

Computer Vision Programming Assignment 01

Topics - HSV color space, Histograms, Nearest Neighbor

Program must be in python using opencv and numpy --- these imports:

```
import cv2 as cv
import numpy as np
```

Learning Goals –

- familiarity with coding using numpy and opencv
- experience implementing some things we learned early in the course including:
HSV color space, Histograms, computing distance between histograms, and 1-Nearest Neighbor

1. Write a function BGR_to_HSV

- it should take a BGR image (that is, ndarray with dtype uint8) as a parameter
- it should return an HSV "image"

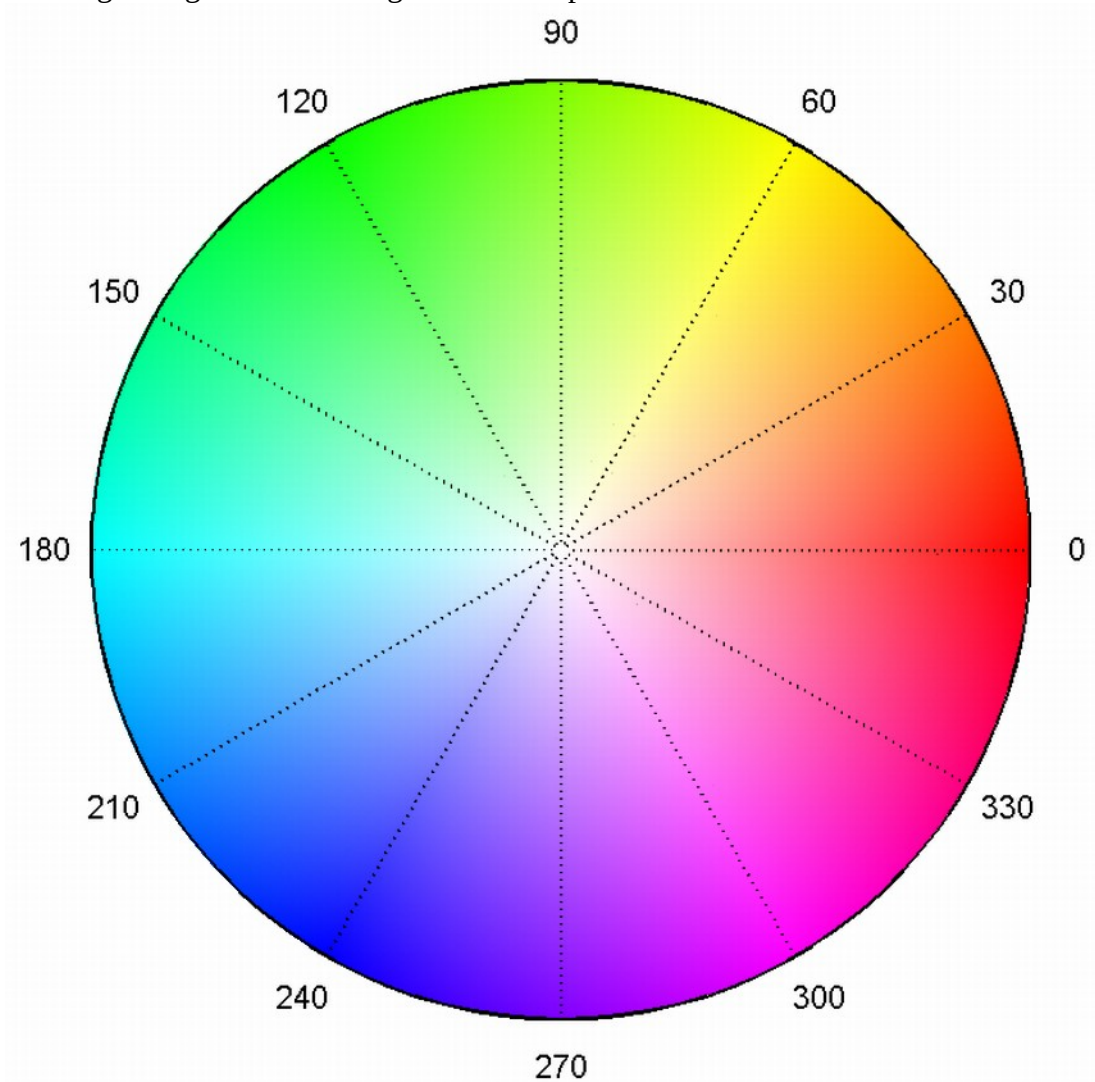
You must choose an appropriate type for the values of the three HSV channels. All three channels are required to be the same type. Make H be in the range 0-359 and S and V both be 0-255. Use the formulas in the conversion code (from Foley, van Dam, Feiner and Hughes – Computer Graphics) below, but instead you have BGR each in the 0-255 range and instead of S and V in range 0-1, make them be in the range 0-255. Note: B, G, R, S, and V are all of the same range. The floats used in the code below should be integers in your code. In the situation when H is undefined, let it be 361. Note: you are not allowed to call cv.cvtColor or any other opencv calls that provide the ability to convert among color spaces.

