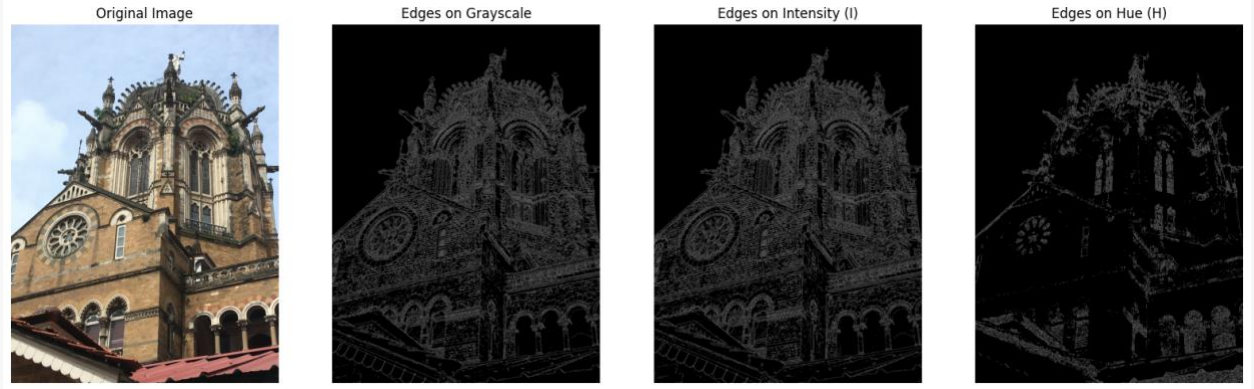
Assignment 5 (45/40 points)

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.

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
# Load Color image
image = cv2.imread('/content/drive/MyDrive/Museum and
Mumbai/IMG_0844.JPG')
image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
# Display Original Image
plt.figure(figsize=(8, 6))
plt.imshow(image_rgb)
plt.title("Original Image")
plt.axis("off")
plt.show()
# Step 1: Convert the image to Grayscale and perform edge detection
# Grayscale conversion (average method)
gray_image = np.mean(image_rgb, axis=2).astype(np.uint8)
# Edge detection on Grayscale image (using Canny method)
edges_gray = cv2.Canny(gray_image, 40, 60)
# Step 2: Convert the image to HSI and extract the Intensity (I) and Hue
(H) channels
# Convert RGB image to HSI (OpenCV doesn't support HSI directly, so we use
HSV as an approximation)
hsi_image = cv2.cvtColor(image_rgb, cv2.COLOR_RGB2HSV)

# Extract the Intensity (V channel in HSV)
intensity_channel = hsi_image[:, :, 2]
# Edge detection on Intensity channel
edges_intensity = cv2.Canny(intensity_channel, 40, 60)
# Extract the Hue (H channel in HSV)
hue_channel = hsi_image[:, :, 0]
# Edge detection on Hue channel
edges_hue = cv2.Canny(hue_channel, 40, 60)
fig, axes = plt.subplots(1, 4, figsize=(20, 10))
axes[0].imshow(image_rgb)
axes[0].set_title("Original Image")
axes[0].axis("off")
```

```
axes[1].imshow(edges_gray, cmap='gray')
axes[1].set_title("Edges on Grayscale")
axes[1].axis("off")

axes[2].imshow(edges_intensity, cmap='gray')
axes[2].set_title("Edges on Intensity (I)")
axes[2].axis("off")

axes[3].imshow(edges_hue, cmap='gray')
axes[3].set_title("Edges on Hue (H)")
axes[3].axis("off")

