01_colorspaces

October 27, 2019

1 Assignment 1: Color Spaces, Morphological Operators

1.1 Exercise 1.1

For an image of your choice, implement the simple binarization method as shown in the lecture. We've put some example images in in /images.

Rough sketch:

1. define the positive" subspace P in the RGB cube
2. iterate over all pixels in I and check if in P or ~P
3. write result to new image
4. play around with size and shape of P and display binary image (**RESULT**)

```
In [79]: from dataclasses import dataclass
         from typing import Tuple, Union

         import numpy as np


         class RgbSubscriptable:
             def __getitem__(self, key):
                 if key == 0:
                     return self.r
                 if key == 1:
                     return self.g
                 if key == 2:
                     return self.b
                 raise KeyError("Key must be an int and one of {0, 1, 2}.")

             def __str__(self):
                 return f"<{self.__class__.__name__} r={self.r} g={self.g} b={self.b}>"


         class Threshold(RgbSubscriptable):
             NONE = Threshold.uniform(0)

             def __init__(self, r, g, b):
```

```
        self.r = r
        self.g = g
        self.b = b

    @classmethod
    def uniform(cls, n):
        return cls(n, n, n)


class Color(RgbSubscriptable):
    def __init__(self, r, g, b, threshold=Threshold.NONE):
        self.r = r
        self.g = g
        self.b = b
        self.threshold = threshold


@dataclass
class ColorSpace:
    name: str
    colors: Tuple[Color]
    global_threshold: Threshold = Threshold.NONE

    @classmethod
    def from_range(cls, r, g, b):
        means = (np.mean(r), np.mean(g), np.mean(b))
        return cls(
            colors=(
                Color(*means, threshold=Threshold(means[0] - r[0], means[1] - g[0], me
            ),
        )

    def contains(self, check_color):
        return any(self._is_similar(own_color, check_color) for own_color in self.colo

    def _is_similar(self, own_color, check_color):
        local_threshold = own_color.threshold
        global_threshold = self.global_threshold

        def comp_similar(i):
            comp_own = own_color[i]
            comp_check = check_color[i]
            thresh = local_threshold[i] + global_threshold[i]
            return comp_own - thresh <= comp_check <= comp_own + thresh

        return all(
            comp_similar(i)
            for i in range(0, 3)
```

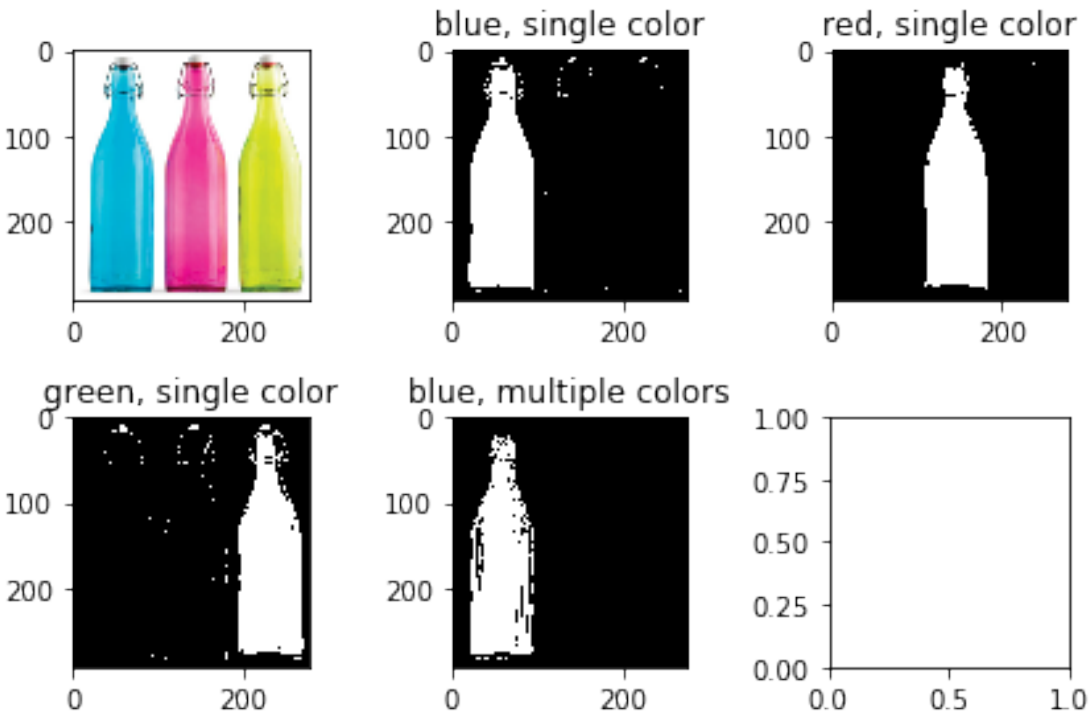
```
            )

        def __str__(self):
            colors = ', '.join(str(color) for color in self.colors)
            return f"<ColorSpace name='{self.name}' colors=({colors}) threshold={str(self
```

