Goal:

In this project, I used the HVS color space for color-based segmentation. This project is targeted to flesh tone color detection. How it works is by using a set of flesh tone training images. This is where I get the data of different skin pigmentation and then use it to create a 2D histogram out of it. After that, I would use the data to filter out the colors of the test images that are not in the set of pigmentation data, this way, in theory, would detect the skin. In other words, color-base segmentation.

Step by Step:

```
# Training Images
    sampGun = cv2.imread('gun1_test.bmp')
    dataGun = skinToneData(sampGun)

    ...

    sampFour = cv2.imread('skin4.bmp')
    dataFour = skinToneData(sampFour)
```

There is a total of 7 train images but each went through the same process as shown above. First, it reads in the image then it calls the function *skinToneData*, this is where the frequency will be calculated.

```
def skinToneData(img):
    hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
    height, width, channel = hsv.shape
    freq = np.zeros((180, 256)) # Hue: 0 - 179 | Sat: 0 - 255

    for i in range(height):
        for j in range(width):
            hue = hsv[i][j][0]
            sat = hsv[i][j][1]
            freq[hue][sat] += 1
    freq_norm = cv2.normalize(freq, None, 0, 255, cv2.NORM_MINMAX, cv2.CV_8U)
    return freq_norm
```

From the main function, it calls the function *skinToneData*. First, this function converts the image from the main function from BRG to HSV since I need the hue and saturation of the image. Then it gets the shape of the HSV images (height, width, and channel – channel is the H, S, V). From the cv2 documentation, it mentioned the min and max value for Hue is ranged 0 – 179, and Saturation is ranged 0 – 255. Therefore, I just increase both by 1 and created an array called freq to store the value. For loops are used to traverse through all the pixels of the training images. While traversing, every hue and saturation of a pixel is recorded in a freq array. The way I separated the hue and sat is by different channels. So, the hue is channel 0, saturation is channel 1, and value is channel 2. After going through all the pixels, I normalized the array and return it to the main.

```
result = dataGun + dataPoint + dataJoy + dataOne + dataTwo + dataThree + dataFour
Histo2D(result)
```

This is in the main function. After returning from the *skinToneData* all the trained image frequency data is combined into one array and set it to variable result. This result will then pass into the Histo2D function.

```
def Histo2D(freq):
    fig, ax = plt.subplots(figsize = (10, 10))
    ax.imshow(
        freq, cmap = plt.cm.nipy_spectral,
        extent=[
            0, 180,
            0, 256
        ]
    )
    ax.set_title('2D Histogram of Trained Images')
    ax.set_xlabel('Hue')
    ax.set_ylabel('Saturation')

    plt.savefig('result.png')
    plt.show()
```

This function creates the 2D histogram that is attached to this report. What it does is takes the data from all the trained images and plots them. In this case the higher the occurrence the hotter – brighter – the color. The axis is ranged from 0 – 180 and 0 – 256 just like the range for hue and saturation.

```python
# Testing Image
    img = cv2.imread('gun1.bmp')
    final_gun = SkinDetect(result, img)

    img = cv2.imread('pointer1.bmp')
    final_pointer = SkinDetect(result, img)

    img = cv2.imread('joy1.bmp')
    final_joy = SkinDetect(result, img)
```

Returning to the main function: each of the sectioned code above does the same thing therefore, I will just explain one.

First, we read in a test image, then calls the *SkinDetect* function (explain later).

```python
def SkinDetect(freq, img):
    hsvImg = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
    height, width, channel = hsvImg.shape

    for i in range(height):
        for j in range(width):
            hue = hsvImg[i][j][0]
            sat = hsvImg[i][j][1]
            if freq[hue][sat] == 0:
                hsvImg[i][j][2] = 0
    hsvImg = cv2.cvtColor(hsvImg, cv2.COLOR_HSV2BGR)
    return hsvImg
```

This is the function that was called from the main function. It takes in the data from the *SkinToneData* and the test image from the main. First, it converts the image from BGR to HVS since I need the hue, saturation, and value data. Then I get the shape of that HSV image. Now in the for loop, we traverse every pixel and tries to find and "match" when the hue and saturate from the *SkinToneData* is 0. When that happens, then we set the value of the HSV test image to 0 which is another word setting the brightness to 0. After going through all the pixels, I made sure to convert it back to BGR and then return the image to the main function.

```
    # This does nothing to the logic or algorithm.
    # Just used to show images in one window.
    allImg = np.hstack((final_gun, final_pointer, final_joy))

    cv2.imwrite("allSkinImage.bmp", allImg)
    cv2.imwrite("gunSkinImage.bmp", final_gun)
    cv2.imwrite("pointerSkinImage.bmp", final_pointer)
    cv2.imwrite("joySkinImage.bmp", final_joy)
    cv2.imshow("Segmentation Image", allImg)
    cv2.waitKey(0)
```

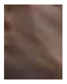
This is what I used to show the final skin detected images in one window using cv2. As it says in the comments, this part has nothing to do with the logic or algorithm. This is only used to show and save my images – not the histogram; 2D histogram is saved and shown through the Histo2D function explained previously.