plt.show()
```



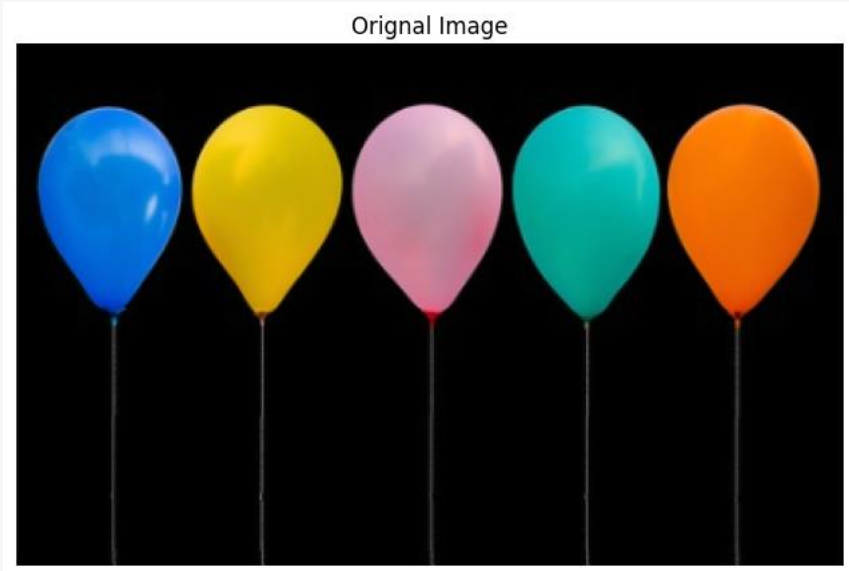
Both the grayscale and intensity images capture edges related to brightness changes and therefore appear very similar. The Intensity (I) component typically produces cleaner edges, as it isolates the brightness independently of color. This can make edge detection slightly more accurate, as it reduces noise caused by chromatic variations. In architectural structures like this building, intensity edges often capture structural details and textural elements more consistently. The Hue (H) component emphasizes edges where there are transitions in color rather than brightness. As a result, some structural details that are prominent in grayscale or intensity edges may appear weaker or are entirely absent.
The Intensity (I) Component generally provides the most relaible and consistent edge results. This is because it isolates brightness, which typically has the most significant role in defining edges in an image. Especially in real worlg images where objects may have varuing shades but similar colors

Problem 2: Color Segmentation

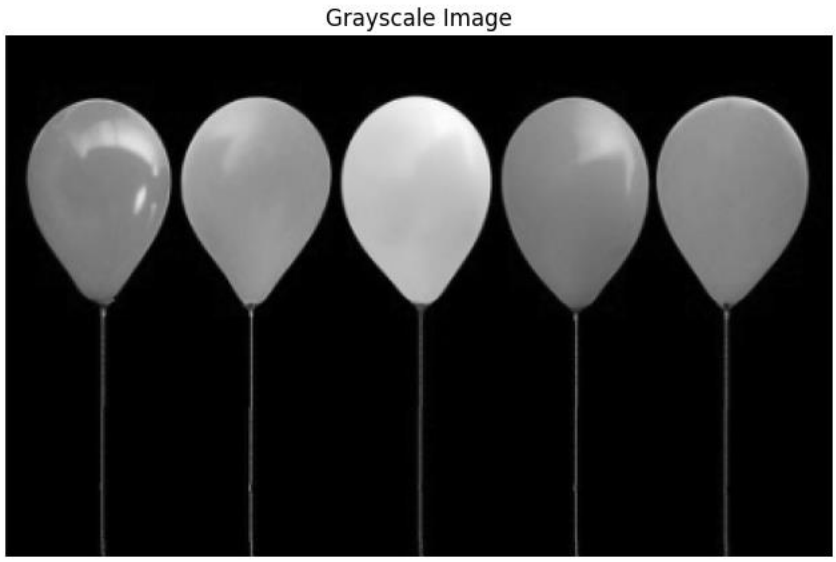
A natural cue to use in segmenting objects from their surroundings in images is color. **This problem contrasts segmenting color regions using red, green, and blue thresholds** (aligned with the RGB axes of color space) **with segmentation using hue, saturation, and intensity bounds** (aligned with the 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).

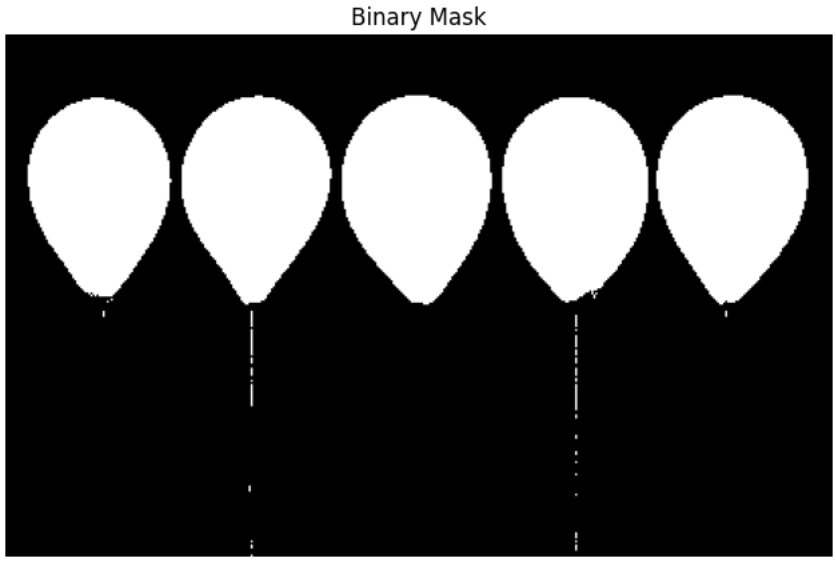
```
import cv2
import numpy as np
import matplotlib.pyplot as plt
image_path = '/content/drive/MyDrive/Introduction to Image
Processing/Assignment 5/segmentation.jpg'
image = cv2.imread(image_path)
image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
plt.figure(figsize=(8, 6))
plt.imshow(image_rgb)
plt.title("Orignal Image")
plt.axis("off")
plt.show()
```




