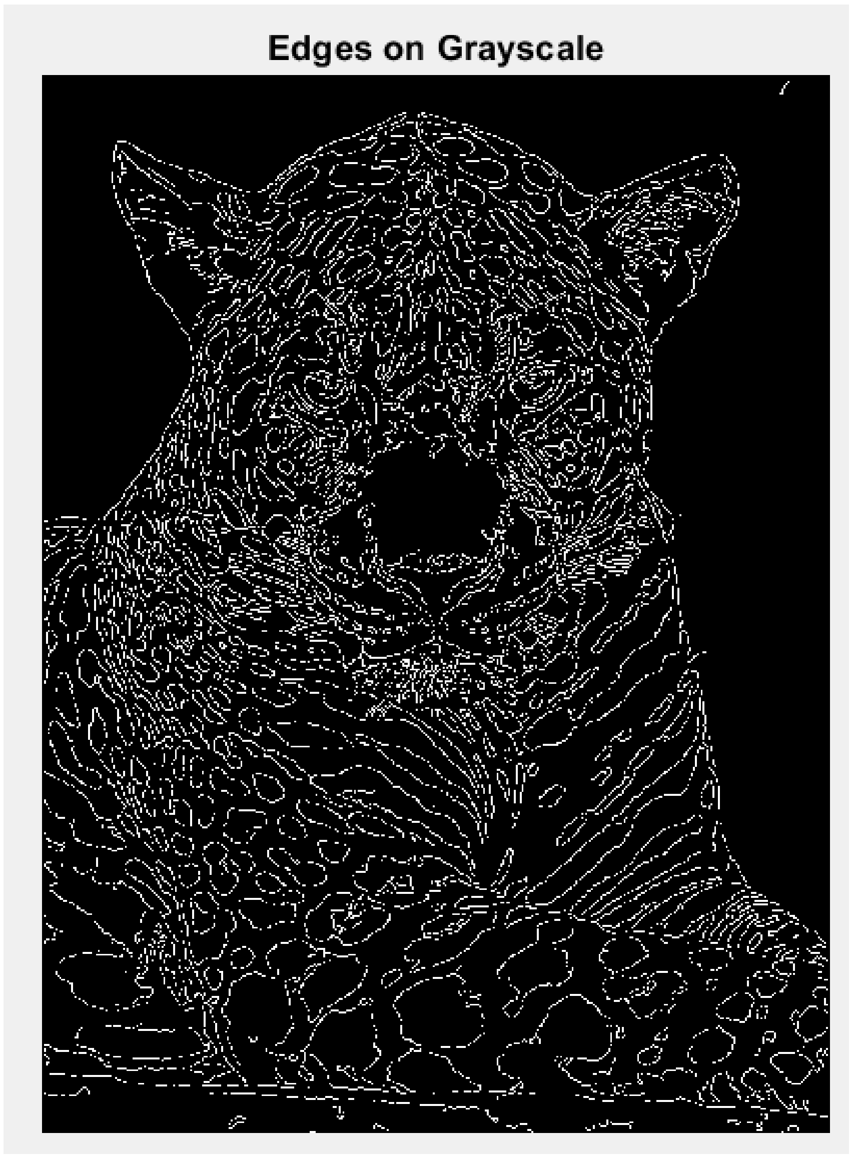
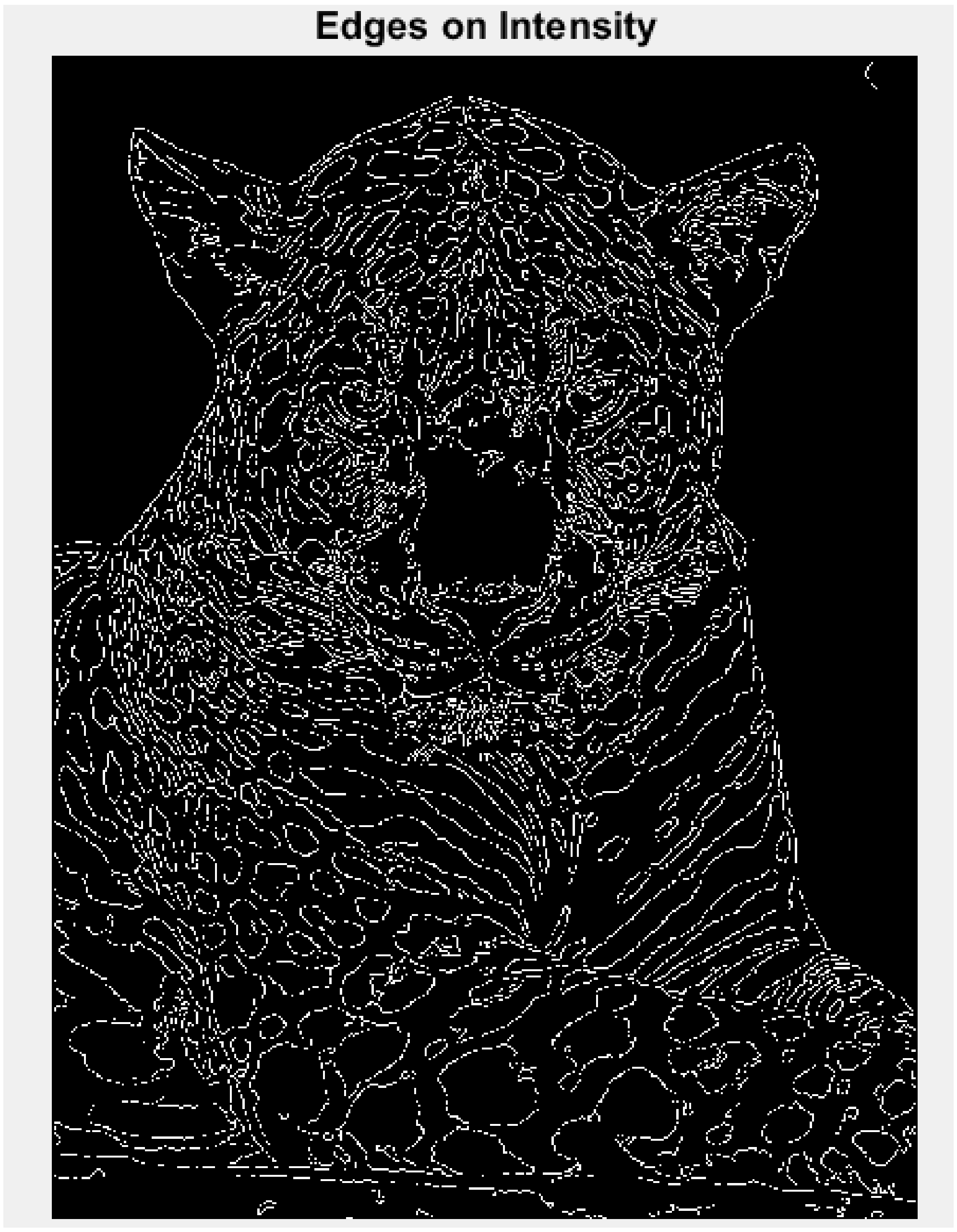
