Reyhane Shahrokhian 99521361

HomeWork4 of Computer Vision Course

Dr. Mohammadi

Q1:

Functions are implemented according to the formulas:

```python
def RGB_to_CMYK(r, g, b, RGB_SCALE = 255, CMYK_SCALE = 100):

  #############
  # Your code #
  #############
  r, g, b = r / RGB_SCALE, g / RGB_SCALE, b /RGB_SCALE
  k = 1 - max(r, g, b)

  c, m, y = 0, 0, 0
  if k != 1:
    c = ((1 - r - k) / (1 - k)) *  CMYK_SCALE
    m = ((1 - g - k) / (1 - k)) *  CMYK_SCALE
    y = ((1 - b - k) / (1 - k)) *  CMYK_SCALE

  k = k * CMYK_SCALE

  return c, m, y, k
```

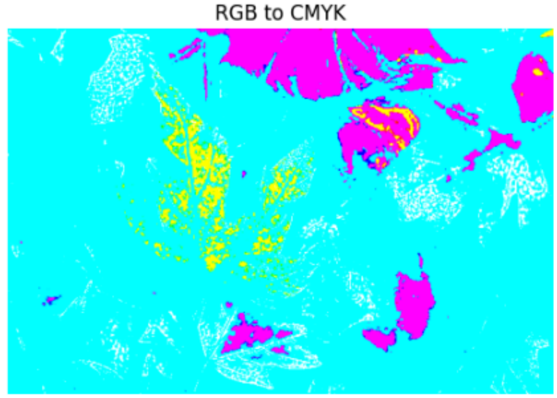
```python
def CMYK_to_RGB(c, m, y, k, RGB_SCALE = 255, CMYK_SCALE = 100):

  #############
  # Your code #
  #############
  c, m, y, k = c / CMYK_SCALE, m / CMYK_SCALE, y / CMYK_SCALE, k / CMYK_SCALE

  r = round((1 - c) * (1 - k) *  RGB_SCALE)
  g = round((1 - m) * (1 - k) *  RGB_SCALE)
  b = round((1 - y) * (1 - k) *  RGB_SCALE)

  return r, g, b
```

```python
import math
def RGB_to_HSI(r, g, b):

  #############
  # Your code #
  #############
  r, g, b = r / 255.0, g / 255.0, b / 255.0
  i = (r + g + b) / 3.0

  minimum = min(r, g, b)
  s = 1 - (3 / (r + g + b)) * minimum if (r + g + b) != 0 else 0

  numerator = 0.5 * ((r - g) + (r - b))
  denominator = math.sqrt((r - g)**2 + (r - b) * (g - b))

  theta = 0
  if denominator != 0:
      theta = math.acos(numerator / denominator) * (180 / math.pi)

  if b <= g:
      h = theta
  else:
      h = 360 - theta

  return h, s, i
```

Outputs:

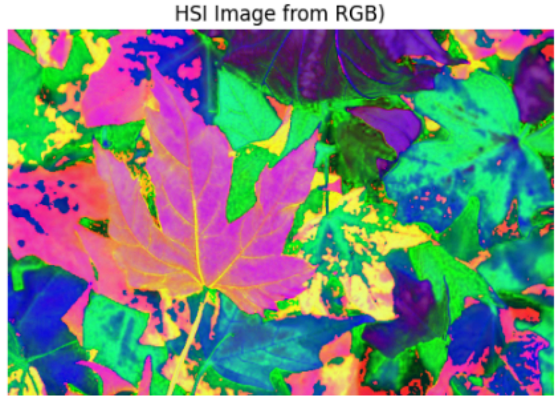
Original RGB Image

RGB to CMYK



RGB Image from CMYK

HSI Image from RGB)



Q2:

Codes are explained via comments:

```
def diff (image1, image2):

  #############
  # Your code #
  #############
  # Resize images to have the same dimensions
  image1_resized = cv2.resize(image1, (image2.shape[1], image2.shape[0]))

  # Compute absolute difference
  diff_image = cv2.absdiff(image1_resized, image2)

  # Create a mask for unchanged pixels
  unchanged_mask = np.all(diff_image == 0, axis=-1)

  # Set color of unchanged pixels to black
  diff_image[unchanged_mask] = [0, 0, 0]

  return diff_image
```

Output(Black parts means that in those areas there was no difference in colors):

Q3:

3.1:

This is the Harris matrix formula:

M = $\sum_{i,j} \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix}$

So:

$$M = \begin{bmatrix} 56 & 56 \\ 56 & 60 \end{bmatrix}$$

3.2:

det(M) = $56 \times 60 - 56 \times 56 = 224$

trace(M) = 56 + 60 = 116

R = 224 - 0.04 × 116 × 116 = -314.24

3.3:

The R score is negative so it's edge. Actually the higher R score indicates the most possibility of being the corner.

- When $|R|$ is small, which happens when $\lambda 1$ and $\lambda 2$ are small, the region is flat.
- When R<0, which happens when $\lambda 1$>>$\lambda 2$ or vice versa, the region is an edge.
- When R is large, which happens when $\lambda 1$ and $\lambda 2$ are large and $\lambda 1 \sim \lambda 2$, the region is a corner.

The image is from this source.

