trial and error. The effects of changing parameters are predictable and intuitive, unlike a color scheme like RGB. These parameters proved to be pretty successful at removing most non-skin pixels, and retaining most skin pixels. However due to my realistic setup of imperfect background and lighting, I had to do more work to my images to correctly classify hands.

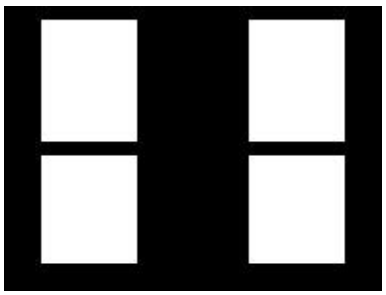
c.   *d.*

Example binary outputs from HSV filter

One problem I had was removing my face from images. My face always remained in the binary images since the Hue is similar to that of my hand. I used the fact that my face was always in the center of images, and that meaningful hand gestures are in one of four quadrants (top left, top right, bottom left, bottom right), to construct a binary image which filtered out the irrelevant regions (via bitwise and with binary image of skin).

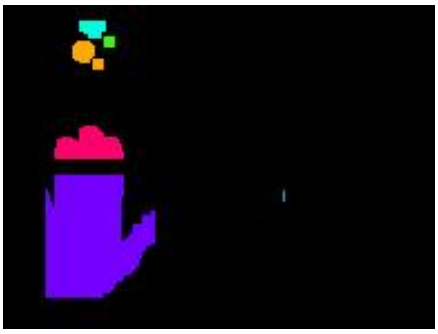
Another issue I faced was that some the binary image outputs removed too much skin from hands, as a result of sharp changes in lighting. This meant that when I ran the connected components algorithm on these binary images to identify distinct objects, hands would sometimes be split into multiple objects. To fix this problem in a way that would not cause other

problems, I added a slight blur to each binary image. Using kernel (5, 5) did a good job of blurring the right amount, such that disjoint objects became conjoined, and noise was not exaggerated too heavily.

e.   *f.*  *g.*

e. = d. after blur, e. bitwise and with f. yields g.

The example outputs above capture a lot of interesting information. Observe that the bitwise and between *e.* and *f.* actually causes the splayed hand in the bottom left corner to be split into two objects. This seems to interfere with the hand negatively, but actually plays an important role in my program - it decisively places the majority of the splayed hand in the lower left quadrant, and leaves only a small portion in the upper left quadrant. To get rid of noise, I use the areas of objects. I calculate the areas of each object by counting the number of label occurrences in the labelled image output from the connected components algorithm. I found that if the object area was greater than 2% of the total image's area, and was less than 12.5% of the total image's area, most objects besides hands were filtered out of the image. The lower bound filters out noise, such as the residual fingertips in the upper left quadrant of *g.*, and the upper bound means that the background is not considered an object. Another benefit of the splitting of the image into four quadrants is that very long, thin objects are segmented into several objects. The long, thin objects are likely to have a percentage area coverage of the image within the 2-12.5% bounds – thus splitting the object up will avoid this false positive.

h. 

Colored image of output of connected component algorithm on g., where each different color is a different object label. After executing the area check, only the purple object remains!

At this point, I was confident enough that the remaining objects were hands – it was time to find "the where" and "the what". I scanned the output of the connected component algorithm twice, once normally (row by row) to find the y position of the center of mass, and once using the transpose of the image (column by column) to find the x position of the center of mass. While doing this, I also kept track of the maximum number of label occurrences in a single row and a single column. After the two scans, I have enough information to find the center of mass of the labelled object of interest. If this value lies within one of the white quadrants defined by *f.*, I identify this quadrant as the location of the hand in the image. Otherwise, the object fails the location test, and is not classified as a hand. Finally, I compare the largest number of label occurrences in a single row and a single column. If there were more occurrences in a single row, I label the hand as a fist (since fists are wider than they are tall). Otherwise, I label the hand as a splayed hand (since splayed hands are taller than they are wide). These labels are returned in text form from the predictions function, and are then printed to stdout, and a new image is displayed, which is the original image with its classification transcribed on it in red text.

