

---

## **TUTORIAL 3**

---

### **Color Image Processing**

Pengolahan Citra - Semester Gasal 2023/2024

# 1 Color Transformation

*Color Transformation* merupakan proses transformasi citra dari ruang warna tertentu ke ruang warna lainnya.

## 1.1 RGB Color Space

$\text{RGB} = \text{Red Green Blue}$

```
1 image = io.imread('./cars.jpg')
2 R = image[:, :, 0]
3 G = image[:, :, 1]
4 B = image[:, :, 2]
5
6 plt.figure(figsize=(10, 7))
7 plt.subplot(2, 2, 1); plt.imshow(image);
8 plt.title('Original RGB'); plt.axis('off')
9 plt.subplot(2, 2, 2); plt.imshow(R, cmap='gray', vmin=0, vmax=255);
10 plt.title('Red channel'); plt.axis('off')
11 plt.subplot(2, 2, 3); plt.imshow(G, cmap='gray', vmin=0, vmax=255);
12 plt.title('Green channel'); plt.axis('off')
13 plt.subplot(2, 2, 4); plt.imshow(B, cmap='gray', vmin=0, vmax=255);
14 plt.title('Blue channel'); plt.axis('off')
15 plt.show()
```



Figure 1: RGB Channel

## 1.2 CMY Color Space

$\text{CMY} = \text{Cyan Magenta Yellow}$

```
1 C = 1 - util.img_as_float(R)
2 M = 1 - util.img_as_float(G)
3 Y = 1 - util.img_as_float(B)
4
5 CMY = np.zeros(image.shape)
6 CMY[:, :, 0] = C
7 CMY[:, :, 1] = M
8 CMY[:, :, 2] = Y
9
10 plt.figure(figsize=(10, 7))
11 plt.subplot(2, 2, 1); plt.imshow(CMY);
12 plt.title('CMY'); plt.axis('off')
13 plt.subplot(2, 2, 2); plt.imshow(C, cmap='gray');
14 plt.title('Cyan channel'); plt.axis('off')
15 plt.subplot(2, 2, 3); plt.imshow(M, cmap='gray');
16 plt.title('Magenta channel'); plt.axis('off')
17 plt.subplot(2, 2, 4); plt.imshow(Y, cmap='gray');
18 plt.title('Yellow channel'); plt.axis('off')
19 plt.show()
```