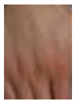
What expected to see if the parameter changed:

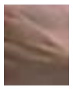
If I change the beta value of the normalization that will change the intensity of the histogram output. For example, changing from 255 to 1000 will increase its intensity. I did see any major effect on skin detection.

If I increase the pool size of training images, it could increase the accuracy or decrease depending on the test images and the skin tone I used. Fortunately, when I increase the image from 3 train images to 7 it helped detect more skin.

Trained Images Used:

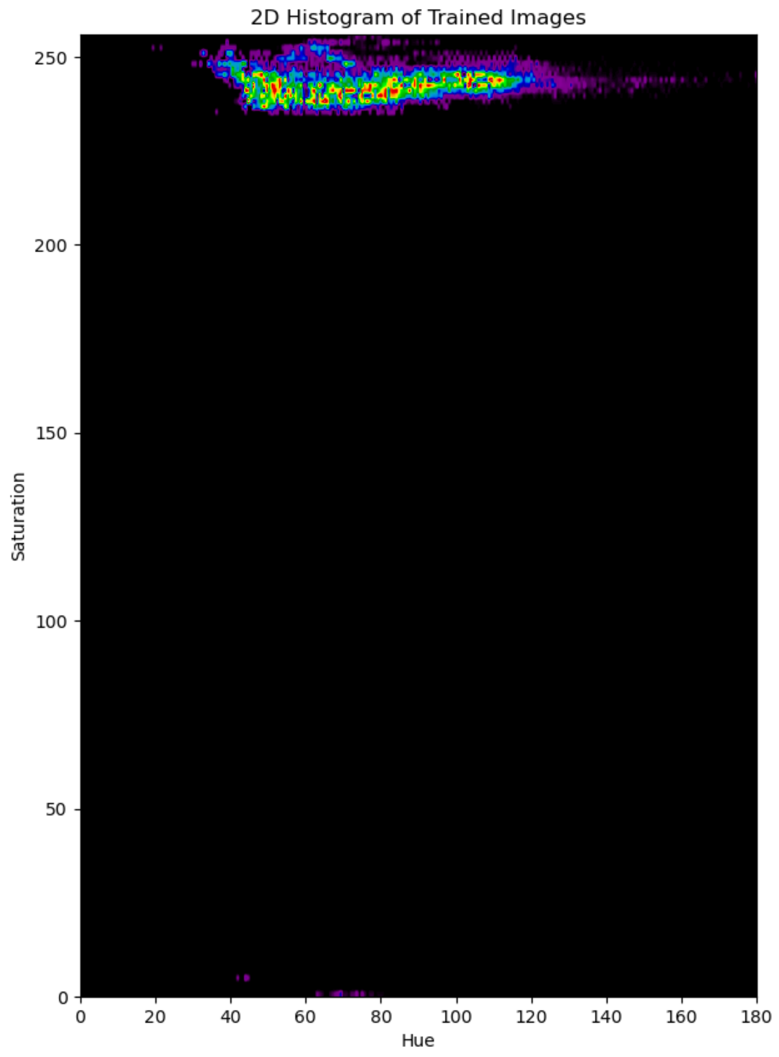
From gun1 

From joy1 

From pointer1 

From Online:

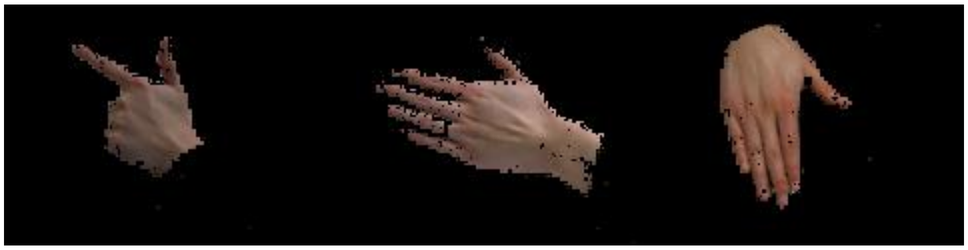
Histogram Model Created:



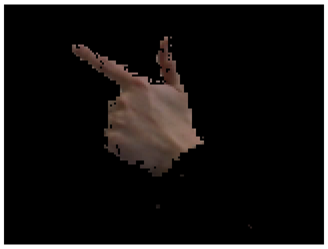
How I would use this 2D histogram to classify the segment is by color intensity. What I mean is the bright parts of the histogram is the color that is trained and will detect when a test image is filtered through using this. As you see only the top portion (and couple scavengers at the bottom – likely due to inaccurate image cropping) of the histogram is bright that should be the color pallet that it is trained to detect which is the skin tone. The more intense/bright the color the more frequent it appears therefore it detects the skin tone.