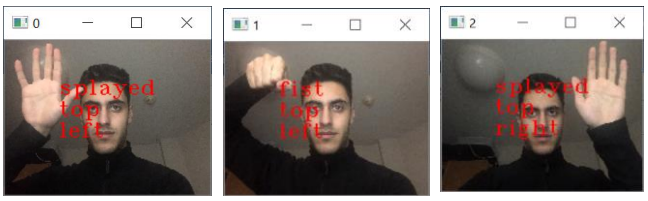
i.

Final output for image used as example thus far

I kept the grammar simple (eight symbol set) because I would rather prioritize ease of use than complexity. One can then use the simplistic grammar to build complex command semantics – binary serves as the perfect example of the feasibility of doing this.

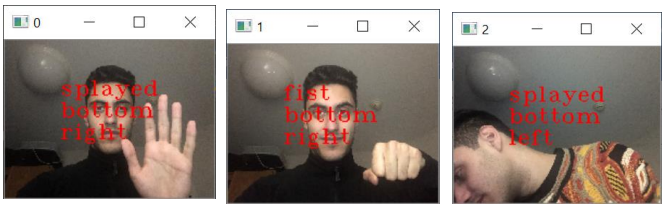
Sequences for 1.3

1.

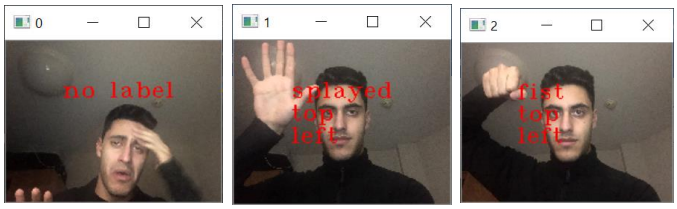


2.

3.



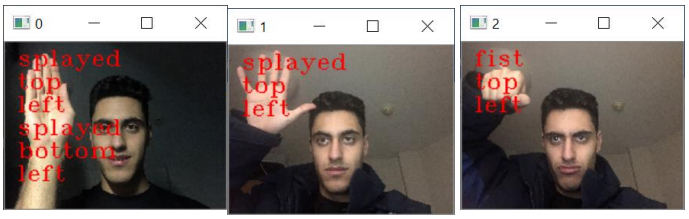
4.



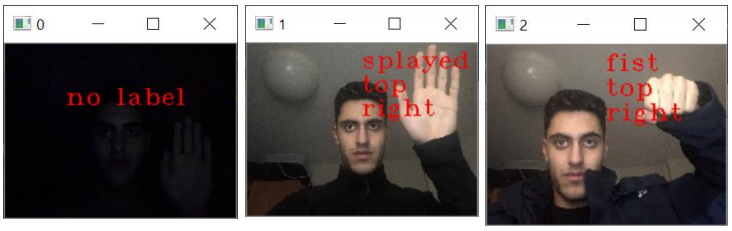
5.

6.

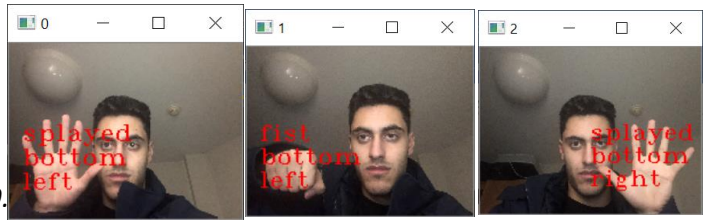
7.



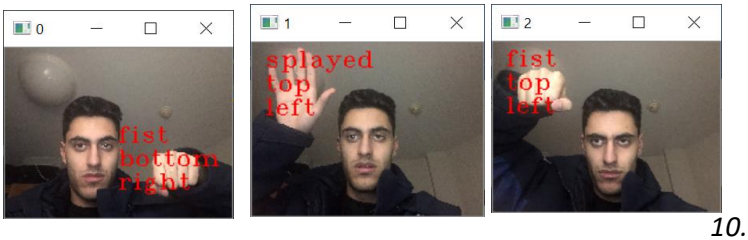
8.