```
gray_image = np.mean(image_rgb, axis=2).astype(np.uint8)
plt.figure(figsize=(8, 6))
plt.imshow(gray_image, cmap='gray')
plt.title("Grayscale Image")
plt.axis("off")
plt.show()
```

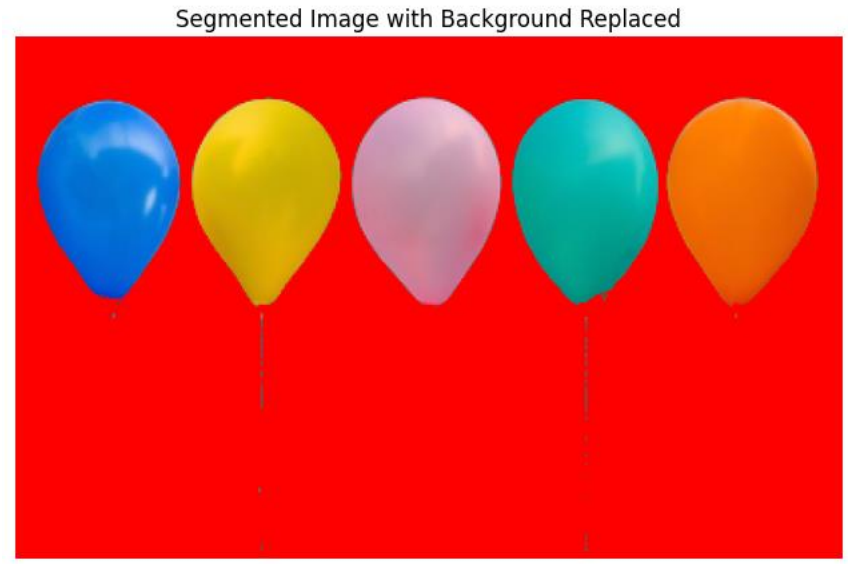



```
# Step 2: apply Intensity thresholding
threshold_value = 85
_, binary_mask = cv2.threshold(gray_image, threshold_value, 255,
cv2.THRESH_BINARY)