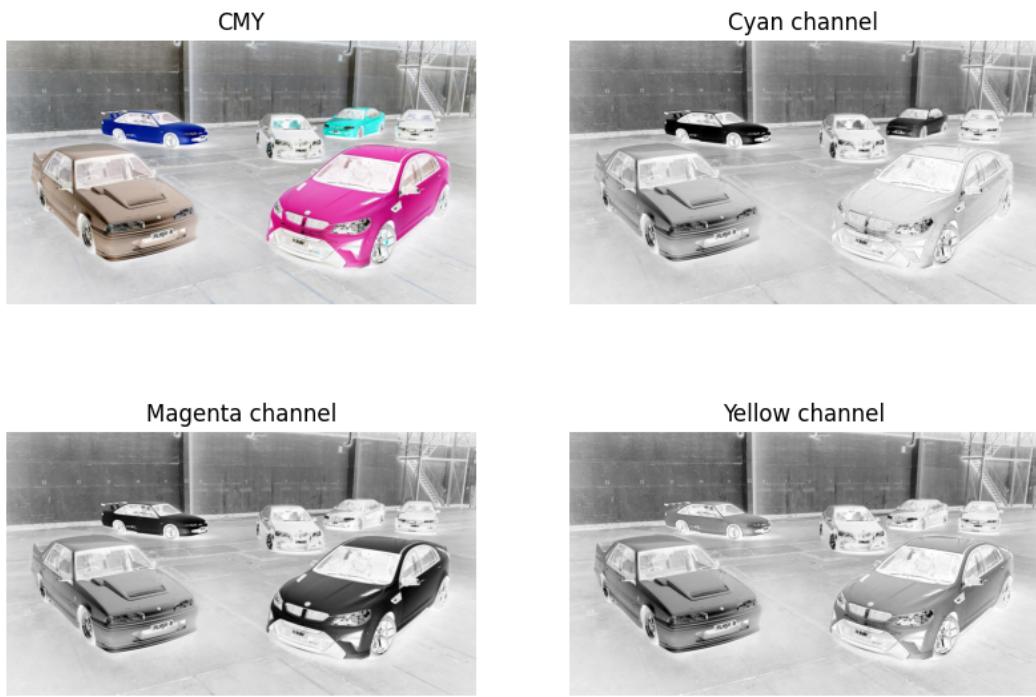


Figure 2: CMY Channel

### 1.3 HSV Color Space

$\text{HSV} = \text{Hue Saturation Value}$

```
1 HSV = color.rgb2HSV(image)
2 H = HSV[:, :, 0]
3 S = HSV[:, :, 1]
4 V = HSV[:, :, 2]
5
6 plt.figure(figsize=(10, 7))
7 plt.subplot(2, 2, 1); plt.imshow(HSV);
8 plt.title('HSV'); plt.axis('off')
9 plt.subplot(2, 2, 2); plt.imshow(H, cmap='gray');
10 plt.title('Hue channel'); plt.axis('off')
11 plt.subplot(2, 2, 3); plt.imshow(S, cmap='gray');
12 plt.title('Saturation channel'); plt.axis('off')
13 plt.subplot(2, 2, 4); plt.imshow(V, cmap='gray');
14 plt.title('Value channel'); plt.axis('off')
15 plt.show()
```

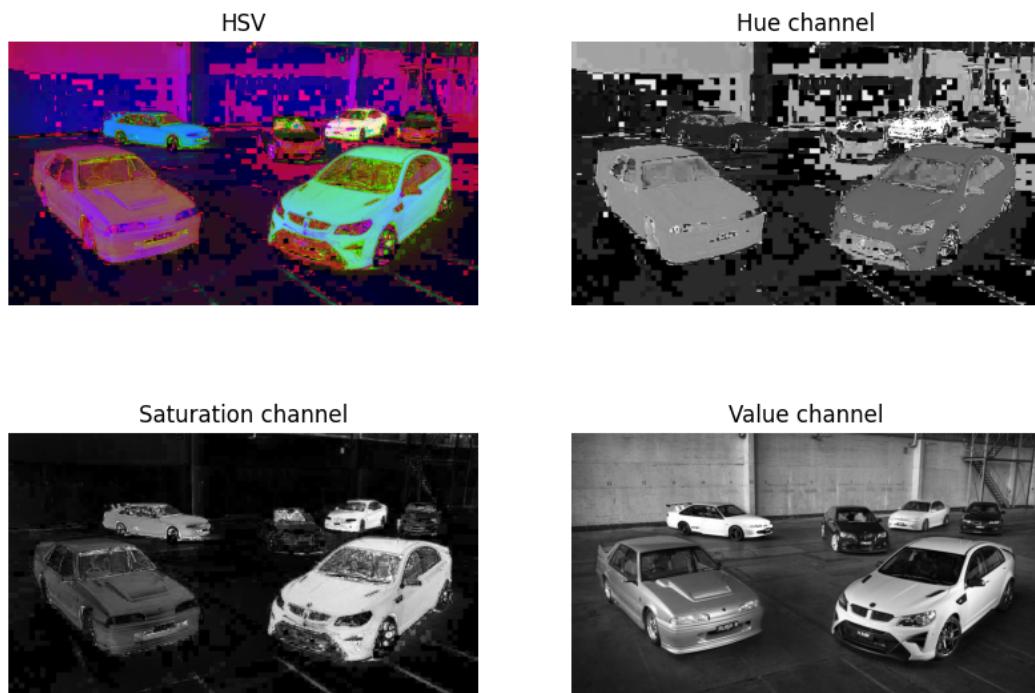


Figure 3: HSV Channel

## 2 Color Histogram Processing

### 2.1 Contrast Stretching

Seperi yang telah diajarkan pada tutorial 1 (Note : Harap membaca ulang tutorial 1 untuk penjelasan lebih rinci), histogram merupakan suatu teknik yang mampu memberikan deskripsi umum pada tampilan citra. Pada bagian color contrast stretching dan histogram equalization dapat digunakan untuk masing-masing channel R, G, B, dan seterusnya.



Figure 4: Contrast Stretching

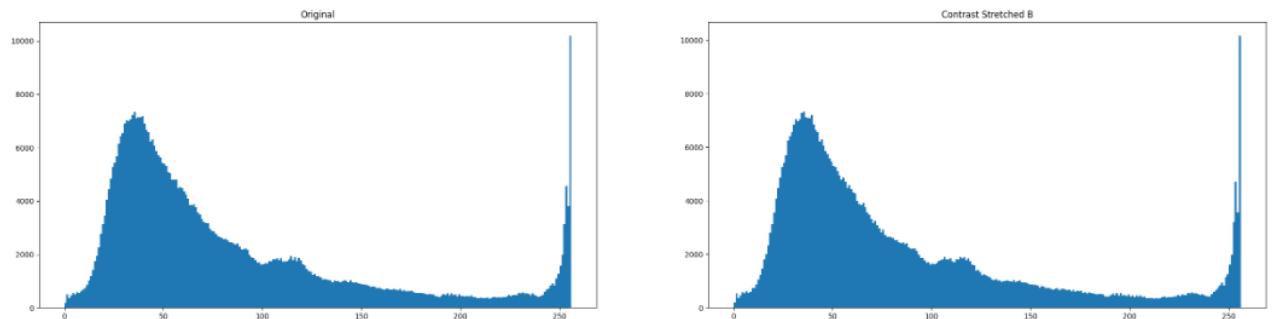


Figure 5: Contrast Stretching

Gambar 4 merupakan salah satu contoh *contrast stretching* pada *channel B*. Perbedaan kurang tampak dikarenakan dominansi warna pada gambar adalah hijau, sedangkan yang dilakukan *stretching* adalah *channel B*.