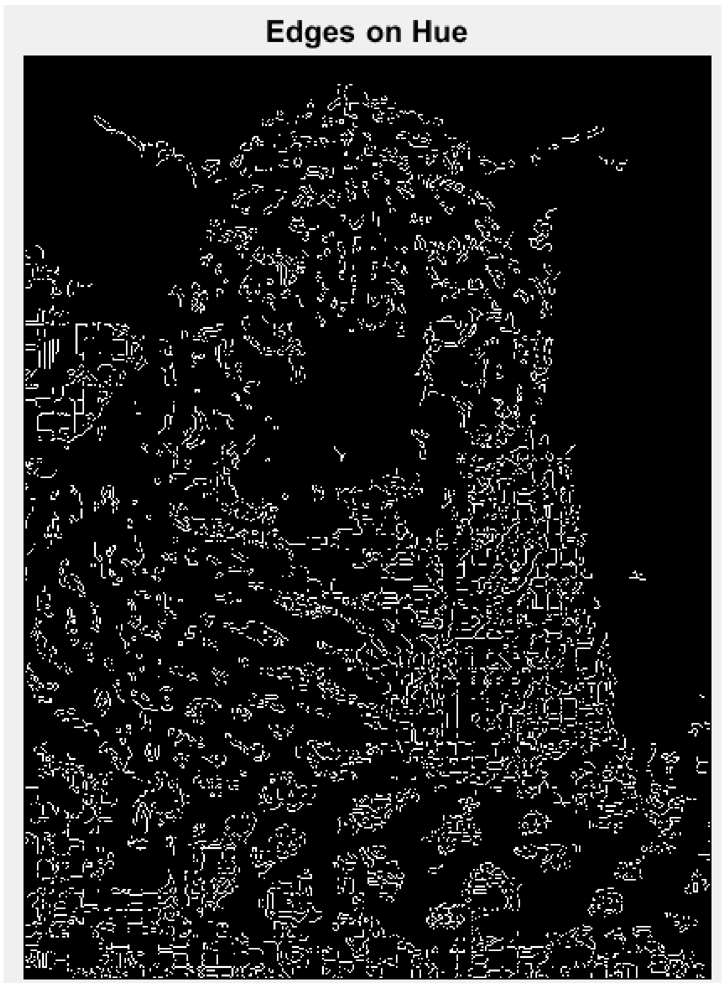
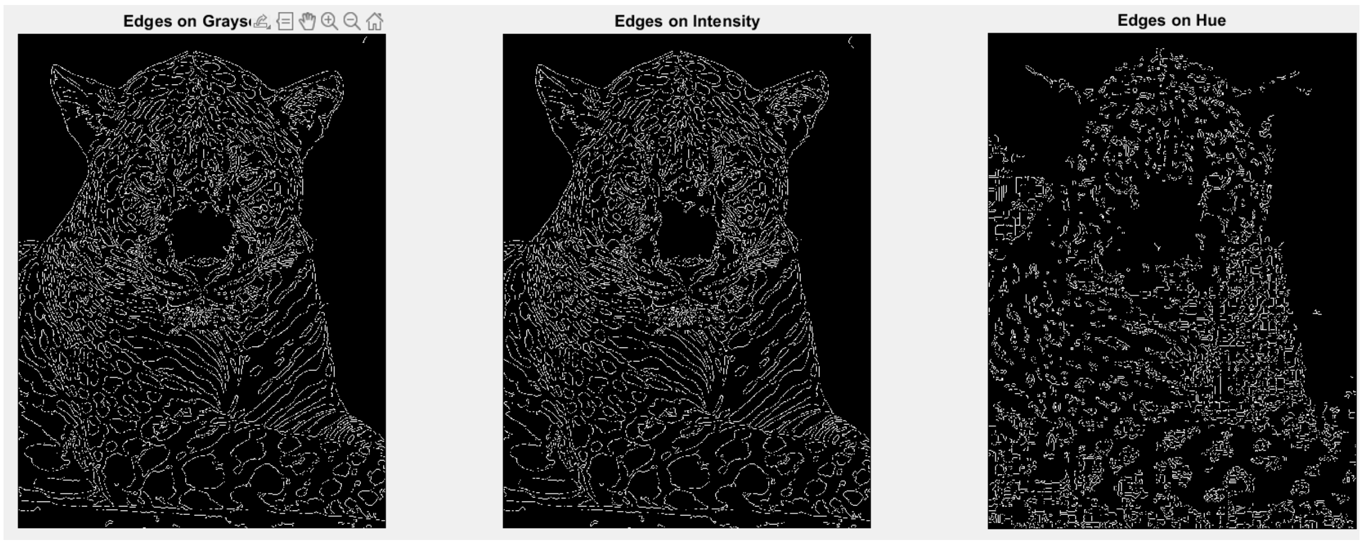
Problem 1: Edge Detection with Color