# Display Binary Mask
plt.figure(figsize=(8, 6))
plt.imshow(binary_mask, cmap='gray')
plt.title("Binary Mask")
plt.axis("off")
plt.show()
```



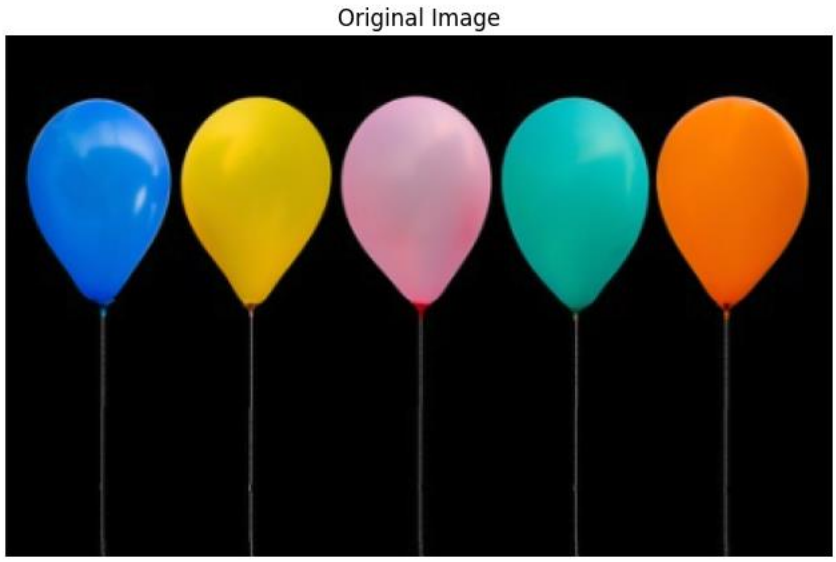

```
# Step 3: Replace backgroud pixels with Distinct Color
```

```
background_color = [255,0,0]
segmented_image = image_rgb.copy()
segmented_image[binary_mask == 0] = background_color
plt.figure(figsize=(8,6))
plt.imshow(segmented_image)
plt.title("Segmented Image with Background Replaced")
plt.axis("off")
plt.show()
```



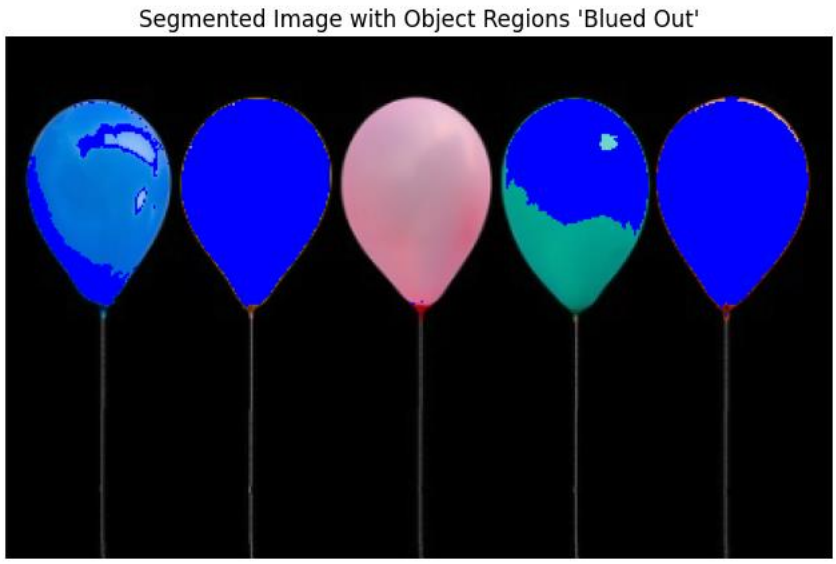
b) **(10/10)** Second, use thresholds in each RGB color band to isolate the objects in your image. Again, display the results by "bluing out" the intended object region. Provide some commentary on how the segmentation succeeded and failed. (**481 Students (5)**: use an automatic thresholding approach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

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
image = cv2.imread('/content/drive/MyDrive/Introduction to Image
Processing/Assignment 5/segmentation.jpg')
image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
plt.figure(figsize=(8, 6))
plt.imshow(image_rgb)
plt.title("Original Image")
plt.axis("off")
```




```
blue_lower = np.array([0, 0, 150])
blue_upper = np.array([100, 100, 255])

yellow_lower = np.array([150, 150, 0])
yellow_upper = np.array([255, 255, 100])

pink_lower = np.array([150, 0, 150])
pink_upper = np.array([255, 100, 255])

green_lower = np.array([0, 150, 150])
green_upper = np.array([100, 255, 255])

orange_lower = np.array([150, 50, 0])
orange_upper = np.array([255, 150, 100])
blue_mask = cv2.inRange(image_rgb, blue_lower, blue_upper)
yellow_mask = cv2.inRange(image_rgb, yellow_lower, yellow_upper)
pink_mask = cv2.inRange(image_rgb, pink_lower, pink_upper)
green_mask = cv2.inRange(image_rgb, green_lower, green_upper)
orange_mask = cv2.inRange(image_rgb, orange_lower, orange_upper)
combined_mask = blue_mask | yellow_mask | pink_mask | green_mask |
orange_mask
blue_color = [0, 0, 255]
segmented_image = image_rgb.copy()
segmented_image[combined_mask == 255] = blue_color

# Display the final segmented image
plt.figure(figsize=(8, 6))
plt.imshow(segmented_image)
plt.title("Segmented Image with Object Regions 'Blued Out'")
```

```
plt.axis("off")
plt.show()
```

