Programming Assignment 7 Report

November 9, 2020

Pooya Kabiri - 96521434

Importing prerequisites

[1]:
```
import cv2
import numpy as np
import matplotlib.pyplot as plt
import os
```

[2]:
```
def plotter(img_list, r, w, gray, wr, hr, fig_name = None):
    '''
    Plots images' list with its' caption and saves result image if you want.

    Parameters:
        img_list (list): The list of tuples of image and its' caption.
        r (int): The number of row(s).
        w (int): The number of column(s).
        gray (bool): The flag for plotting images in grayscale mode.
        wr (int): The width of one figure.
        hr (int): The height of one figure.
        fig_name (str): The name of the image of the plot. if not set this␣
 ↪parameter the plot doesn't save.
    '''

    plt.rcParams['figure.figsize'] = (wr, hr)
    for i in range(len(img_list)):
        plt.subplot(r, w, i + 1)
        if img_list[i][2] == 'img':
            if gray:
                plt.imshow(img_list[i][0], cmap = 'gray')
            else:
                plt.imshow(img_list[i][0])
            plt.xticks([])
            plt.yticks([])
        elif img_list[i][2] == 'hist':
            plt.bar(np.arange(len(img_list[i][0])), img_list[i][0], color = 'c')
        else:
```

```
            raise Exception("Only image or histogram. Use third parameter of␣
↪tuples in img_list and set it to img or hist.")
        plt.title(img_list[i][1])
    if fig_name is not None:
        plt.savefig(fig_name + '.png')
    plt.show()
```

1 Color Space Manipulation

1.1 Color Space Conversion

1.1.1 BGR to HSV

`convert_to_hsv(image)` Gets an image as an argument and returns the image in HSV color space using OpenCv `cvtColor` method.

[3]:
```
def convert_to_hsv(image):
    '''
    Converts the color space of the input image to the HSV color space.

    Parameters:
        image (numpy.ndarray): The input image.

    Returns:
        numpy.ndarray: The result image.
    '''

    out_img = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)

    return out_img
```

1.1.2 BGR to YCbCr

`convert_to_ycbcr(image)` Gets an image as an argument and returns the image in YCbCr color space using OpenCv `cvtColor` method.

[4]:
```
def convert_to_ycbcr(image):
    '''
    Converts the color space of the input image to the YCbCr color space.

    Parameters:
        image (numpy.ndarray): The input image.

    Returns:
        numpy.ndarray: The result image.
    '''

    out_img = cv2.cvtColor(image, cv2.COLOR_BGR2YCrCb)
```

```
    return out_img
```

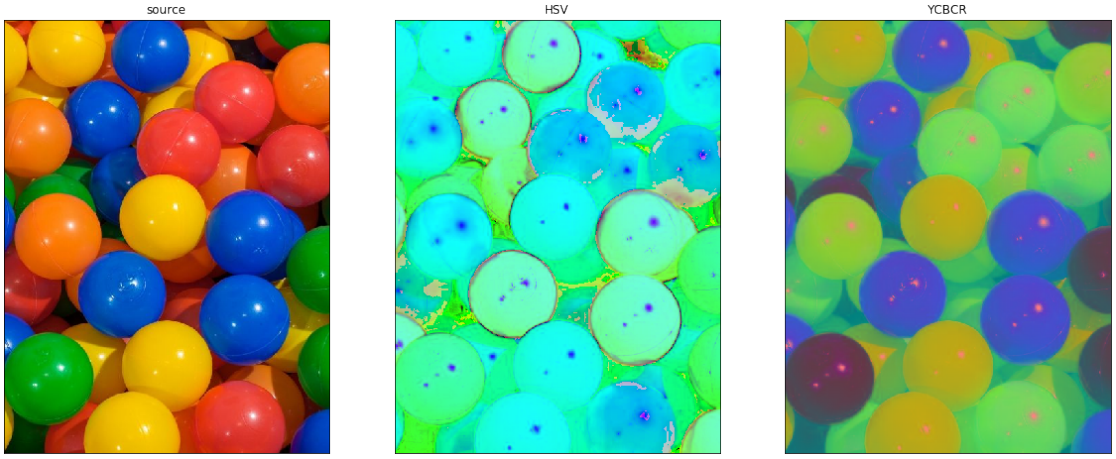
Test of implementation.