592 Achromatic and Colored Light

```
void RGB_To_HSV (double r, double g, double b, double *h, double *s, double *v)
/* Given: r, g, b, each in [0,1]. */
/* Desired: h in [0,360), s and v in [0,1] except if s = 0, then h = UNDEFINED, */
/* which is some constant defined with a value outside the interval [0,360]. */
{
    double max = Maximum (r, g, b);
    double min = Minimum (r, g, b);
    *v = max;                                /* This is the value v. */
    /* Next calculate saturation, s. Saturation is 0 if red, green and blue are all 0 */
    *s = (max != 0.0) ? ((max - min) / max) : 0.0;
    if (*s == 0.0)
        *h = UNDEFINED;
    else {
        double delta = max - min;           /* Chromatic case: Saturation is not 0, */
                                           /* so determine hue. */
        if (r == max)
            *h = (g - b) / delta;           /* Resulting color is between yellow and magenta */
        else if (g == max)
            *h = 2.0 + (b - r) / delta;     /* Resulting color is between cyan and yellow */
        else if (b == max)
            *h = 4.0 + (r - g) / delta;     /* Resulting color is between magenta and cyan */
        *h *= 60.0;                         /* Convert hue to degrees */
        if (*h < 0.0)
            *h += 360.0;                   /* Make sure hue is nonnegative */
    }
} /* Chromatic case */
} /* RGB_To_HSV */
```

Fig. 13.33 Algorithm for converting from RGB to HSV color space.

2. Write a function to select pixels in the yellow range of an HSV image. Write another function to select pixels in the green range. See this for a visual representation of Hue angles. Come up with the Hue angle ranges from looking at the circle picture below.



The function to select the yellow pixels should take in an HSV image and return a mask of same size whose pixels are white (value 255) if the corresponding pixel in the HSV image is yellow and black (value 0) everywhere else.

The function to select the green pixels should take in an HSV image and return a mask of same size whose pixels are white (value 255) if the corresponding pixel in the HSV image is green and black (value 0) everywhere else.

Note: By same size I mean the mask should be a 2d array of same number of rows and columns and have type uint8 e.g.

```
mask=np.empty((hsvimg.shape[0],hsvimg.shape[1]),np.uint8)
```

Note: In both functions, ignore pixels whose Saturation and/or Value are below 50.

Function names: `yellow_mask(hsvimg)` and `green_mask(hsvimg)`

Note well: for these two functions, do not use `cv.inrange`, instead do this yourself in some manner. Use the masks returned from those functions with the original BGR image as is done in the demo code here (just do the `cv.bitwise_and` call):

https://docs.opencv.org/4.13.0/df/d9d/tutorial_py_colorspaces.html

to get an image of only the pixels in original image that are green or yellow and everywhere else black.

For example, if the mask that is returned is in `green_msk` and the original BGR image is named `img`, you can call the `bitwise_and` like so:

```
green_pixels=cv.bitwise_and(img, img, mask = green_msk)
```

Use these masks to test out your HSV conversion to some degree (pun intended).