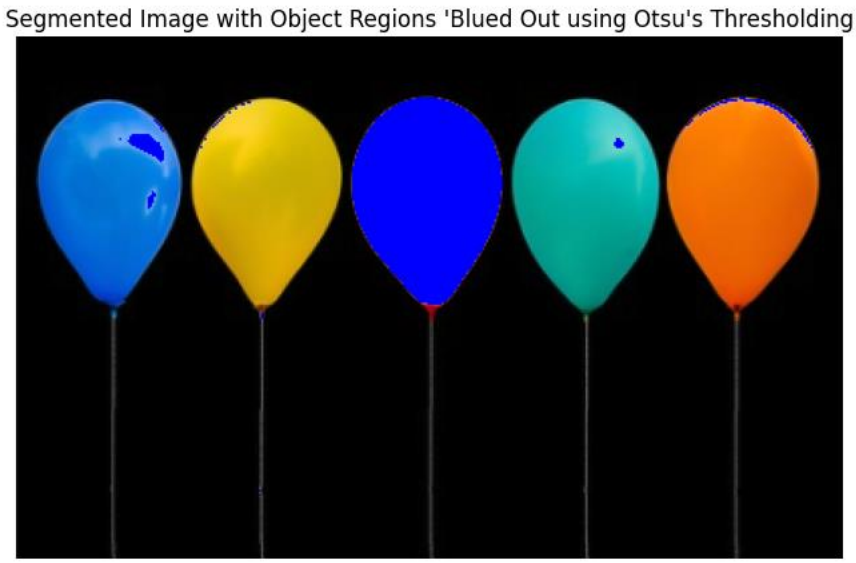


1. Successes:
The RGB thresholding approach sucessfully isolates and 'blued out' regions of each balloon that closely matched the predefined RGB ranges.
Balloons with more distinct colors (such as the blue and yellow) are particularly well segmented, as their colors are less likely to overlap with the background.

2. Failures:
Some Balloons (particularly pink and green) were only partially segmented due to variations in shading or slight overlaps in the RGB threshold values.
The Segmentation may have missed parts of certain balloons, especially if the thresholds do not perfectly cover the color range within each balloon.

```
import cv2
import numpy as np
import matplotlib.pyplot as plt

image_path = '/content/drive/MyDrive/Introduction to Image
Processing/Assignment 5/segmentation.jpg'
image = cv2.imread(image_path)
image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
r_channel, g_channel, b_channel = cv2.split(image_rgb)
# Apply Otsu's thresholding on each channel
```

```
_, r_thresh = cv2.threshold(r_channel, 0, 255, cv2.THRESH_BINARY +
cv2.THRESH_OTSU)
_, g_thresh = cv2.threshold(g_channel, 0, 255, cv2.THRESH_BINARY +
cv2.THRESH_OTSU)
_, b_thresh = cv2.threshold(b_channel, 0, 255, cv2.THRESH_BINARY +
cv2.THRESH_OTSU)
combined_mask = r_thresh & g_thresh & b_thresh
blue_color = [0, 0, 255]
segmented_image = image_rgb.copy()
segmented_image[combined_mask == 255] = blue_color

# Display the final segmented image
plt.figure(figsize=(8, 6))
plt.imshow(segmented_image)
plt.title("Segmented Image with Object Regions 'Blued Out using Otsu's
Thresholding")
plt.axis("off")
plt.show()
```