[5]:
```
image_list = []

image = cv2.imread(os.path.join('images', '5.jpg'))
image_list.append([cv2.cvtColor(image, cv2.COLOR_BGR2RGB), 'source', 'img'])

image_list.append([convert_to_hsv(image), 'HSV', 'img'])
image_list.append([convert_to_ycbcr(image), 'YCBCR', 'img'])

plotter(image_list,1 , 3, True, 20, 10, '2A')
```



1.2 Finding Differences

In this section, we try to make subtle differences in two similar images more visible to human eye. For achieving this, we use the first image as an Red color channel in RGB color space, and the second image as the Green and Blue color channels in RGB color space.

`get_dif(image1, image2)` Gets two grayscale images as arguments and uses OpenCV `merge` method to merge 3 channels of RGB color space and create an output image. We use `image1` as the R channel and `image2` as the B and G channel. In the output generated by the `get_dif` function, the differences are shown in color in contrast to similarities which are show using shades of gray.

[6]:
```
def get_dif(image1, image2):
    '''
    Creates a new image that differences between two input images are shown.

    Parameters:
```

```
        image1 (numpy.ndarray): The first input image.
        image2 (numpy.ndarray): The second input image.

    Returns:
        numpy.ndarray: The result difference image.
    '''

    diff_image = cv2.merge((image1, image2, image2))

    return diff_image
```

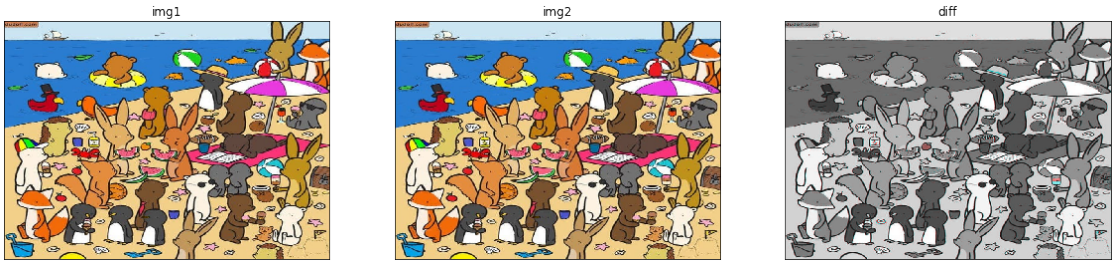
Test of implementation

[7]:
```
image_list = []

image1 = cv2.imread(os.path.join('images', '5b1.jpg'))
image_list.append([cv2.cvtColor(image1, cv2.COLOR_BGR2RGB), 'img1', 'img'])
image2 = cv2.imread(os.path.join('images', '5b2.jpg'))
image_list.append([cv2.cvtColor(image2, cv2.COLOR_BGR2RGB), 'img2', 'img'])

image1 = cv2.imread(os.path.join('images', '5b1.jpg'), cv2.IMREAD_GRAYSCALE)
image2 = cv2.imread(os.path.join('images', '5b2.jpg'), cv2.IMREAD_GRAYSCALE)
image_list.append([get_dif(image1, image2), 'diff', 'img'])

plotter(image_list,1 , 3, True, 20, 10, '2B')
```



As we can see in the result image, There are 6 areas of difference:

1. The flag in bottom right
2. The ice cream in bottom right
3. The hat in center top
4. The orange in center mid left
5. The hand lotion / sanitizer in center left
6. The sea shell in bottom left

2 Video Processing

In this section the goal is to process an entire video, frame by frame, and change the red balls to appear white. To achieve this, each frame of video is processed. **First** the frame is splitted into 3 color channels of R, G, B. **Then** the frame is converted to YCbCr color space and splitted to three Y, Cr, Cb channels. As Cr channel shows the difference with red colors, The areas in Cr channel with high pixel intensity values shows locations of pixels in the original image which had values of high red. We simply change pixel values of this areas to (255, 255, 255) which is white in BGR space. For detecting the mentioned areas in Cr channel we use a threshold (here 150) which is empirically determined.

`process_frame(frame)` Takes a frame and changes red areas to white with the method explained above. Returns the processed frame.

