Assignment 5 (45/40 points)

Problem 1: Edge Detection with Color (10/10)

Choose a color image. Convert it to grayscale using an average of all three color channels and find the edges (anyway you like). Then, convert the original image to HSI, and find edges on the I component using the same method you used for the grayscale image. Finally, find edges on the H component using the same method. Compare the three edge images you found and discuss similarities and/or differences that you notice. Make sure to include in your comparison which method gave you the best results and why you think so.

I performed edge detection on an image using various components, including grayscale, HSI (Hue, Saturation, Intensity) color space, and individual RGB components. The grayscale image (subplot 2) was quite effective at capturing intensity variations, which made it a good starting point for edge detection. When I applied the Canny edge detector to the intensity component (subplot 4) in the HSI space, it excelled at highlighting fine details and transitions related to luminance. On the other hand, when I focused on the H component (subplot 5), it emphasized color differences, proving useful for segmenting objects based on their hues.

CODE:

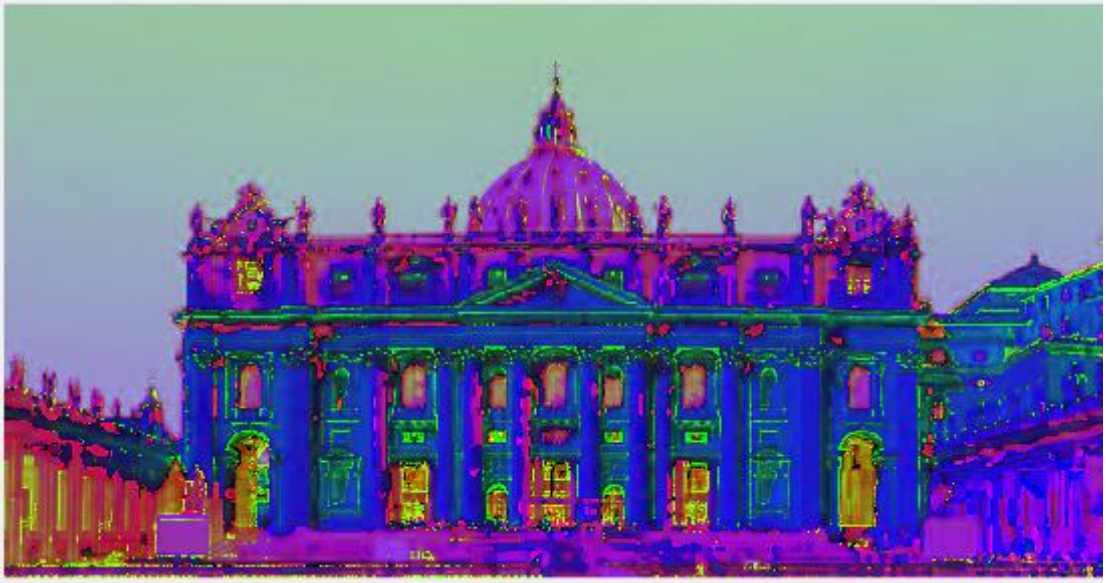
```
OG_Image = imread('Vatican.jpg');
G_Image = mean(OG_Image, 3);

% Using Canny edge detector to find edges
grayEdges = edge(G_Image, 'Canny');

hsiImage = rgb2hsv(OG_Image);

% Extract the H, S, and I components
H = hsiImage(:, :, 1);
S = hsiImage(:, :, 2);
I = hsiImage(:, :, 3);

% Find edges in the I component using the Canny edge detector
IEdges = edge(I, 'Canny');

% Find edges in the H component using the Canny edge detector
HEdges = edge(H, 'Canny');

subplot(2, 3, 1);
imshow(OG_Image);
title('Original Image');

subplot(2, 3, 2);
imshow(G_Image, []);
title('Grayscale Image');

subplot(2, 3, 3);
```

```
imshow(hsiImage);
title('HSI Image');

subplot(2, 3, 4);
imshow(IEdges);
title('Edges in I Component');

subplot(2, 3, 5);
imshow(HEdges);
title('Edges in H Component');

subplot(2, 3, 6);
imshow(grayEdges);
title('Edges in Grayscale Image');
```