### 2.2 HSV Color Space

HSV = Hue Saturation Value

```
1 import skimage.exposure as exp  
2
```

```

3  image = io.imread('naik_daun.jpg')
4  stretched_image_b = image.copy()
5
6  B = image[:, :, 2]
7
8  # In_Range dapat diatur tergantung seberapa contrast warna
   pada gambar yang kita inginkan
9  stretched_image_b[:, :, 2] = exp.rescale_intensity(B,
   in_range=(np.percentile(B, 2), np.percentile(B, 98)))

```

### 2.3 Histogram Equalization

Seperti yang telah dijelaskan pada tutorial 1, histogram equalization adalah teknik untuk membuat pemetaan piksel pada citra lebih menyebar pada kisaran 0-255.



Figure 6: Histogram Equalization

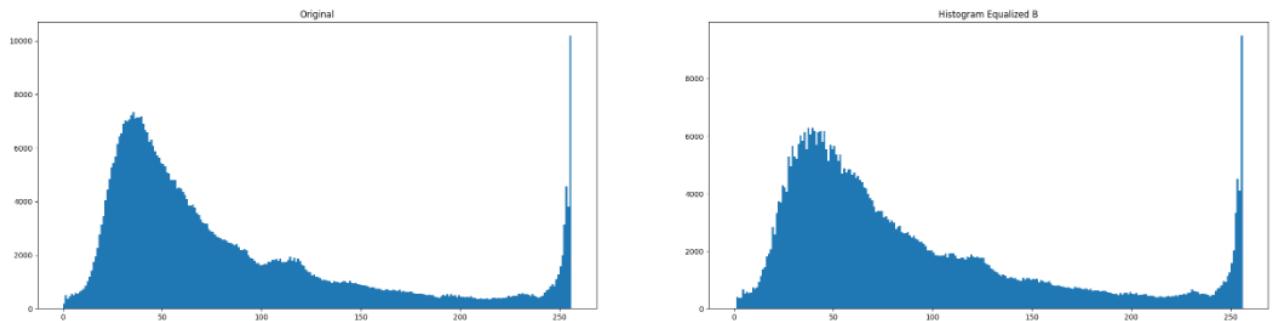


Figure 7: Histogram Equalization

Pada gambar 7 dapat dilihat bahwa dibandingkan dengan *contrast stretching*, menggunakan *histogram equalization* memiliki histogram yang cenderung lebih diratakan persebarannya.

```

1 # Extract the Blue (B) channel
2 B = image[:, :, 2]
3
4 # Apply histogram equalization to the Blue channel
5 equalized_B = exp.equalize_hist(B)
6
7 # Create a copy of the original image
8 equalized_image = image.copy()
9
10 # Replace the Blue channel with the equalized Blue channel
11 equalized_image[:, :, 2] = (equalized_B * 255).astype('uint8')
12
13
14 # Display or save the result
15 plt.figure(figsize=(15,15))
16 plt.subplot(1,2,1)
17 plt.imshow(image); plt.title('Original Naik Daun')
18 plt.subplot(1,2,2)
19 plt.imshow(equalized_image); plt.title('Naik Daun with
    Histogram Equalization B Channel')
20 plt.show()

```

## 2.4 Catatan tentang pemrosesan histogram

Ruang warna RGB merupakan ruang warna paling umum yang biasanya digunakan dalam representasi citra berwarna, oleh karena itu, banyak teknik pengolahan citra berwarna dilakukan pada setiap channel seperti contoh di atas. Namun sesungguhnya dapat pula dilakukan dengan melakukan transformasi warna ke ruang warna yang memiliki channel yang berkorelasi dengan brightness, seperti HSV atau La\*b\* dan melakukan pemrosesan histogram pada channel L atau H saja. Kedua metode dapat digunakan, namun akan menimbulkan perbedaan hasil.