[8]:
```
def process_frame(frame):
    '''
    Converts red circles in the input image to white circles.

    Parameters:
        frame (numpy.ndarray): The input frame.

    Returns:
        numpy.ndarray: The result output frame.
    '''

    result = frame.copy()
    blue, green, red = cv2.split(frame)

    ycrcb = cv2.cvtColor(frame, cv2.COLOR_BGR2YCrCb)
    y, cr, cb = cv2.split(ycrcb)

    ### 150 is the threshold which is empirically determined for the purpose of␣
↪this example.
    rows, cols = np.where(cr > 150)
    result[rows, cols, :] = 255

    return result
```

Test your implementation (dont change this cell)

[9]:
```
cap = cv2.VideoCapture('balls.mp4')

frame_width = int(cap.get(3))
frame_height = int(cap.get(4))

size = (frame_width, frame_height)
fps = cap.get(cv2.CAP_PROP_FPS)
out = cv2.VideoWriter('balls_processed.mp4',
```

```
                        cv2.VideoWriter_fourcc(*'DIVX'),
                        fps, size)

while(cap.isOpened()):
    ret, frame = cap.read()
    if ret:
        out.write(process_frame(frame))
    else:
        break
out.release()
cap.release()
```

[10]:
```
from IPython.display import HTML

text =  """
<table>
<tr>
    <td>
        <h3>Input</h3>
        <video alt="input" width="400" height="240" autoplay>
            <source src="balls.mp4" type="video/mp4">
        </video>
    </td>
    <td>
      <h3>Processed</h3>
      <video alt="output" width="400" height="240" autoplay>
            <source src="balls_processed.mp4" type="video/mp4">
      </video>
    </td>
</tr>
</table>

"""
HTML(text)
```

[10]:
```
<IPython.core.display.HTML object>
```

3 Corner Detection

In this section we are going to achieve corner detection. for the purpose of this assignment we are going to implement Harris Corner Detector.

`harris_points(image)` Takes an image as an argument and returns the corner detected image. The corners detected are shown by red points.

[13]:
```
### Intuition taken from: https://github.com/hughesj919/HarrisCorner/blob/
 ↪master/Corners.py
```

```
def harris_points(image):
    '''
    Gets corner points by applying the harris detection algorithm.

    Parameters:
        image (numpy.ndarray): The input image.

    Returns:
        numpy.ndarray: The result image.
    '''
    out_image = image.copy()

    ### Threshold for Non-maximum suppression
    threshold = 0.5

    K = 0.05
    window_size = 3

    ### This kernel is used as a window
    sum_kernel = np.ones((window_size, window_size))

    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

    ### Computing necessary gradients
    Ix, Iy = np.gradient(gray)

    Ixx = np.square(Ix)
    Iyy = np.square(Iy)
    Ixy = Ix * Iy

    ### Computing the sums over windows on gradients using convolution
    SIxx = cv2.filter2D(Ixx, -1, sum_kernel)
    SIyy = cv2.filter2D(Iyy, -1, sum_kernel)
    SIxy = cv2.filter2D(Ixy, -1, sum_kernel)

    result = (SIxx * SIyy) - np.square(SIxy) - (K * np.square(SIxx + SIyy))

    ### We normalize the values of result using OpenCV normalize method and␣
↪using MINMAX method
    ### which maps the lowest value to alpha=0 and the maximum value to beta=1␣
↪and uses a linear transform
    ### for the values between.
    cv2.normalize(result, result, 0, 1, cv2.NORM_MINMAX)

    ### Finding promising locations using NMS
    rows, cols = np.where(result >= threshold)
```

```
    points = list(zip(rows, cols))

    ### Drawing red circles on the locations which are determined by NMS
    for point in points:
        cv2.circle(out_image, point, 5, (255, 0, 0), -1)

    return out_image
```

Test of implementation.

[14]:
```
image_list = []

image = cv2.imread(os.path.join('images', '7.jpg'))
image_list.append([cv2.cvtColor(image, cv2.COLOR_BGR2RGB), 'source', 'img'])

image_list.append([harris_points(image), 'harris_points', 'img'])

plotter(image_list,1 , 2, True, 20, 10, '7')
```