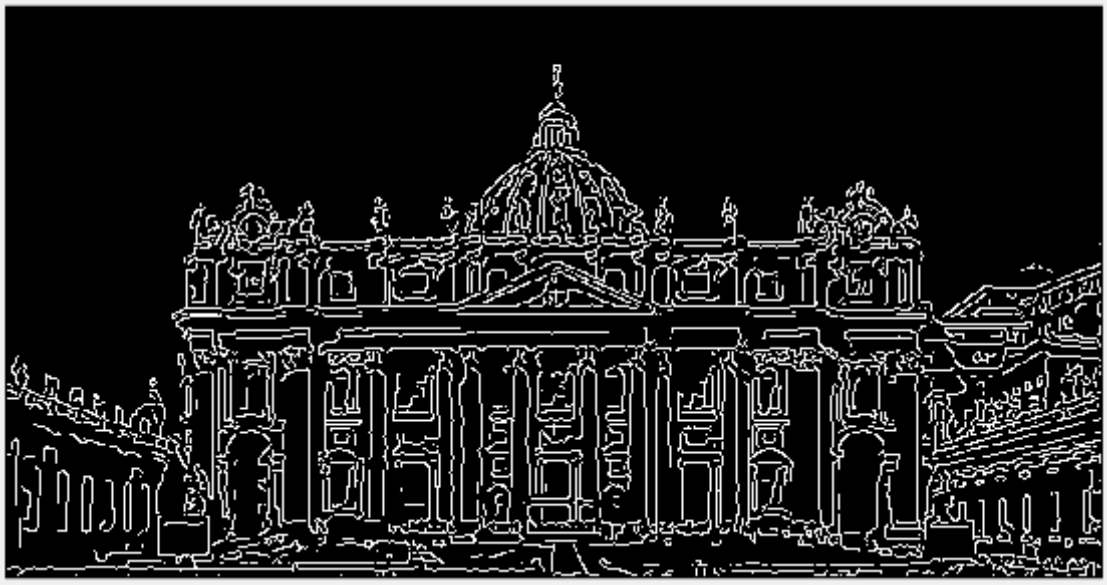
O/P:

Grayscale Image

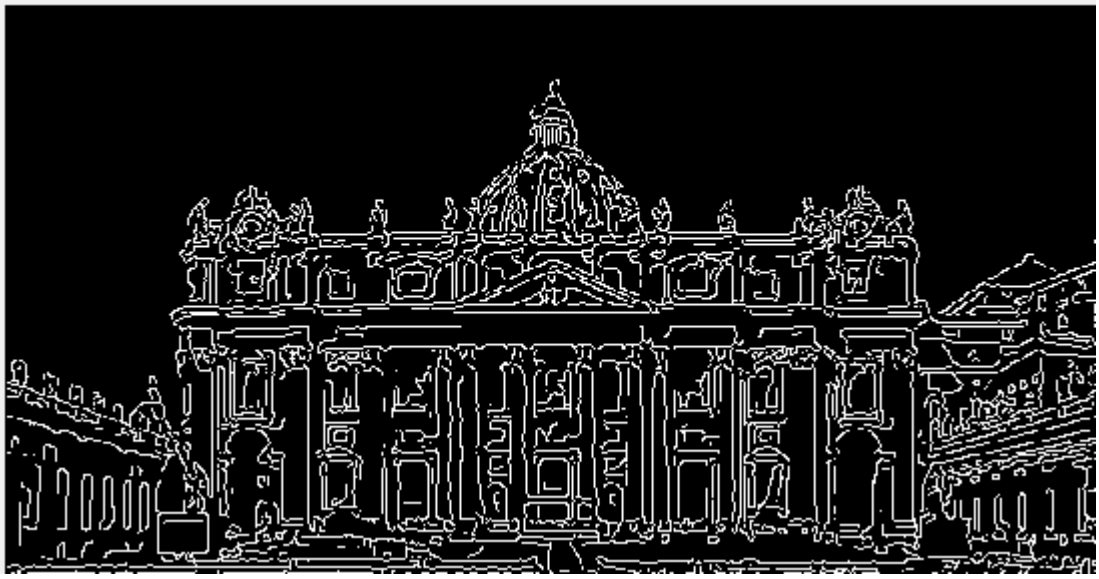
HSI Image

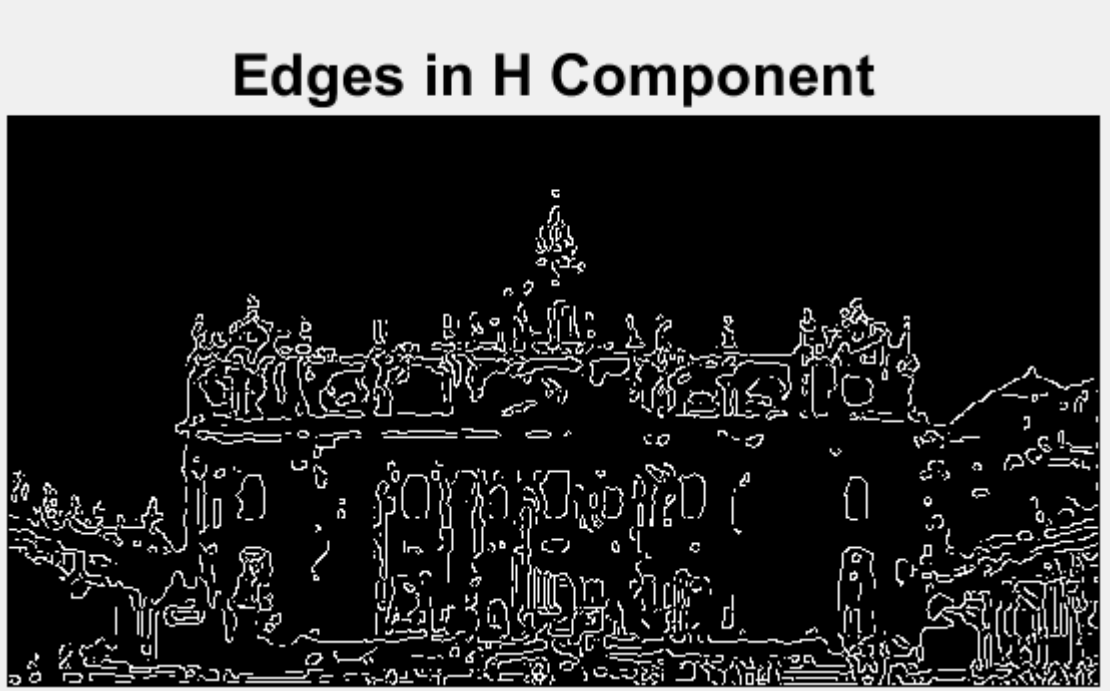


Edges in Grayscale Image



Edges in I Component





Problem 2: Color Segmentation

A natural cue to use in segmenting objects from their surroundings in images is color. **This problem contrasts segmenting color regions using red, green, and blue thresholds** (aligned with the RGB axes of color space) **with segmentation using hue, saturation, and intensity bounds** (aligned with the coordinate system of HSI or HSV space). For this assignment, it will be more educational to choose an image with strongly colored objects. For example, an image of party balloons works well – make sure they are on a dark or light background.

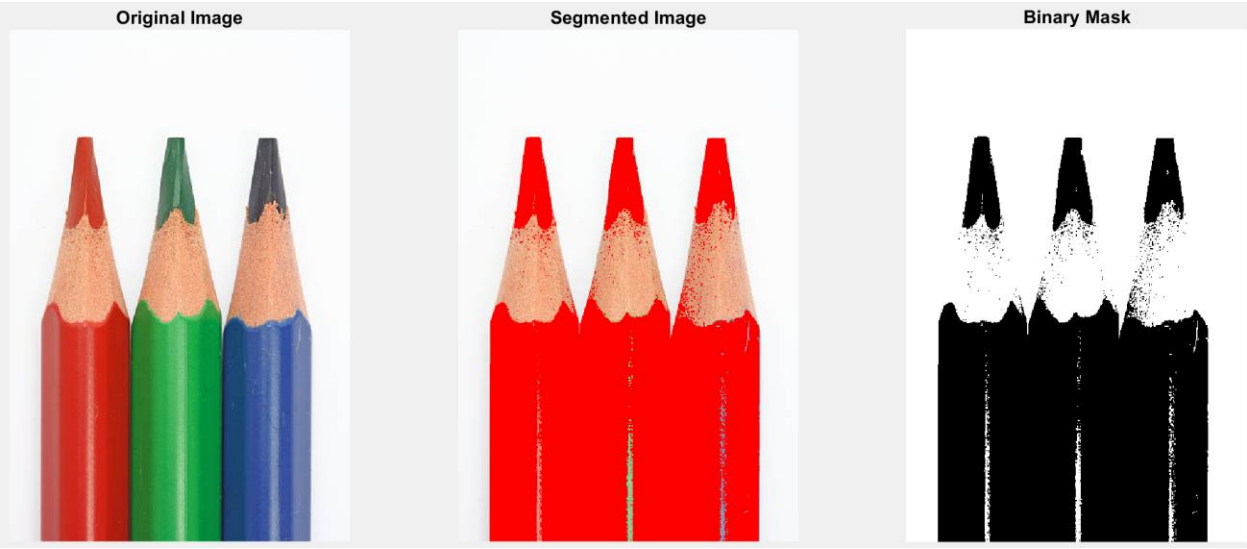
a) **(10/10)** First, segment your image into objects and background using a threshold on the intensity of the pixels. You can get a grayscale image from an RGB image simply by averaging the three color components of each pixel. Demonstrate your segmentation by replacing the background pixels with a visually distinct color. (In fact, just the reverse -- replacing blue or green pixels with those of some preset image -- is the technique used in TV or movies to superimpose objects against some preset background. Since thresholding is used, this (and not fashion) is why so few weathercasters wear saturated blue items. This technique is called travelling matte. Blue is good because it turns black under a red filter; green is good because most digital cameras have less noise in the green channel. Matte techniques have become even more sophisticated as digital video becomes more sophisticated).