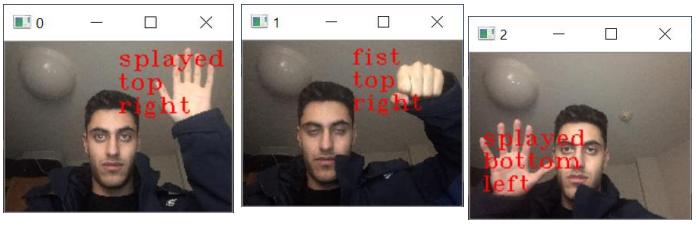
9.



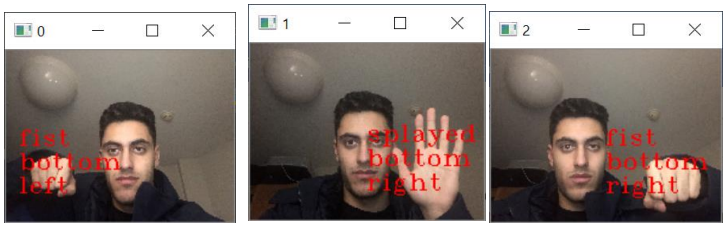
10.



11.



12.

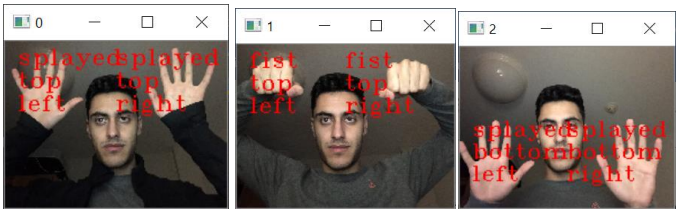


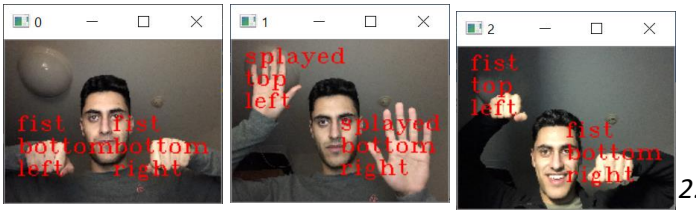
Observe that 8/12 sequences contain images which were all classified correctly, and the 4/12 sequences which contained images which were not classified correctly each contained only one misclassified image. Thus, in total, 32/36 images were classified correctly. Each image is unique, and there is variance present both in the lighting and with my outfits, demonstrating my program's competence at this classification task. The first case of misclassification occurs in sequence 3 image 2. I was bending over to get something out of my bag beside me, and due to the matching skin tone of my face and my hands, a gesture was captured (false positive). The shape of the object formed is wider than it is tall, hence the image was classified as a fist. This is a reasonable error to expect, and shouldn't be too much of a problem – the user's head will probably remain static in the middle of the screen the majority of the time, where it is ignored. The second case of misclassification occurs in sequence 4 image 0. I was feeling great fatigue, as you can see on my face, and was unmotivated to lift my hand high enough such that enough of it was visible to the camera. Consequently, my gesture was not recognized (false negative). The third case of misclassification occurs in sequence 7 image 0. I was wearing short sleeves in this image, thus my forearm, which is the same color as my hand, was detected and treated as a hand in the bottom left quadrant (false positive). My program identified two hands, because this is my creativity step in part 1.4 of the assignment (otherwise two hands would be restricted from the

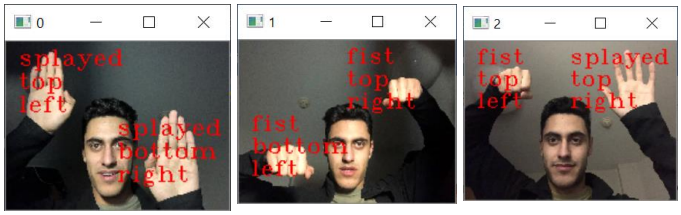
grammar, and ignored). Note that this case also made me realize that my classification text had the potential to conflict with other text when transcribed on the image – hence the labels are positioned differently from this point onwards. Lastly, the fourth case of misclassification occurs in sequence 8 image 0. All of the lights in my room were off, and my screen light was insufficient to illuminate my hand enough to be within the HSV range used to generate preliminary binary images (false negative).