1) **Problem 1: Edge Detection**




Edges on Intensity


Edges on Hue


Edges on Grays
Edges on Intensity
Edges on Hue

Edges Detection on grayscale

- Grayscale edge detection finds edges based on brightness changes, ignoring color.
- It's sensitive to noise and may create false edges
- In noisy areas due to small intensity fluctuations

Edge Detection on Intensity Component (HSI Space):

- Emphasizes changes in brightness (intensity).
- Ignores color information.
- Sensitive to intensity variations but not color.

Edge Detection on Hue Component (HSI Space):

- Captures edges related to changes in color (hue).
- Highly sensitive to color changes.
- Does not consider intensity variations.

Here are Differences and Use Cases:

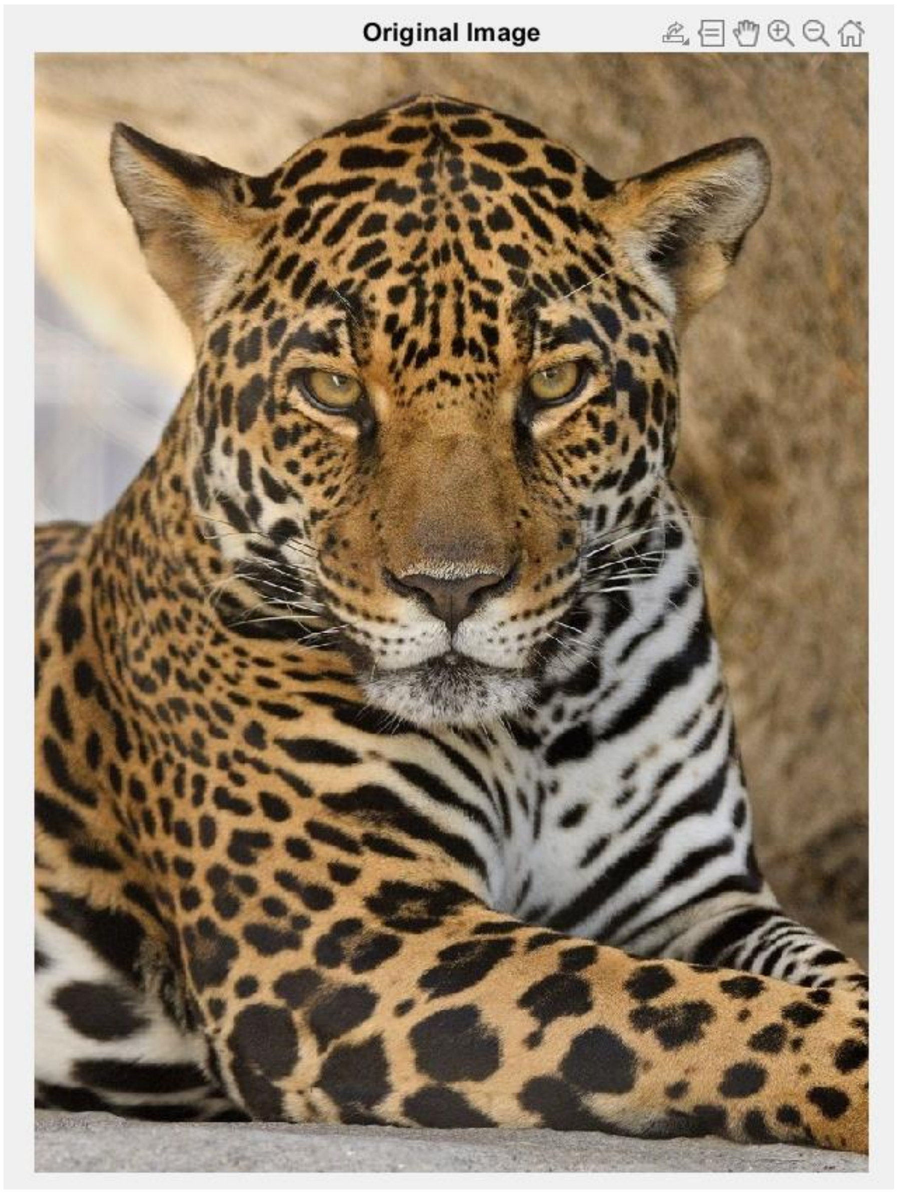
- Grayscale and intensity-based edge detection are suitable for general structural analysis.
- Hue-based edge detection is specialized for color-based edge detection, ideal for segmenting objects by specific colors.
- The choice of method depends on the specific task: hue-based for color, grayscale or intensity-based for structure.