CODE:

```
OG_Image = imread('RGB_Pencils.jpg');
```

```
G_Image = mean(OG_Image, 3);
iThreshold = 150;
binaryMask = G_Image > iThreshold;
backgroundMask = ~binaryMask;
backgroundReplColor = [255, 0, 0];
backgroundReplColor = cast(backgroundReplColor, class(OG_Image));
segmentedImg = OG_Image;

for channel = 1:3
   segmentedImg(:, :, channel) = segmentedImg(:, :, channel) .* uint8(binaryMask) +
backgroundReplColor(channel) * uint8(backgroundMask);
end

figure;
subplot(1, 3, 1);
imshow(OG_Image);
title('Original Image');

subplot(1, 3, 2);
imshow(segmentedImg);
title('Segmented Image');

subplot(1, 3, 3);
imshow(binaryMask);
title('Binary Mask');
```

O/P:

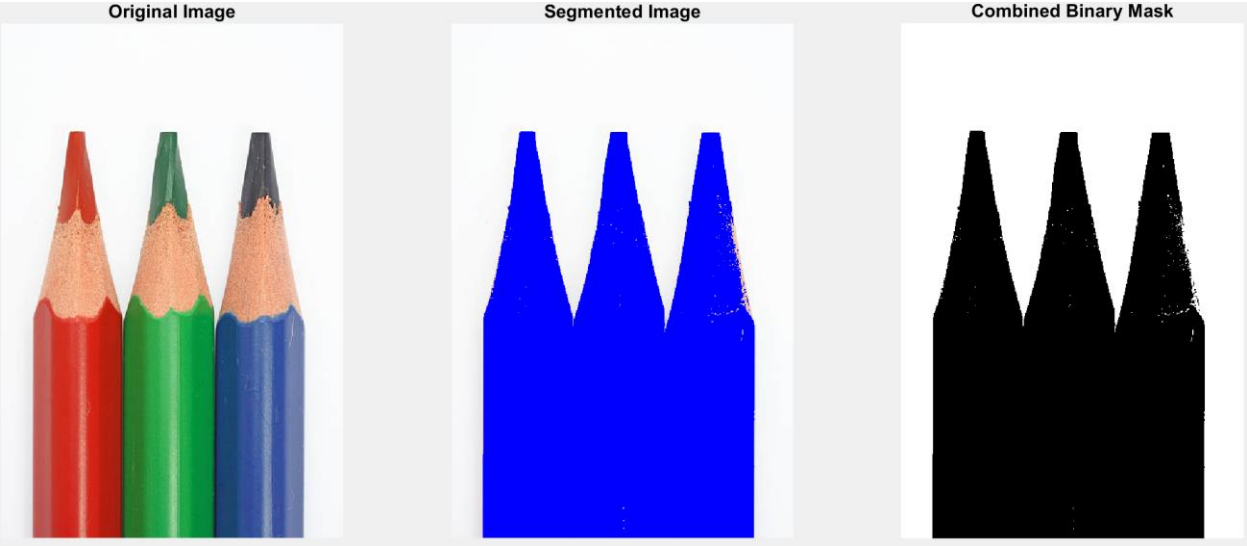


b) **(10/10)** Second, use thresholds in each RGB color band to isolate the objects in your image. Again, display the results by "bluing out" the intended object region. Provide some commentary on how the segmentation succeeded and failed. (**481 Students (5)**: use an automatic thresholding approach instead of choosing thresholds by hand. Hint: look at `otsuthresh`.)