Results:

All three images combined:

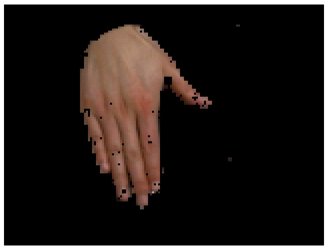


Gun image:

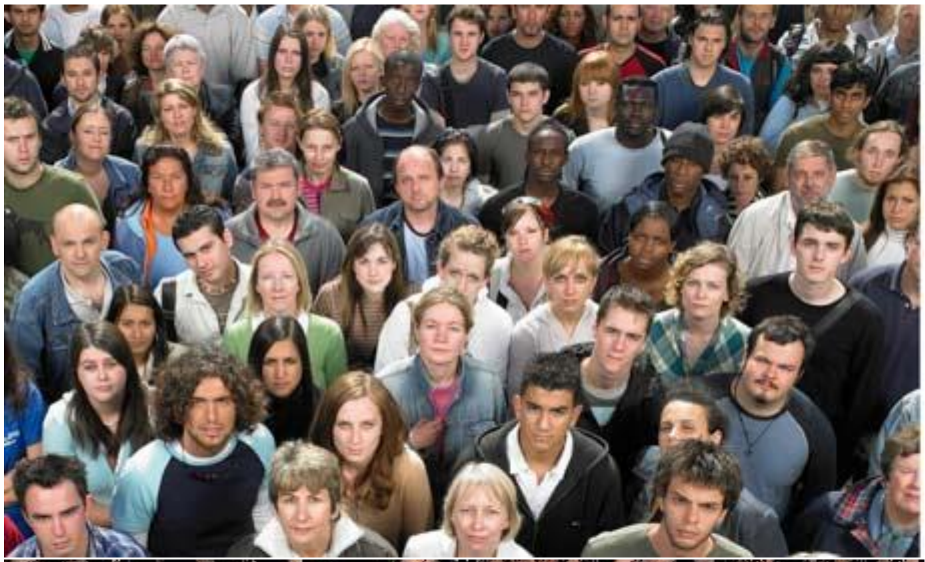


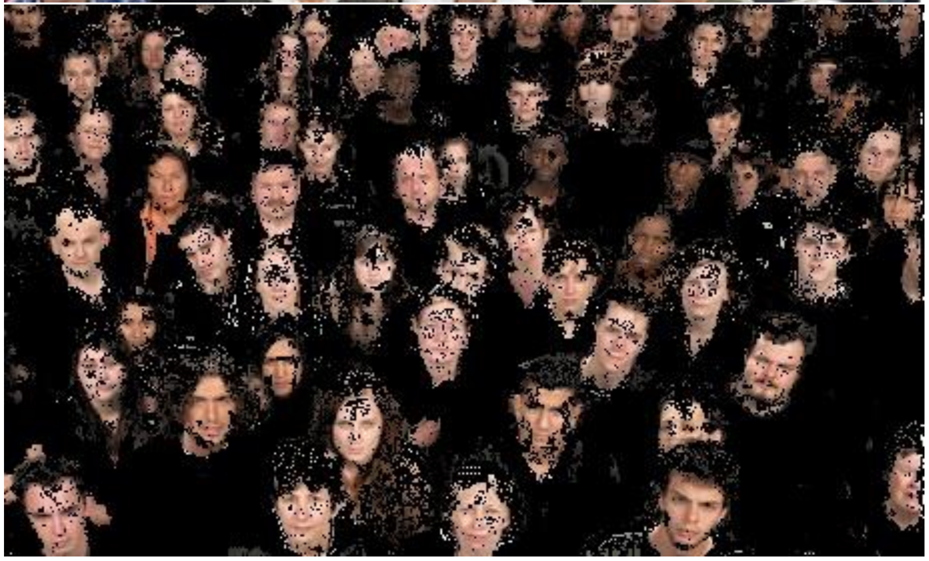
Pointer image:

Joy image:



Extra Image Tested:





Analysis:

From the resulting images, you could see that it is not as clean of skin detection. This is due to various pigmentation that humans have and even if the pigmentation is the same, the lighting from different angles can also affect the skin tone. If everything is perfect (lighting and angle) the amount of training data I have is not enough to detect every skin pigmentation. So, I believe that by increasing the pool size of the training image we could lower the noise and inaccurate skin tone detection. Possibly combining with different image manipulation could improve detection like gaussian-based color segmentation which, unfortunately, I did not have time to do.