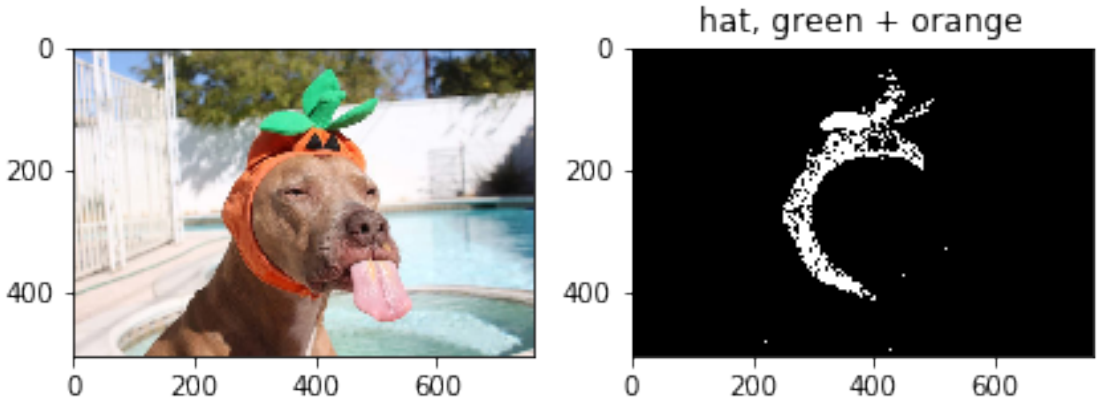
In [106]:
```
from skimage import io, data, color
from skimage.util import img_as_ubyte
import numpy as np


images = io.imread_collection('images/*')


def handle_image(image, subspaces):
    if image.shape[2] == 4:
        # simply drop alpha channel
        image = image[:,:,:-1]

    shape = image.shape
    print('shape =', shape)

    linear_shape = (image.shape[0]*image.shape[1], 3)
    linear_image = image.reshape(linear_shape)
    ones = np.full(linear_shape[1:], 255)

    binarized_images = []
    for i, subspace in enumerate(subspaces):
        print(subspace.name, '...........')
        binarized_image = np.full(linear_shape, 0)

        for i, pixel in enumerate(linear_image):
            if subspace.contains(pixel, i):
                binarized_image[i] = ones
        binarized_image = binarized_image.reshape(shape)
        binarized_images.append(binarized_image)
    fig = io.imshow_collection([image] + binarized_images)
    for subspace, ax in zip(("",) + subspaces, fig.axes):
        ax.set_title(getattr(subspace, 'name', subspace))
    fig.tight_layout()


colors = (
    # BOTTLES
    (
        # 1. single color for each bottle
        # bottle 1 (blueish)
        ColorSpace(
```

```
            name="blue, single color",
            colors=(
                Color(0, 179, 215, Threshold(200, 50, 50)),
            ),
        ),
        # bottle 2 (reddish)
        ColorSpace(
            name="red, single color",
            colors=(
                Color(236, 75, 155, Threshold(50, 100, 50)),
            ),
        ),
        # bottle 3 (greenish)
        ColorSpace(
            name="green, single color",
            colors=(
                Color(207, 220, 39, Threshold(60, 60, 160)),
            ),
        ),

        # 2. more colors per bottle, smaller threshold
         # bottle 1
        ColorSpace(
            name="blue, multiple colors",
            colors=(
                Color(0, 183, 215),
                Color(0, 188, 220),
                Color(0, 178, 216),
                Color(107, 204, 228),
                Color(155, 216, 235),
                Color(1, 158, 188),
                Color(83, 198, 224),
                Color(1, 76, 88),
                Color(1, 177, 207),
                Color(0, 167, 203),
            ),
            global_threshold=Threshold.uniform(20),
        ),
        # picking colors for the other 2 bottles is no fun... :/
    ),


    # DOG
    (
        ColorSpace(
            name="hat, green + orange",
            colors=(
                # green
```

```
                        Color(82, 236, 174),
                        Color(23, 137, 79),
                        Color(38, 176, 107),
                        # orange
                        Color(186, 45, 8),
                        Color(89, 6, 0),
                        Color(241, 122, 90),
                        Color(248, 147, 103),
                        Color(237, 91, 42),
                        Color(188, 88, 49),
                    ),
                    global_threshold=Threshold(38, 38, 15),
                ),
            ),
        )

        i = -1
        for image, subspaces in zip(images, colors):
            i += 1
        #     if i != 0:
        #         continue
            handle_image(image, subspaces)
```

```
shape = (293, 277, 3)
blue, single color ...
red, single color ...
green, single color ...
blue, multiple colors ...
shape = (506, 760, 3)
hat, green + orange ...
```





1.2 Exercise 1.2

- starting from the binary color detection image
- erase noise with an erosion operation
- dilate once to get original size of object
- find connected components with the two-pass algorithm
- extract bounding box on the fly
- draw bounding box on original image (**RESULT**)

1.3 Exercise 1.3

- use your color detection and connected components algorithm
- implement simplest tracking algorithm
- draw history of all previous points on frame (**RESULT**)

(see images/racecar or images/taco for sample image sequences)