Extra credit: Write a general function named `hue_mask` to select a range of hues. Parameters are (some with default values if not provided) `minhue`, `maxhue`, `minsat=50`, `minval=50`

Also allow `minhue` to be $> \text{maxhue}$ and if that is the case, then hue should span 0. e.g. if `minhue = 340` and `maxhue = 20` that would be a range representing reds.

3. Write a `compute_HS_histogram` function that takes in an HSV image and ignores all the pixels with $S \leq 50$ (those are the very low saturation, closer to white pixels) and all the pixels with $V < 50$ (those are the very dark, closer to black pixels) and generates a 1d histogram with the following bins:

Nine Hue ranges	index
H 0-39	0
H 40-79	1
H 80-119	2
H 120-159	3
H 160-199	4
H 200-239	5
H 240-279	6
H 280-319	7
H 320-359	8

and Five Saturation
ranges for each

Hue range	index
S 51-91	0
S 92-132	1
S 133-173	2
S 174-214	3
S 215-255	4

for a total of $9 \times 5 = 45$ bins

then normalize this histogram so that it produces a ndarray of size 45 whose values all add up to 1 and return this normalized histogram

Note: You can't take advantage of that bit shifting idea I presented because the channels are not divided into a power-of-two groups. I suggest you create a 2d numpy array whose first index is on Hue and second index is on Saturation. `//` is integer divide, so `H//40` will give a number 0 .. 8 which will be the first index and `(S-51)//41` should result in an int 0 .. 4 which is the second index.

Then after you create the 2d array you can change its shape to a 1d array of length 45. That's probably the easiest way to do it.

Example: An HSV = (295, 60, 80) will be counted because $60 > 50$ and $80 \geq 50$. The indices of the histogram (as a 2d array) whose element you'll want to increment by 1 are: `295//40`, `(60-51)//41`
Notice: `295//40` is 7 and `(60-51)//41` is 0 which are the two indices you need (see the ranges list above with the indices next to them)

Extra credit: generalize this function to allow extra parameters to generate histograms of various sizes. Parameters w/ their defaults are: `huebinwidth=40`, `satbinwidth=41`, `minsat=51`, `minval=50`
These values will determine the number of bins in your histogram (default values result in 45 bins.)

4. Implement L1 and L2 distance functions (each should take in 2 normalized 1d histograms (ndarrays) of any length) and return the distance. For L1 it should return the sum of the absolute differences of corresponding elements. For L2 it should return the square root of the sum of the squared differences of corresponding elements.

5. Implement the 1-nearest neighbor algorithm in a function name `nn1`. Your function should take in a list of image file names as one parameter. The first element (at index 0) will be the key image file name and the remaining image file names are potential matches of the key image. The second parameter should be a string that is expected to either be 'L1' or 'L2', and this should default to 'L1' if not passed in.

For each image name in the list, read in the image using `cv.imread`, convert to HSV using your `BGR_to_HSV` function, and compute the normalized 45 bin HS histogram using your `compute_HS_histogram` function. Compute either the L1 or L2 distance between the key histogram and the histogram of every other image and keep track of the image name that is the least distance. Return a tuple of the key image name and the closest match image name.

Extra credit: generalize this function to take in a parameter `k` which defaults to 1
Your function should keep track of (and return) the `k` nearest neighbors.

=====

I'll provide code to generate the image file names and put them in a list. I'll also provide code to generate all the lists of image file names with different key image names. That is if list is size `n`, I'll create `n` lists all with a different first element. Assumptions: my code will expect that your python program will live in some folder that has a folder named `example-images` within it. In the `example-images` folder will live files with extension `.jpg` (not `.jpeg`, or `.JPG` or others). Also, you will run your python program in the folder in which it lives.

When I call your 1-nearest neighbor with each of the lists of names, I'll have code to `imshow` the key and the match images and have a key that needs to be pressed to work on the next list.