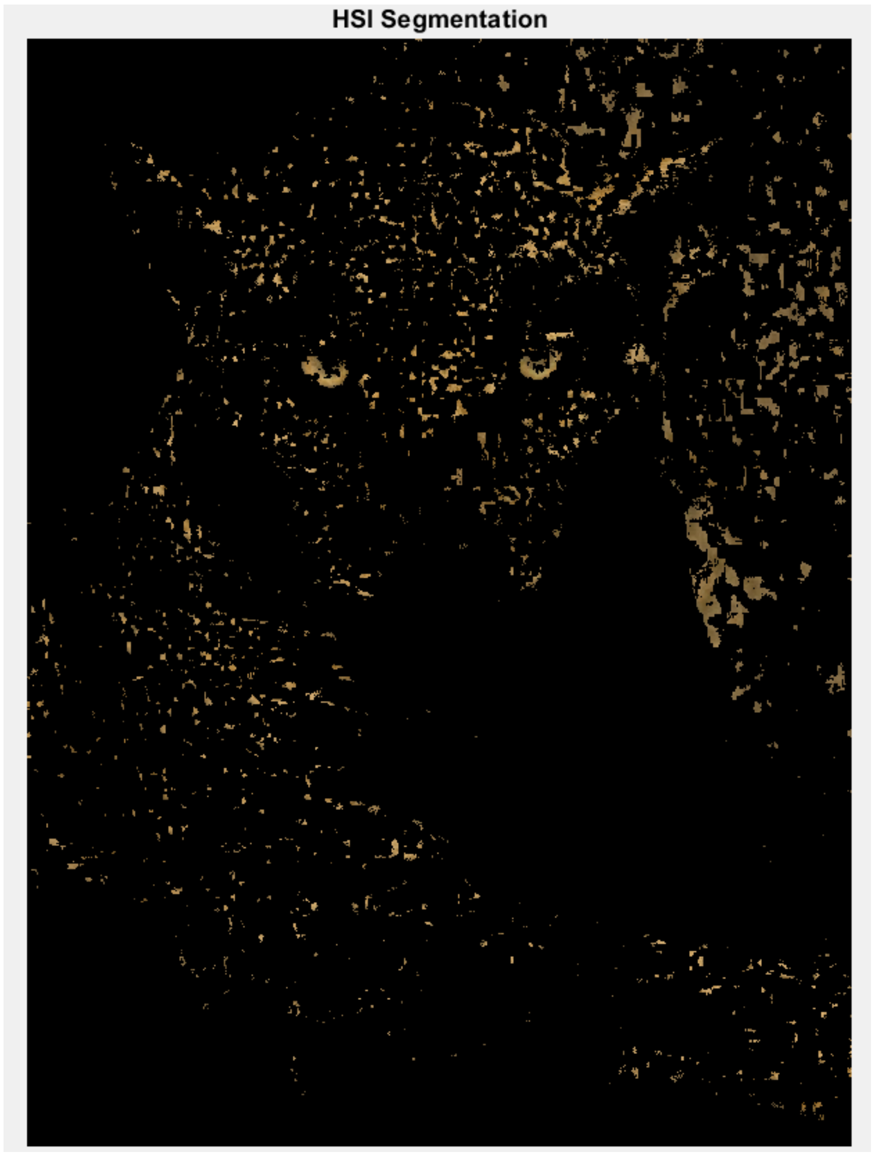
```
% Read the color image

colorImage = imread('img2.jpg');


% Convert the color image to grayscale

grayImage = rgb2gray(colorImage);


% Perform edge detection on the grayscale image

edgesGray = edge(grayImage, 'Canny');


% Convert the color image to HSI manually

r = double(colorImage(:, :, 1)) / 255;
```

```
g = double(colorImage(:, :, 2)) / 255;

b = double(colorImage(:, :, 3)) / 255;


numerator = 0.5 * ((r - g) + (r - b));

denominator = sqrt((r - g).^2 + (r - b) .* (g - b));

theta = acos(numerator ./ (denominator + eps));


% Calculate the hue (H) component

H = theta;

H(b > g) = 2 * pi - H(b > g);

H = H / (2 * pi);


% Calculate the saturation (S) component

S = 1 - 3 * min(min(r, g), b) ./ (r + g + b + eps);


% Calculate the intensity (I) component

I = (r + g + b) / 3;


% Perform edge detection on the intensity component

edgesIntensity = edge(I, 'Canny');


% Perform edge detection on the hue component

edgesHue = edge(H, 'Canny');


% Display the three edge images

subplot(1, 3, 1), imshow(edgesGray), title('Edges on Grayscale');

subplot(1, 3, 2), imshow(edgesIntensity), title('Edges on Intensity');

subplot(1, 3, 3), imshow(edgesHue), title('Edges on Hue');
```