CODE:

```
thresholdR = graythresh(OG_Image(:, :, 1));
thresholdG = graythresh(OG_Image(:, :, 2));
thresholdB = graythresh(OG_Image(:, :, 3));

binaryMaskR = OG_Image(:, :, 1) > (thresholdR * 255);
binaryMaskG = OG_Image(:, :, 2) > (thresholdG * 255);
binaryMaskB = OG_Image(:, :, 3) > (thresholdB * 255);

combinedBinaryMask = binaryMaskR & binaryMaskG & binaryMaskB;

segmentedImg = OG_Image;

backgroundReplColor = [0, 0, 255];
backgroundReplColor = cast(backgroundReplColor, class(OG_Image));

for channel = 1:3
   segmentedImg(:, :, channel) = segmentedImg(:, :, channel) .* uint8(combinedBinaryMask) +
backgroundReplColor(channel) * uint8(~combinedBinaryMask);
end

figure;
subplot(1, 3, 1);
imshow(OG_Image);
title('Original Image');

subplot(1, 3, 2);
imshow(segmentedImg);
title('Segmented Image');

subplot(1, 3, 3);
imshow(combinedBinaryMask);
title('Combined Binary Mask');
```

O/P:



Observations:
In question A, the segmentation method based on intensity, using grayscale images and fixed thresholds, effectively highlighted regions with high brightness. However, it failed to consider color variations, making it less suitable for images with strongly colored objects. The approach was limited in distinguishing objects with similar intensity values but different colors, resulting in incomplete object-background separation. In contrast, question B utilized automatic RGB thresholding with Otsu's method, offering a more successful approach for preserving color information and separating objects from the background. This method adaptively determined optimal thresholds for each color channel, enhancing object recognition based on color properties. Although the effectiveness of RGB thresholding depends on image lighting and color distribution, it demonstrated greater versatility in handling color-based segmentation, making it better suited for images with strongly colored objects against various backgrounds.

c) **(10/10)** Repeat the segmentation using thresholds of hue in HSI space. MatLab has a function for converting from RGB to HSI (MatLab calls it HSV):

```
B = rgb2hsv(A);
```

Note: When described in matlab's HSV space, the hue (first component) of a pixel ranges from 0.0 (red) to 1.0 (red again), passing through orange, yellow, green, cyan, blue, purple, and magenta along the way. The second component, saturation, varies from 0.0 (grayscale) to 1.0 (completely saturated -- no white at all). The final component, intensity (or "value"), also ranges from 0.0 (no intensity) to 1.0 (max intensity).