## 3 Smoothing & Sharpening

### 3.1 Smoothing

Smoothing adalah proses untuk menghilangkan noise spasial dari citra.

```

1 from skimage import filters, morphology
2
3 image = io.imread('smurfen.jpg')
4 R = image[:, :, 0]
5 G = image[:, :, 1]
6 B = image[:, :, 2]
7

```

```

8  B2 = filters.rank.mean(B, selem=morphology.square(9))
9  RGB = util.img_as_ubyte(np.zeros(image.shape))
10 RGB[:, :, 0] = R
11 RGB[:, :, 1] = G
12 RGB[:, :, 2] = B2
13
14 plt.figure(figsize=(15, 8))
15 plt.subplot(1, 2, 1); plt.imshow(image, cmap='gray')
16 plt.title("Original")
17 plt.subplot(1, 2, 2); plt.imshow(RGB, cmap='gray')
18 plt.title("Image with smoothed B component")

```

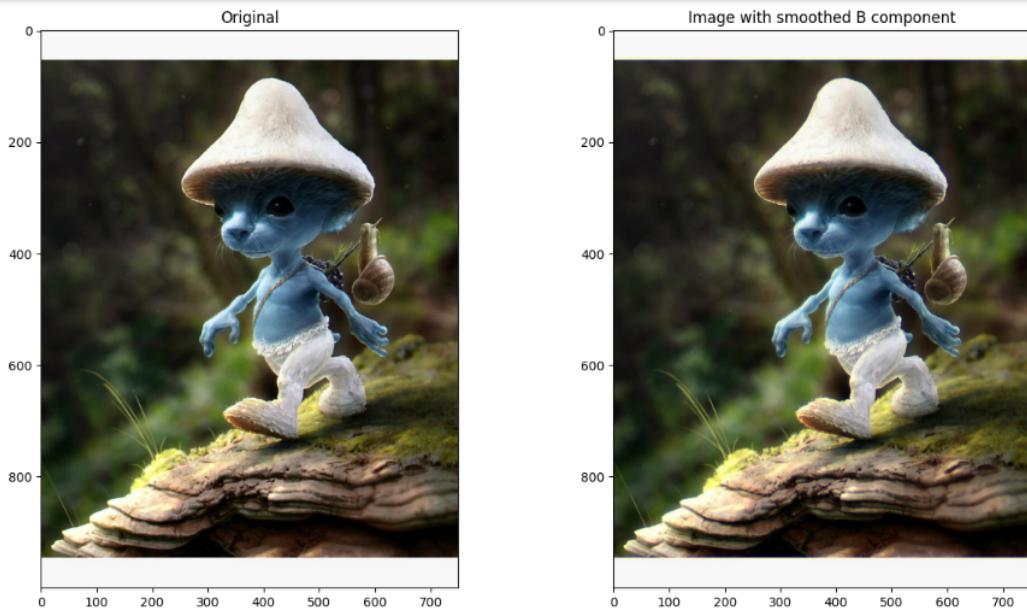


Figure 8: Smoothing B Channel Gambar Smurfen

### 3.2 Sharpening

Sharpening adalah proses untuk meningkatkan kontras dari edges pada citra.

```

1  B2 = util.img_as_ubyte(filters.unsharp_mask(B, radius=5,
2      amount=2))
3  RGB2 = util.img_as_ubyte(np.zeros(image.shape))
4  RGB2[:, :, 0] = R
5  RGB2[:, :, 1] = G
6  RGB2[:, :, 2] = B2
7
8  plt.figure(figsize=(15, 8))
9  plt.subplot(1, 2, 1); plt.imshow(image)
10 plt.title("Original")

```

```

10 plt.subplot(1,2,2); plt.imshow(RGB2)
11 plt.title("Image with sharpened B component")