2)

a)

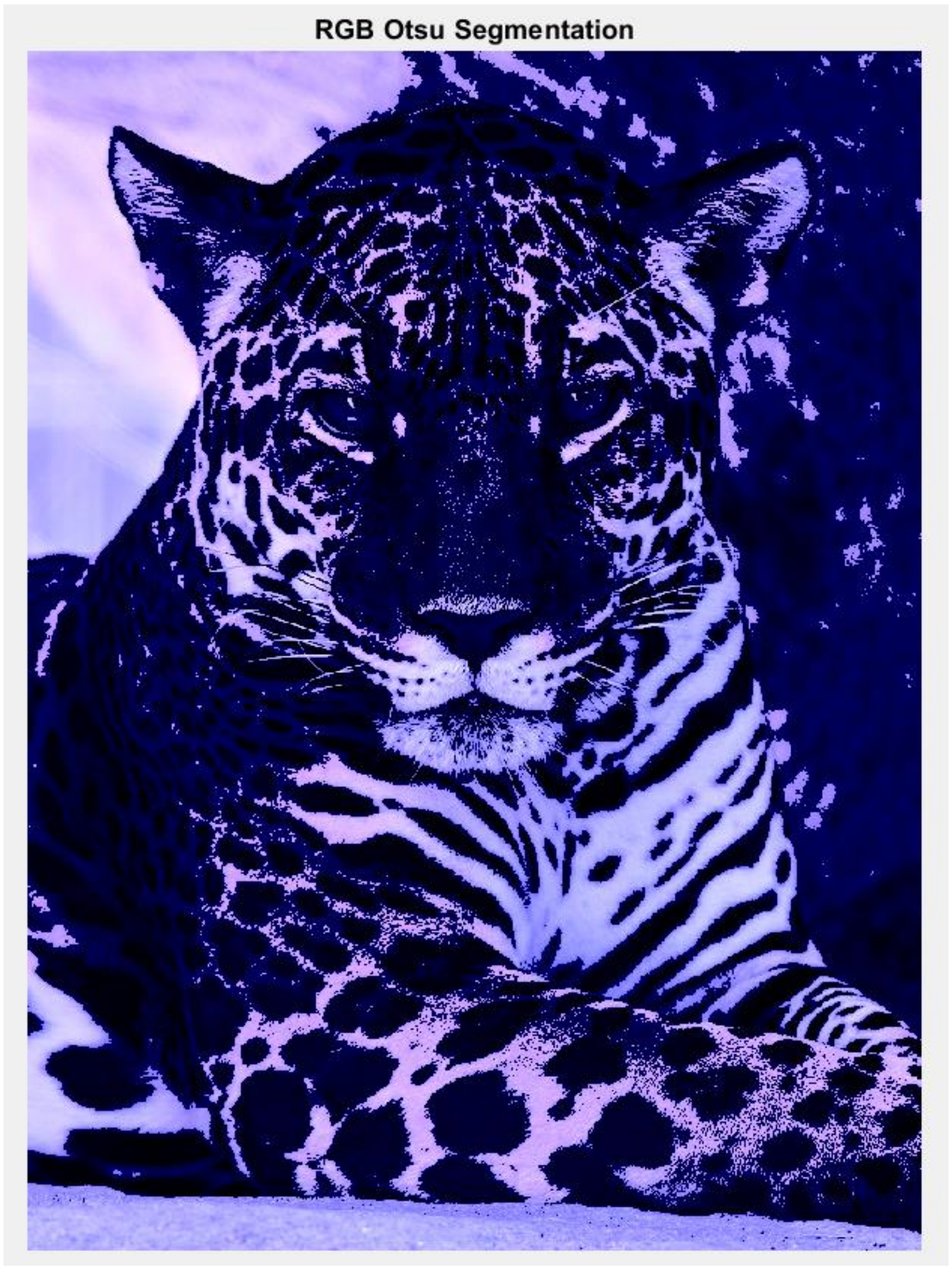


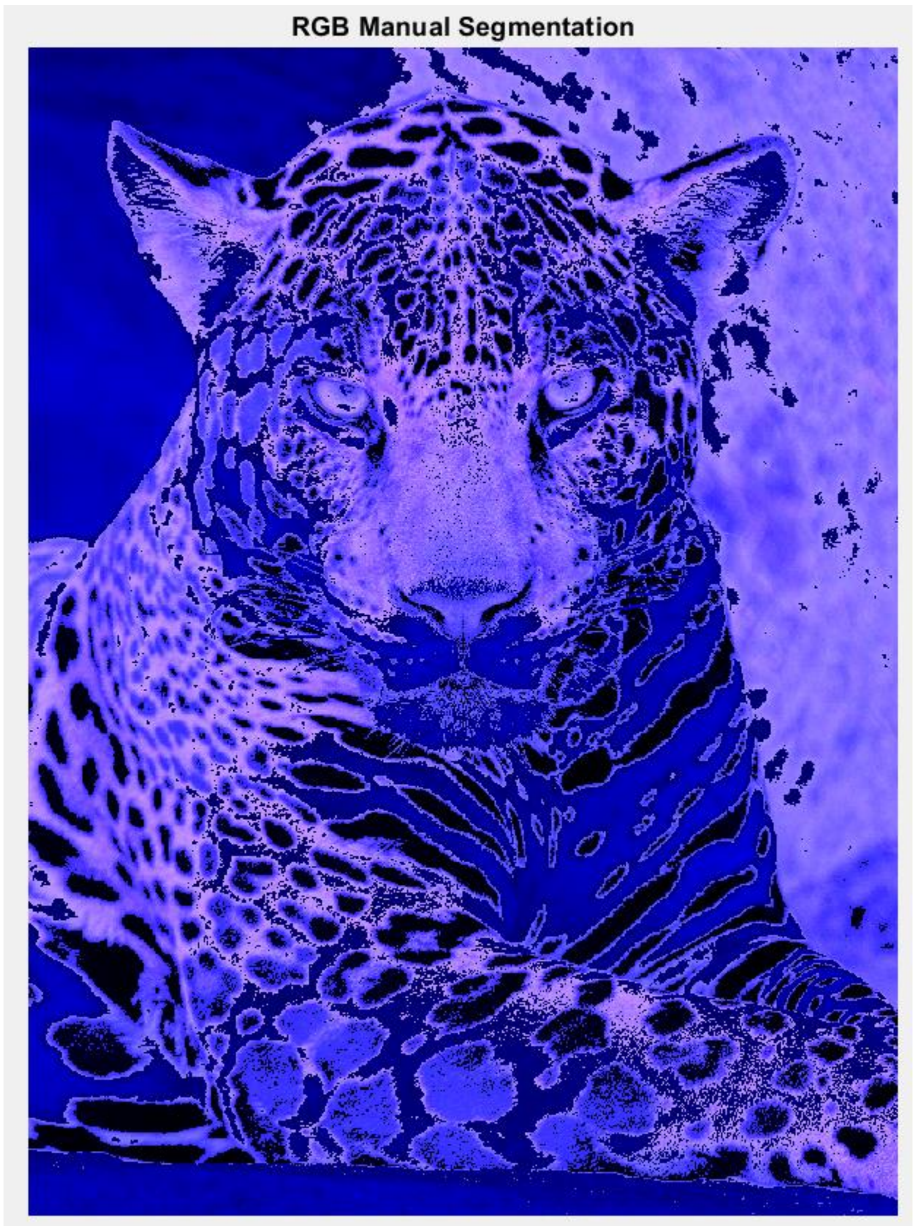


```
% Load the RGB image
rgbImage = imread('img2.jpg');


% Convert the RGB image to HSI color space
hsiImage = rgb2hsv(rgbImage);
```

```
% Extract the H, S, and I channels

hueChannel = hsiImage(:, :, 1);

saturationChannel = hsiImage(:, :, 2);

intensityChannel = hsiImage(:, :, 3);


% Define thresholds for segmentation based on hue, saturation, and intensity

hueThreshold = 0.1;  % Adjust this value based on your image

saturationThreshold = 0.2;  % Adjust this value based on your image

intensityThreshold = 0.5;  % Adjust this value based on your image


% Create a binary mask based on the thresholds

mask = (hueChannel < hueThreshold) & (saturationChannel > saturationThreshold) &
(intensityChannel > intensityThreshold);


% Create a copy of the original image

resultImage = rgbImage;


% Replace background pixels with a visually distinct color (e.g., red)

backgroundReplacementColor = [255, 0, 0];  % Red color


resultImage(repmat(~mask, [1, 1, 3])) = repmat(backgroundReplacementColor, [sum(~mask(:)), 1]);


% Display the segmented image

imshow(resultImage);

title('Segmented Image');
```

b) 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.