Q4:

The code explanations are all mentioned in the comments:

add black ribbon to grandpa image

```
# Create a black ribbon mask
ribbon_mask = np.zeros_like(grandpa_image[:, :, 0], dtype=np.uint8)

# Define the vertices of the ribbon
ribbon_vertices = np.array([[250, 0], [150, 0], [0, 150], [0, 250]], np.int32)

# Fill the ribbon area in the mask with white color (255)
cv2.fillPoly(ribbon_mask, [ribbon_vertices], 255)

# Make the ribbon area black
grandpa_image[np.where(ribbon_mask == 255)] = [0, 0, 0]

# Display the modified image
cv2_imshow(grandpa_image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

The output:

Then:

```python
def project_image(grandpa_image, room_image, wall_points):
  # Get image dimensions
  grandpa_h, grandpa_w, _ = grandpa_image.shape
  room_h, room_w, _ = room_image.shape

  # Define the coordinates of grandpa_image
  source_points = np.float32([[0, 0], [grandpa_w, 0], [grandpa_w, grandpa_h], [0, grandpa_h]])

  # Define the coordinates on the wall in room_image
  destination_points = np.float32(wall_points)

  # Calculate homography matrix
  homography, _ = cv2.findHomography(source_points, np.float32(wall_points))

  # Perform perspective transform
  warped_image = cv2.warpPerspective(grandpa_image, homography, (room_w, room_h))

  # Add warped image to room image
  room_img = cv2.addWeighted(room_image, 1, warped_image, 1, 0)

  return room_img
```

The cv2.findHomography() function computes the homography matrix. Homography is a transformation matrix that maps the points from one plane to another.

The cv2.warpPerspective() function applies the perspective transformation to the grandpa_image using the calculated homography matrix.

The cv2.addWeighted() function blends the two images together according to their weights. It creates a composite image where the transformed grandpa_image is overlaid onto the original room_image.

But before calling this function, I put a black box in that place then I called that function. The problem is that if I had called the function directly, the image would be blended. But now that is added to a black region(0) the colors had no changes.

Final output:

Q5:

5.1:

For affine transform we need 3 points as there are 6 degrees of freedom.

For A:

$$0a_{11} + 0a_{12} + t_x = 3$$

$$0a_{21} + 0a_{22} + t_y = 2$$

For B:

$$1a_{11} + 0a_{12} + t_x = 4$$

$1a_{21} + 0a_{22} + t_y = 1$

For D:

$1a_{11} + 2a_{12} + t_x = 1$

$1a_{21} + 2a_{22} + t_y = 2$

By solving the above equalizations we have:

$a_{11} = 1,\ a_{12} =- \frac{3}{2},\ a_{21} =- 1,\ a_{22} = \frac{1}{2},\ t_x = 3,\ t_y = 2$

So:

$$\begin{bmatrix} x^{'} \\ y^{'} \end{bmatrix} = \begin{bmatrix} 1 & -\frac{3}{2} & 3 \\ -1 & \frac{1}{2} & 2 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

5.2:

$$\begin{bmatrix} x_c^{'} \\ y_c^{'} \end{bmatrix} = \begin{bmatrix} 1 & -\frac{3}{2} & 3 \\ -1 & \frac{1}{2} & 2 \end{bmatrix} \begin{bmatrix} 2 \\ 1 \end{bmatrix} = \begin{bmatrix} 3.5 \\ 0.5 \end{bmatrix}$$

$$\begin{bmatrix} x_e^{'} \\ y_e^{'} \end{bmatrix} = \begin{bmatrix} 1 & -\frac{3}{2} & 3 \\ -1 & \frac{1}{2} & 2 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 1.5 \\ 2.5 \end{bmatrix}$$

Q6:

تصویری	affine	شباهت	rigid	انتقال	نوع تبدیل
✗	✗	✗	✓	✓	فاصله جفت نقطات ثابت میماند
✗	✗	✓	✓	✓	زاویه بین جفت خط ثابت میماند
✓	✓	✓	✓	✓	خط ها، خط باقی مانند
✗	✗	✗	✗	✓	زاویه بین هر خط و محور ایکس ثابت میماند
✓	✓	✓	✓	✓	چهار ضلعی ها، چهار ضلعی باقی می مانند
✗	✓	✓	✓	✓	خطوط موازی، موازی باقی می مانند
✗	✗	✓	✓	✓	دایره ها، دایره باقی می مانند
✗	✗	✗	✓	✓	نسبت بین مساحت دو شکل ثابت باقی می ماند

Q8:

8.1:

- First we should multiply the matrix with the 3-dimension point(X):

$$\begin{bmatrix} 5 & -14 & 2 & 17 \\ -10, & -5, & -10, & 50 \\ 10 & 2 & -11 & 19 \end{bmatrix} \times \begin{bmatrix} 0 \\ 2 \\ 2 \\ 1 \end{bmatrix} = \begin{bmatrix} -7 \\ 20 \\ 1 \end{bmatrix}$$

Converting to cartesian:

As the third value is 1 so the cartesian form of x is:

$$\begin{bmatrix} -7 \\ 20 \\ 1 \end{bmatrix}$$

8.2:

- The camera calibration matrix is actually the intrinsic matrix that can be calculated using the given parameters:

  $f_x = f_y = \frac{5mm}{0.02} = 250$ (focus lengths)

  $c_x = c_y = 500$ (coordinates of principle point)

  => k = [[250, 0, 500],]
  [[0, 250, 500],]
  [[0, 0 , 1]]

- Rotation matrix: R= [[1, 0, 0],]
  [[0, 1, 0],]
  [[0, 0, 1]]

  Translation matrix: T = [0]
  [0]
  [0]

  The extrinsic matrix(including R and T): E= [[1, 0, 0, 0],]
  [[0, 1, 0, 0],]
  [[0, 0, 1, 0]]

- $P_i^{'}$ = [[250, 0, 500, 0],] * [100, = [425000, 437500, 800]
  [[0, 250, 500, 0],] 150,
  [[0, 0 , 1 , 0]] 800,
  1]

  $P_i = (P_i^{'}[0]\ /\ P_i^{'}[2], P_i^{'}[1]\ /\ P_i^{'}[2]) = (531.25, 546.875)$