```

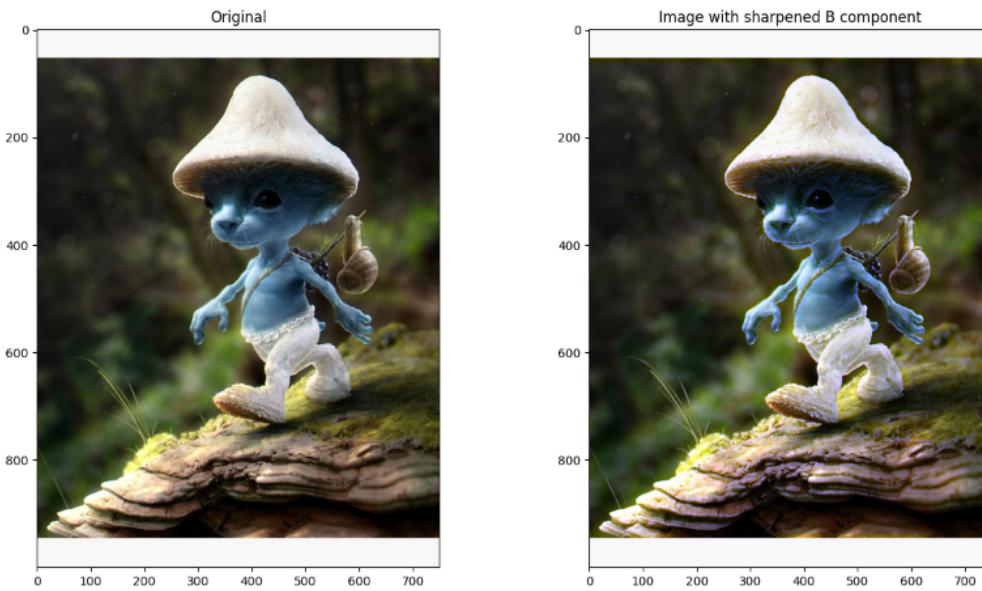


Figure 9: Sharpening B Channel Gambar Smurfen

## 4 Color Segmentation

Color segmentation digunakan untuk mempartisi objek tertentu dari suatu citra menjadi beberapa region berdasarkan warnanya. Tujuan dari color segmentation adalah untuk mendapatkan representasi citra yang lebih bermakna. Metode color segmentation menggunakan distance dan threshold. Terdapat berbagai jenis distance measure yang dapat digunakan, begitu pula dengan color space yang digunakan. Pemilihan semua parameter tsb akan mempengaruhi hasil akhir yang diperoleh.

### 4.1 Distance (K-Means)

Metode K-Means mempartisi N piksel ke dalam K cluster. Cluster sendiri merupakan dapat dianggap sebagai himpunan yang terdiri dari piksel yang memiliki similaritas (warna) yang mirip.

Algoritma K-Means:

1. Pilih k centroid cluster secara acak.
2. Untuk setiap piksel citra bandingkan distance piksel dengan centroid masing - masing cluster.
3. Assign piksel ke dalam cluster dengan nilai distance piksel-centroid yang paling kecil.
4. Update nilai centroid cluster yang baru berdasarkan nilai piksel dalam masing - masing cluster (umumnya nilai centroid yang baru didapat dengan mencari nilai rata - rata piksel ).

5. Bila nilai centroid yang baru maka algoritma telah selesai. Jika nilai centroid tidak sama, ulangi langkah 2 sampai 5.

Pada beberapa kasus kondisi nilai centroid dapat diberi nilai toleransi tertentu untuk menghentikan iterasi.

```

1 import cv2
2 import numpy as np
3 import matplotlib.pyplot as plt
4
5 sidewalk_img = cv2.imread("side_walk.jpg")
6 sidewalk_img = cv2.cvtColor(sidewalk_img, cv2.COLOR_BGR2RGB)
7
8 pixel_values = sidewalk_img.reshape((-1, 3))
9 pixel_values = np.float32(pixel_values)
10
11 criteria = (cv2.TERM_CRITERIA_EPS +
12             cv2.TERM_CRITERIA_MAX_ITER, 100, 0.2)
13
14 k = 10
15 _, labels, centers = cv2.kmeans(pixel_values, k, None,
16                                   criteria, 10, cv2.KMEANS_RANDOM_CENTERS)
17
18 centers = np.uint8(centers)
19 labels = labels.flatten()
20
21 segmented_image = centers[labels.flatten()]
22 segmented_image = segmented_image.reshape(sidewalk_img.shape)
23
24 plt.figure(figsize=(16, 8))
25 plt.subplot(1, 2, 1); plt.imshow(sidewalk_img)
26 plt.title("Original")
27 plt.subplot(1, 2, 2); plt.imshow(segmented_image)
28 plt.title("Color segmentation with k=10")
29 plt.show()

```

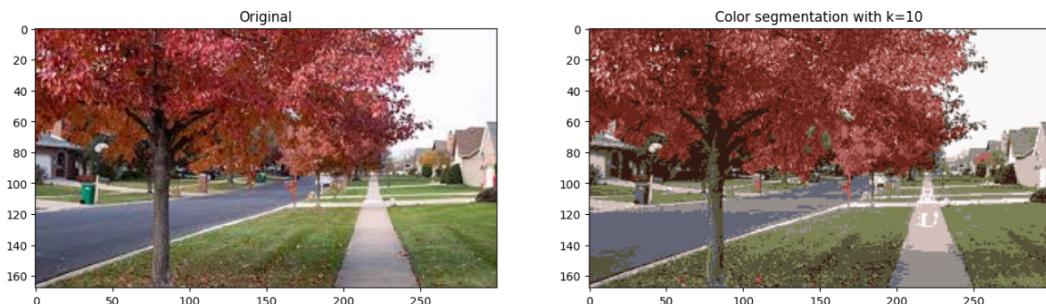


Figure 10: Color Segmentation menggunakan K-Means untuk k=10

Pemilihan nilai  $k$  tentunya sesuai kebutuhan dan bergantung pada data citra yang sedang digunakan. Umumnya, semakin banyak variasi warna yang terdapat pada citra, akan memiliki nilai  $k$  yang lebih besar. Pada gambar 10, terlihat bahwa jalan utama terdiri dari 2 warna yang berbeda. Jika dipilih nilai  $k$  yang cukup kecil, kemungkinan besar jalan tersebut hanya akan memiliki 1 warna saja nantinya.