There is also an inverse function

```
A = hsv2rgb(B);
```

It returns an RGB image with pixel components between 0.0 and 1.0

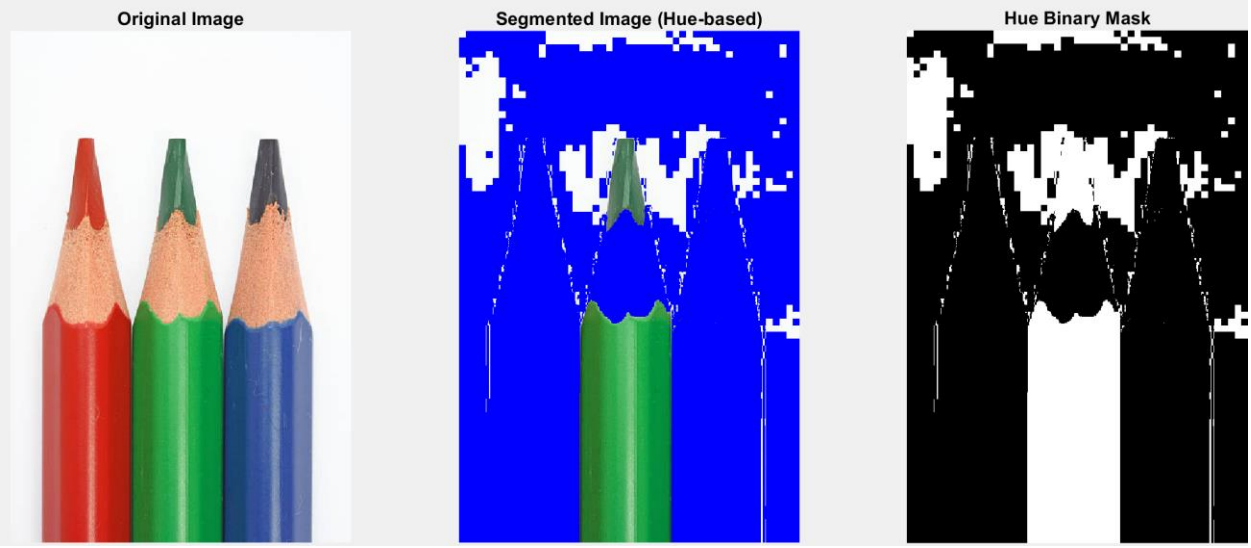
Note: A "threshold" of the hue component of pixels must be an interval, because the hue actually wraps around and is best envisioned as a circle. Thus, to segment a blue region, you need to accept only hues around 2/3 (0 = red, 1/3 = green, 2/3 = blue).

How does your segmentation based on hue differ from your segmentations based on RGB?

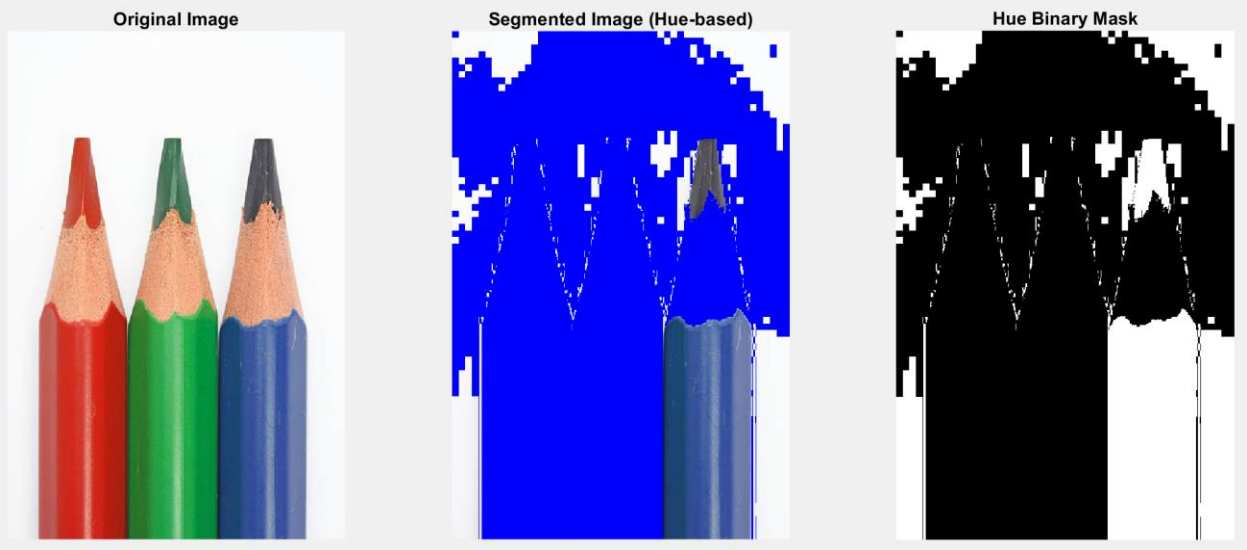
```
hsvImg = rgb2hsv(OG_Image);

hueImg = hsvImg(:, :, 1);

hueThresholdLow = 0.15;
hueThresholdHigh = 0.55;

hueBinaryMask = (hueImg >= hueThresholdLow) & (hueImg <= hueThresholdHigh);

segmentedImg = OG_Image;

backgroundReplColor = [0, 0, 255];
backgroundReplColor = cast(backgroundReplColor, class(OG_Image));

for channel = 1:3
   segmentedImg(:, :, channel) = segmentedImg(:, :, channel) .* uint8(hueBinaryMask) +
backgroundReplColor(channel) * uint8(~hueBinaryMask);
end

figure;
subplot(1, 3, 1);
imshow(OG_Image);
title('Original Image');

subplot(1, 3, 2);
imshow(segmentedImg);
title('Segmented Image (Hue-based)');

subplot(1, 3, 3);
imshow(hueBinaryMask);
title('Hue Binary Mask');
```