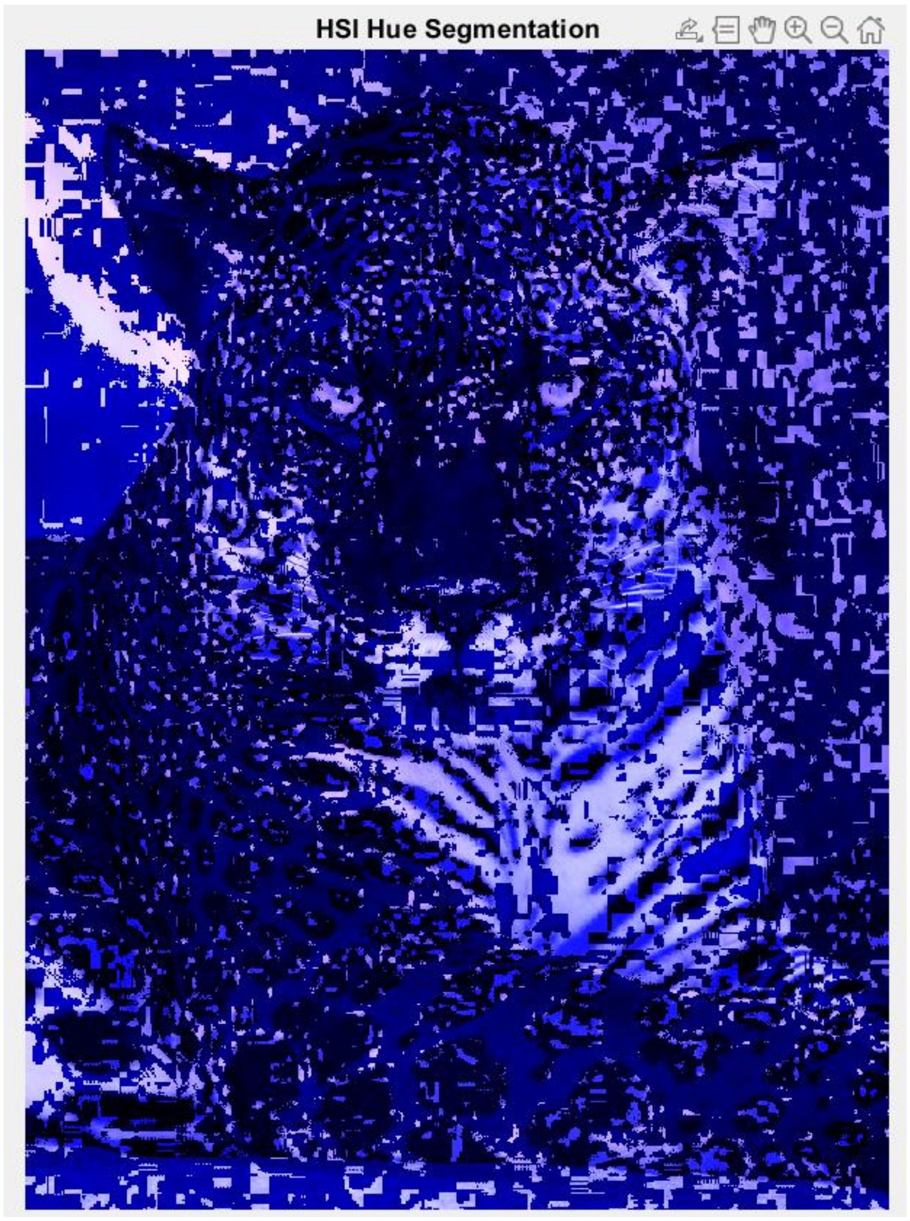
```
% Load the image
originalImage = imread('img2.jpg');
imshow(originalImage);
title('Original Image');

% Convert the image to RGB
rgbImage = originalImage;
```