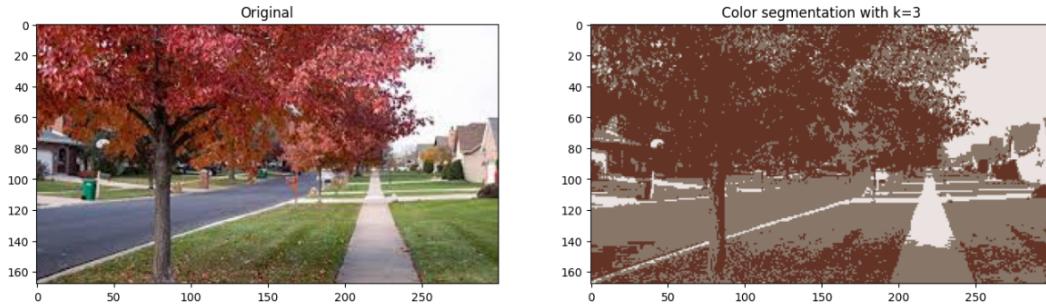


Figure 11: Color Segmentation menggunakan K-Means untuk  $k=3$

Sebagai contoh pada gambar 11, yang mana hanya terdapat 3 warna. Pada citra tersebut, warna coklat yang cenderung dominan pada citra asli menjadi salah satu warna yang muncul. Selain itu, jalan utama menjadi sulit dideteksi karena sebagian tampilannya memiliki warna yang sama dengan rumput.

## 4.2 Threshold

Metode threshold mengambil piksel pada citra yang berada pada range tertentu. Umumnya proses color segmentation lebih mudah dilakukan dalam representasi HSV. Hal ini dikarenakan channel hue memodelkan warna dari citra sehingga mempermudah proses segmentasi.

```

1  hsv = cv2.cvtColor(sidewalk_img, cv2.COLOR_RGB2HSV)
2  low_green = (30, 0, 0)
3  high_green = (86, 255, 255)
4  mask = cv2.inRange(hsv, low_green, high_green)
5  result = cv2.bitwise_and(sidewalk_img, sidewalk_img, mask=mask)

6
7  plt.figure(figsize=(24, 8))
8  plt.subplot(1,3,1); plt.imshow(sidewalk_img)
9  plt.title("Original")
10 plt.subplot(1,3,2); plt.imshow(mask)
11 plt.title("Mask")
12 plt.subplot(1,3,3); plt.imshow(result)
13 plt.title("Result")
14 plt.show()

```

Pada contoh ini, ditunjukkan bagaimana warna hijau pada gambar *sidewalk* dideteksi. Pendekripsi warna hijau berdasarkan hue, dengan threshold diantara 30 an 86. Sehingga pada citra result, didapatkan bagian citra yang berwarna hijau saja.

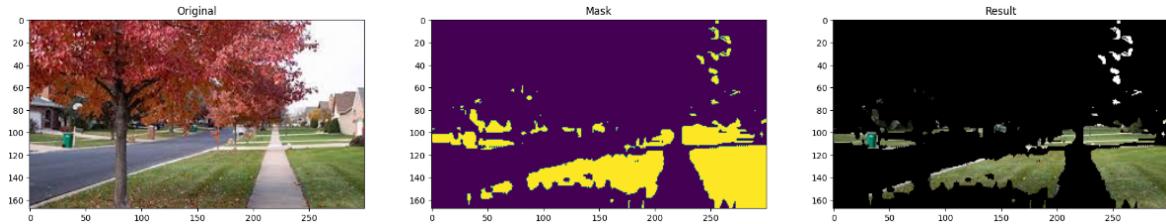


Figure 12: Color Segmentation menggunakan threshold

Berikut ini adalah contoh lain bagaimana untuk mendeteksi suatu warna yang ada dalam sebuah citra "bags.jpg".