O/P:

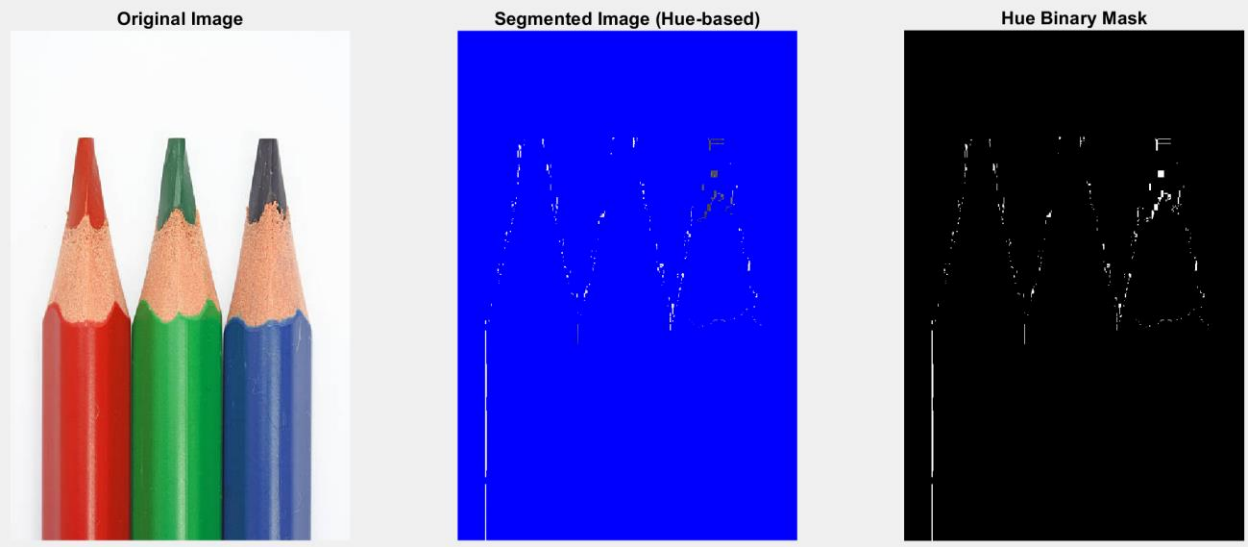
For Threshold 0.15 to 0.55



For Threshold 0.55 to 0.90



Threshold Range O.75 to 0.95



Observation:

In the first range (0.15 to 0.55), the segmentation left out the green object because green hues typically fall within this range. Since pixel colors are quantized, the selected hue threshold range might not fully encompass all variations of green, causing the omission of the green object.

In the second range (0.55 to 0.90), the segmentation left out blue objects, possibly due to the abrupt transition from blue to other colors within this range. The digital discretization could lead to difficulties in distinguishing the exact boundary between blue and adjacent hues, resulting in incomplete masking.

In the third range (0.65 to 0.95), the complete masking of blue objects suggests that this range closely matches the hue range of blue in the image. The success in this case can be attributed to a more precise alignment of the threshold range with the actual hue distribution in the image.

General submission instructions:

(a) Be kind to your aging, over-worked professor and submit only a single document. This can be pdf, MS Word, OpenOffice, etc. Do not submit a zip file.

(b) Your single document should include the input image for your problem, if required, and answers to each of the sub-problems (text, image or both, as appropriate). Your document should also include code that you wrote to generate your answers.

(c) You may use any images you like for the programming; I encourage you to use images that might be useful/interesting for your final project.

(d) Feel free to use whatever functions MatLab supplies, except where otherwise specified. Also feel free to write your own, if you are so inclined; it will take more time, but you will gain a deeper understanding of the material. It is one thing, for example, to

implement Otsu thresholding using `otsuthresh`, quite another to write an thresholding technique yourself.

(e) Point values for each question are indicated as ***x/y*** in which ***x*** is the point value for 481 students and ***y*** is the point value for 381 students.