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);
```

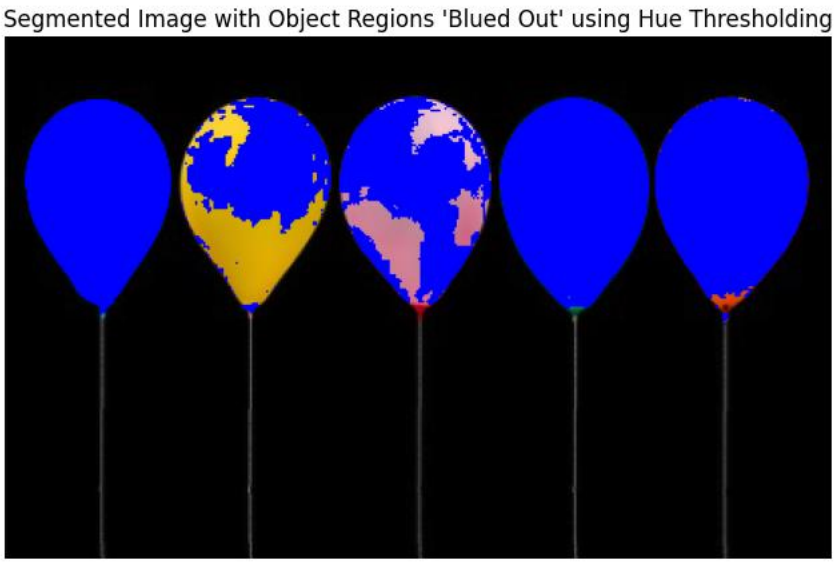
It returns an RGB image with pixel components between 0.0 and 1.0

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?

```
# Convert RGB image to HSV
hsv_image = cv2.cvtColor(image_rgb, cv2.COLOR_RGB2HSV)

# Define hue ranges for each color (normalized between 0 and 180 in
OpenCV's HSV space)
# OpenCV hue values range from 0 to 180, so we adjust our ranges
accordingly.
blue_lower = np.array([100, 50, 50])   # Approximate blue range
blue_upper = np.array([130, 255, 255])

yellow_lower = np.array([25, 50, 50])  # Approximate yellow range
yellow_upper = np.array([35, 255, 255])

pink_lower = np.array([150, 50, 50])   # Approximate pink (or magenta)
range
pink_upper = np.array([170, 255, 255])

green_lower = np.array([85, 50, 50])   # Approximate green range
green_upper = np.array([95, 255, 255])

orange_lower = np.array([10, 50, 50])  # Approximate orange range
orange_upper = np.array([20, 255, 255])

# Create masks for each color range
blue_mask = cv2.inRange(hsv_image, blue_lower, blue_upper)
yellow_mask = cv2.inRange(hsv_image, yellow_lower, yellow_upper)
pink_mask = cv2.inRange(hsv_image, pink_lower, pink_upper)
green_mask = cv2.inRange(hsv_image, green_lower, green_upper)
orange_mask = cv2.inRange(hsv_image, orange_lower, orange_upper)

# Combine all color masks to isolate all balloons
combined_mask_hue = blue_mask | yellow_mask | pink_mask | green_mask |
orange_mask
```

```
# Define blue color for "bluing out" the object regions
blue_color = [0, 0, 255]  # Blue in RGB

# Apply the blue color to the object regions based on the combined mask
segmented_image_hue = image_rgb.copy()
segmented_image_hue[combined_mask_hue == 255] = blue_color

# Display the final segmented image
plt.figure(figsize=(8, 6))
plt.imshow(segmented_image_hue)
plt.title("Segmented Image with Object Regions 'Blued Out' using Hue
Thresholding")
plt.axis("off")
plt.show()
```



Hue Based Segmentation: Since the hue seperated color from intensity, it is often more effective in isolating color regions in image with variable lightitng or brightness. This method is particularly useful in scenarios with colored objects on a dark or light background, as it ignores intensity variations and focuses purely on color. RGB-Based Segmentation: RGB Based Segmentation relies heavily on intensity, so regions with similar RGB values but different intensities may not be accurately segmented. RGB channels are also more susceptible to noise, which can make it challenging to isolate colors with consistency.

Advantages: Hue Segmentation is more consistent under differnt lightitng conditions, as it focuses only on color aspect. Hue segmentation allows for precise targeting of colors which is particularly useful for isolating specific objects based on color alone.

Limitations: Overlap in Hue Ranges and Complex Color Variations.