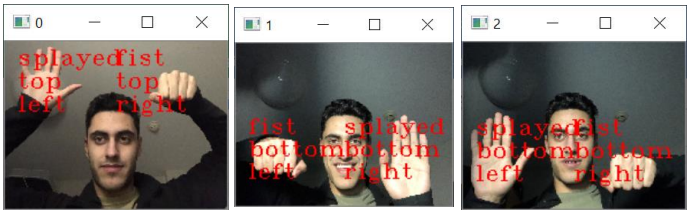
As previously mentioned, my creativity step was to introduce a second hand. This greatly expands my grammar, while keeping the simplicity of the gestures themselves.

Sequences for 1.4

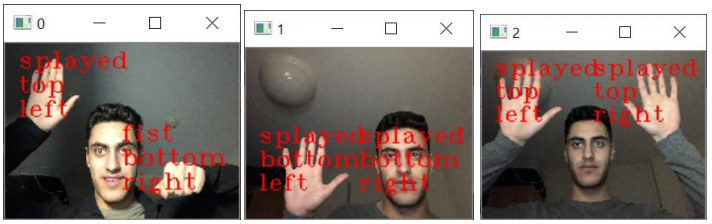
1. 

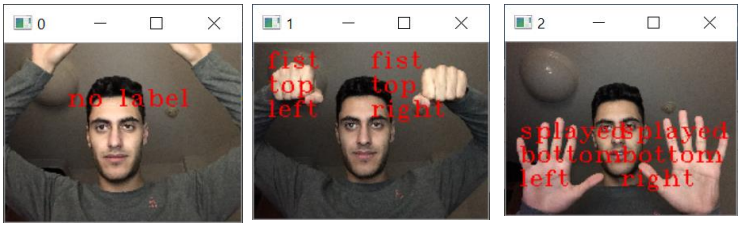
2. 

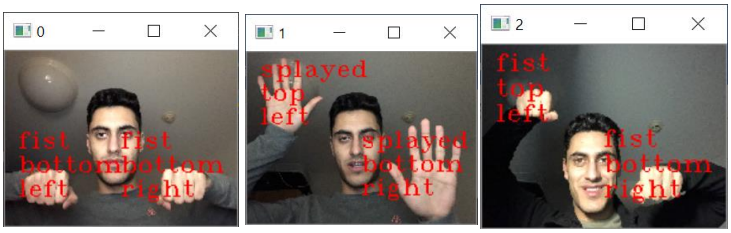
3. 

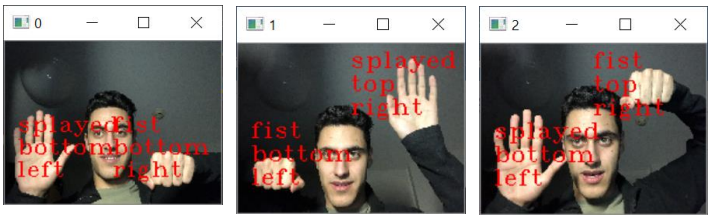
4. 

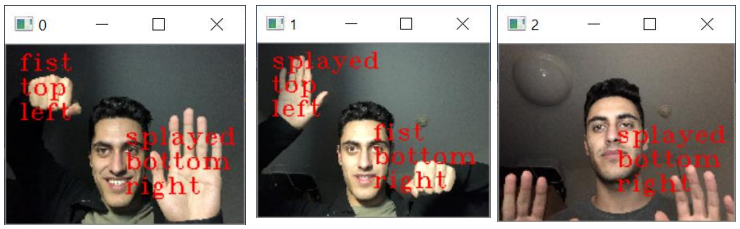
5. 