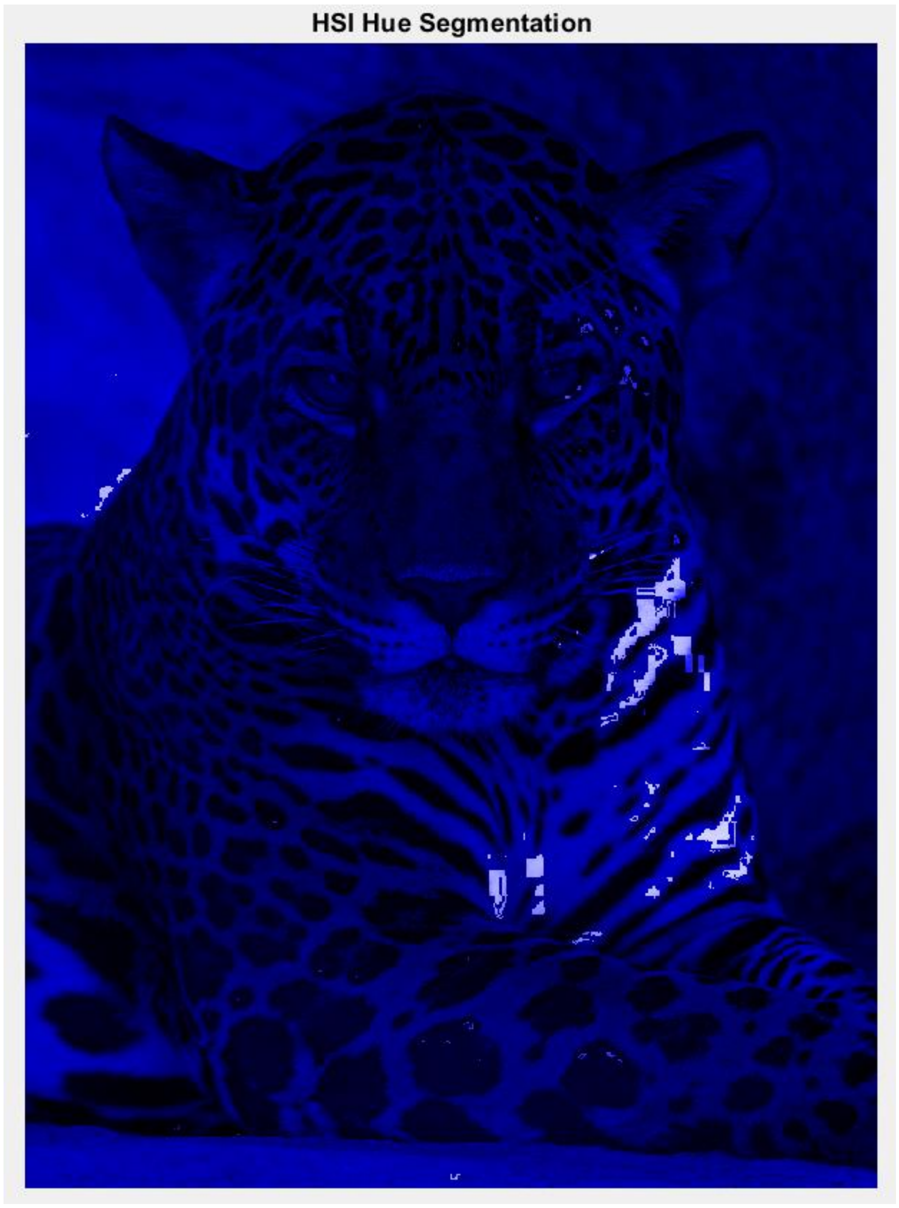
```
% Segment objects using manual thresholding in each RGB color band
% Define manual threshold values for each RGB channel (replace with actual values)
lowerR = 50;
upperR = 255;
lowerG = 0;
upperG = 150;
lowerB = 0;
upperB = 100;

% Create binary masks for each color channel
binaryMaskR = (rgbImage(:, :, 1) >= lowerR) & (rgbImage(:, :, 1) <= upperR);
binaryMaskG = (rgbImage(:, :, 2) >= lowerG) & (rgbImage(:, :, 2) <= upperG);
binaryMaskB = (rgbImage(:, :, 3) >= lowerB) & (rgbImage(:, :, 3) <= upperB);

% Combine the binary masks to obtain the segmented region
binaryMaskRGBManual = binaryMaskR & binaryMaskG & binaryMaskB;

% Show the segmented region in blue
segmentedRGBManual = rgbImage;
segmentedRGBManual(:, :, 1) = segmentedRGBManual(:, :, 1) .* uint8(binaryMaskRGBManual);
segmentedRGBManual(:, :, 2) = segmentedRGBManual(:, :, 2) .* uint8(binaryMaskRGBManual);
segmentedRGBManual(:, :, 3) = segmentedRGBManual(:, :, 3) + 255 * uint8(binaryMaskRGBManual);
figure;
imshow(segmentedRGBManual);
title('RGB Manual Segmentation');

% Convert the image to grayscale for Otsu's thresholding
grayImage = rgb2gray(rgbImage);

% Use Otsu's thresholding for each color channel
thresholdR = graythresh(rgbImage(:, :, 1));
```

```
thresholdG = graythresh(rgbImage(:, :, 2));

thresholdB = graythresh(rgbImage(:, :, 3));


% Create binary masks for each color channel using Otsu's threshold

binaryMaskR = imbinarize(rgbImage(:, :, 1), thresholdR);

binaryMaskG = imbinarize(rgbImage(:, :, 2), thresholdG);

binaryMaskB = imbinarize(rgbImage(:, :, 3), thresholdB);


% Combine the binary masks to obtain the segmented region

binaryMaskRGBOtsu = binaryMaskR & binaryMaskG & binaryMaskB;


% Show the segmented region in blue

segmentedRGBOtsu = rgbImage;

segmentedRGBOtsu(:, :, 1) = segmentedRGBOtsu(:, :, 1) .* uint8(binaryMaskRGBOtsu);

segmentedRGBOtsu(:, :, 2) = segmentedRGBOtsu(:, :, 2) .* uint8(binaryMaskRGBOtsu);

segmentedRGBOtsu(:, :, 3) = segmentedRGBOtsu(:, :, 3) + 255 * uint8(binaryMaskRGBOtsu);

figure;

imshow(segmentedRGBOtsu);

title('RGB Otsu Segmentation');
```

c) 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);
```





```
% Load the image

originalImage = imread('img2.jpg');

imshow(originalImage);

title('Original Image');
```

```
% Convert the image to HSV
hsvImage = rgb2hsv(originalImage);

% Extract the hue channel
hueChannel = hsvImage(:, :, 1);

% Define hue threshold values for blue color
lowerHue = 0.6;  % Lower threshold for blue
upperHue = 0.7;  % Upper threshold for blue

% Create a binary mask based on hue thresholds
binaryMaskHue = (hueChannel >= lowerHue) & (hueChannel <= upperHue);

% Show the segmented region in blue
segmentedHSIHue = originalImage;
segmentedHSIHue(:, :, 1) = segmentedHSIHue(:, :, 1) .* uint8(binaryMaskHue);
segmentedHSIHue(:, :, 2) = segmentedHSIHue(:, :, 2) .* uint8(binaryMaskHue);
segmentedHSIHue(:, :, 3) = segmentedHSIHue(:, :, 3) + 255 * uint8(binaryMaskHue);
figure;
imshow(segmentedHSIHue);
title('HSI Hue Segmentation');
```