Figure 13: Citra bags.jpg

```

1  from skimage.io import imread, imshow
2  from skimage.color import rgb2hsv
3  import matplotlib.pyplot as plt
4
5  bags = io.imread('bags.jpg')
6
7  bags_hsv = rgb2hsv(bags)
8  fig, ax = plt.subplots(1, 3, figsize=(12,4))
9  ax[0].imshow(bags_hsv[:, :, 0], cmap='gray')
10 ax[0].set_title('Hue')
11 ax[1].imshow(bags_hsv[:, :, 1], cmap='gray')
12 ax[1].set_title('Saturation')
13 ax[2].imshow(bags_hsv[:, :, 2], cmap='gray')
14 ax[2].set_title('Value');
```

Untuk mempermudah color segmentation, citra HSV perlu diberikan colorbar. Colorbar sendiri akan berguna untuk mengetahui nilai-nilai warna pada sebuah citra yang ditampilkan. Sehingga dapat dipilih threshold yang lebih baik berdasarkan warna yang akan

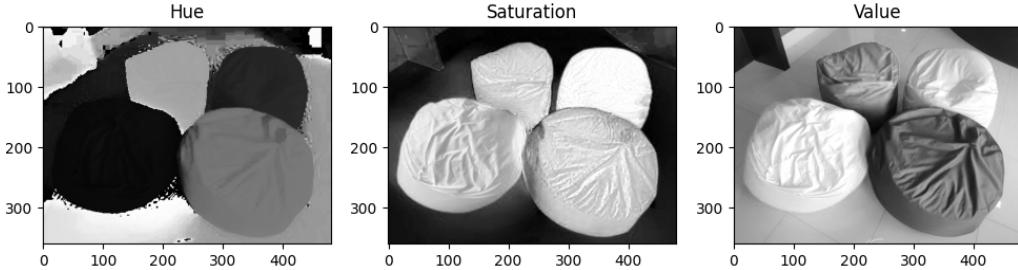


Figure 14: Hasil perubahan citra ke channel HSV dalam tampilan gray

disegmentasi.

```

1 fig, ax = plt.subplots(1, 3, figsize=(15, 5))
2 ax[0].imshow(bags_hsv[:, :, 0], cmap=' hsv ')
3 ax[0].set_title('hue')
4 ax[1].imshow(bags_hsv[:, :, 1], cmap=' hsv ')
5 ax[1].set_title('transparency')
6 ax[2].imshow(bags_hsv[:, :, 2], cmap=' hsv ')
7 ax[2].set_title('value')
8 fig.colorbar(imshow(bags_hsv[:, :, 0], cmap=' hsv '))
9 fig.tight_layout()

```

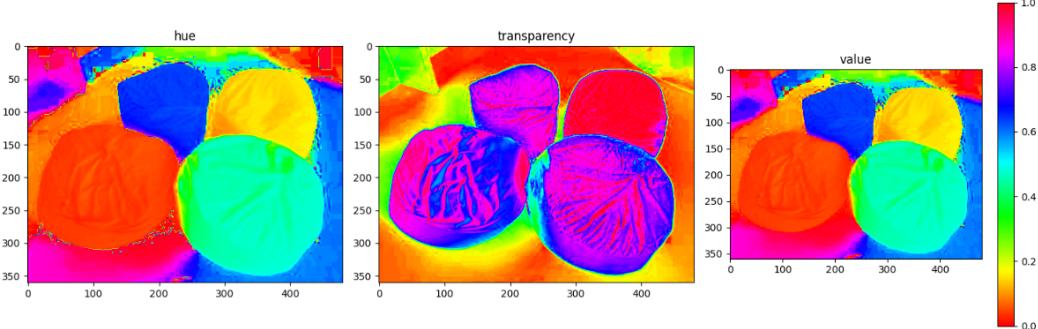


Figure 15: Hasil perubahan citra ke channel HSV dalam tampilan HSV dengan colorbar

Jika diperhatikan, pada colorbar di gambar 15, terdapat warna merah di awal dan di akhir yang mana seakan terlihat sama. Namun jika dilihat pada menggunakan "cmap=gray" di gambar 14, warna merah yang ada di tas kuning dan warna merah di bagian atas tas kuning pada channel Saturation/Transparency, memiliki perbedaan yang cukup signifikan. Oleh karena itu perlu diperhatikan dengan seksama dalam pemilihan threshold nantinya.

Pemilihan threshold juga dapat dilakukan pada 2 channel seperti contoh segmentasi tas warna biru berikut yang memanfaatkan channel hue dan channel saturation.

```

1 #refer to hue channel (in the colorbar)
2 lower_mask = bags_hsv[:, :, 0] > 0.6
3 #refer to hue channel (in the colorbar)
4 upper_mask = bags_hsv[:, :, 0] < 0.7

```

```

5 #refer to transparency channel (in the colorbar)
6 saturation_mask = bags_hsv[:, :, 1] > 0.3
7
8 mask_blue = upper_mask*lower_mask*saturation_mask
9 red = bags[:, :, 0]*mask_blue
10 green = bags[:, :, 1]*mask_blue
11 blue = bags[:, :, 2]*mask_blue
12 bags_masked = np.dstack((red, green, blue))
13 imshow(bags_masked)