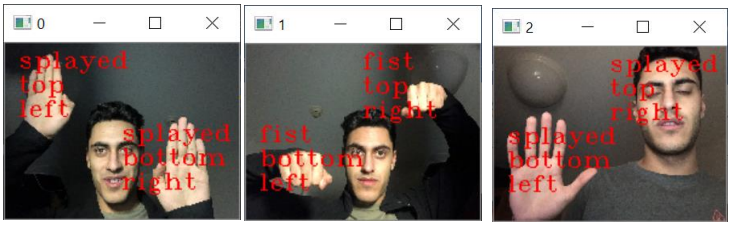
6. 

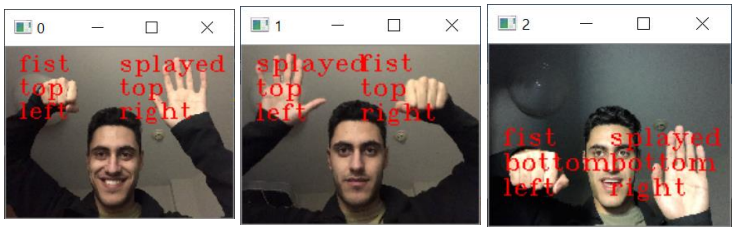
7. 

8. 

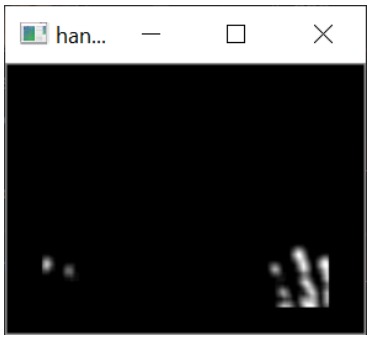
9. 

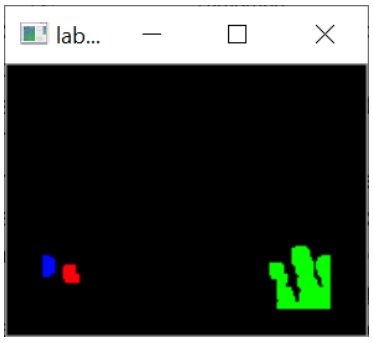
10. 

11. 

12. 

Again, 8/12 sequences were classified correctly, and again, the 4/12 sequences which contained images which were not classified correctly each contained only one misclassified image. Thus, in total, 32/36 images were classified correctly. Each image is unique, and there is a lot of variance present both in the lighting and with my outfits. My outfits also include color this time. Regardless, this accuracy rate shows that despite all of these new potential sources of error, my program was more than able to parse the images for skin, and classify the hand correctly for most cases. The first misclassification occurs in sequence 6 image 1. My head drifted sideways, and was mistakenly identified as a hand due to skin tone similarity (false positive). The next misclassification occurs in sequence 7 image 0. My hands were too high, meaning that not enough skin area was present in the upper quadrants to not be overlooked as noise (false negative). The third misclassification occurs in sequence 10 image 2. My hands were positioned poorly with respect to light and the camera. Looking more deeply, we see that a very small amount of my fingertips were identified as skin, after blurring the binary image produced by the get_skin_binaries() function. Additionally, the areas found were disjoint despite the blurring, and were also both very small, causing the objects to fail the area test and be discarded.

j. 



k.

Binary image and colored (by label) connected components image output for sequence 10 image 2

The final misclassification occurs in sequence 11 image 2, where again my face was in a quadrant, so was not zeroed out via bitwise and with image *f.*, and was classified as a splayed hand due to its geometry (height > width) (false positive). This recurring issue of faces being mistaken for gestures has an elegant solution – with the larger set of gesture symbols, as a result of being able to use a second hand, one simply has to allocate a single, two-hand gesture to mean "ignore input until this gesture is seen again". This gesture should require two hands, so that one misplaced face will not trigger it, thus keeping the program in an idle state of anticipation. So, when the user wants to move their face to reach something, or get up from their seat, they can avoid generating an unwanted gesture by first doing this meta-gesture.

In summary, I succeeded in creating the program I set out to create! A simple, image-based gesture system, which could be used as an interface with any application.