```

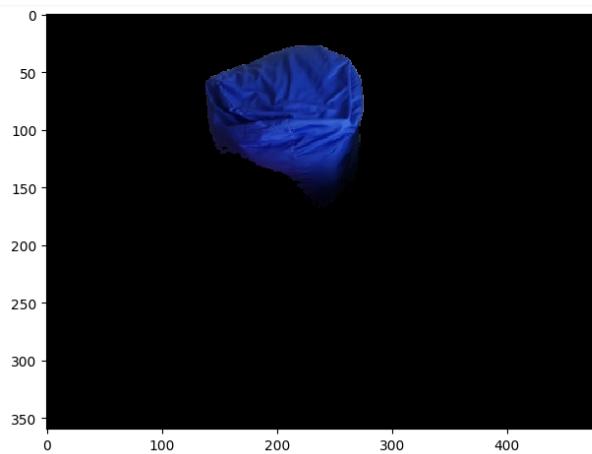


Figure 16: Hasil segmentasi warna biru pada bags.jpg

Berdasarkan gambar 15, tas biru memiliki nilai di antara 0.6 dan 0.7 pada channel hue. Adapun pada channel saturation, tas biru memiliki nilai 0.3. Masing-masing variabel 'lower\_mask', 'upper\_mask', dan 'saturation\_mask' merupakan sebuah array 2 dimensi yang berisi nilai boolean. Untuk didapatkan mask dari tas berwarna biru, dilakukan operasi AND untuk ketiga variabel ini. Hasil operasi AND tersebut akan dikalikan ke masing-masing channel, dan ketika ditampilkan akan muncul tas berwarna biru saja.

Contoh untuk tas kuning dapat diperhatikan pada kode berikut:

```

1 lower_mask = bags_hsv[:, :, 0] > 0.1
2 #refer to hue channel (in the colorbar)
3 upper_mask = bags_hsv[:, :, 0] < 0.2
4 #refer to transparency channel (in the colorbar)
5 saturation_mask = bags_hsv[:, :, 1] > 0.6
6
7 mask_yellow = upper_mask*lower_mask*saturation_mask
8 red = bags[:, :, 0]*mask_yellow
9 green = bags[:, :, 1]*mask_yellow
10 blue = bags[:, :, 2]*mask_yellow
11 bags_masked = np.dstack((red, green, blue))
12 imshow(bags_masked)

```

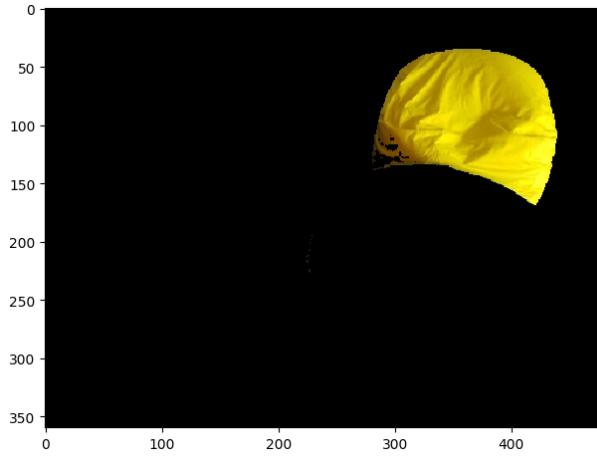


Figure 17: Hasil segmentasi warna kuning pada bags.jpg

Setelah didapat segmentasi untuk tas berwarna biru dan kuning, kita dapat memunculkan kedua tas biru dan kuning ini dalam sebuah tampilan citra. Untuk melakukan hal ini, cukup lakukan operasi 'OR' yang ditandai dengan operasi '+' pada variabel 'mask\_by'.

```

1  mask_by = mask_yellow+mask_blue
2  red = bags [:,:,0]*mask_by
3  green = bags [:,:,1]*mask_by
4  blue = bags [:,:,2]*mask_by
5  bags_masked = np.dstack((red,green,blue))
6  imshow(bags_masked)

```

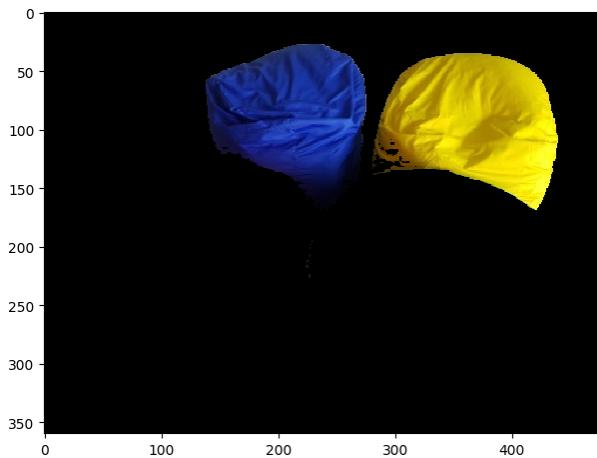


Figure 18: Hasil segmentasi warna biru dan kuning pada bags.jpg