Image Operations

February 15, 2018

1 Image operations

Table of Contents

In image manipulations, we can perform operations on colors as well as drowing and adding fonts on them. The dataset we have is grayscaled. I will get a colorful image so that effects of colors will be clear.

1.1 1.0 Import Libraries

```
In [1]: # important imports:
        import cv2
        import numpy as np
        import matplotlib.pyplot as plt

        ## make sure to install dependencies above.
```

1.2 1.1 Read the Image

In [2]:
```
# reading the image
clrd_img = cv2.imread('colorful_image.jpeg', cv2.IMREAD_COLOR)
```

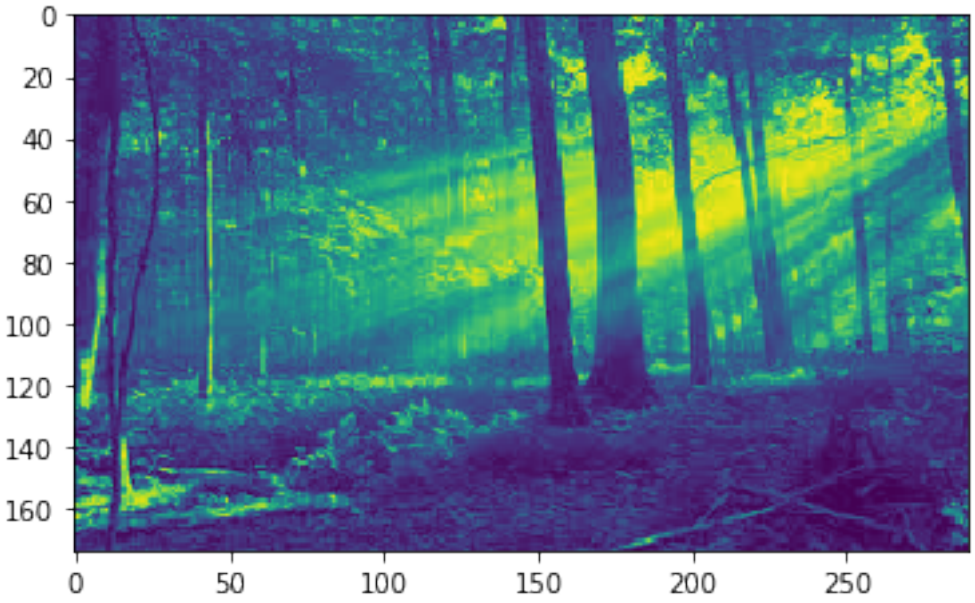
In [3]:
```
# view the image with plot
plt.imshow(clrd_img)
```

Out[3]: <matplotlib.image.AxesImage at 0x7fe3f2803278>

1.3 1.2 Convert whole Image to Grayscale

In [4]:
```
#change the image to grayscale
gray_crld_img = cv2.cvtColor(clrd_img, cv2.COLOR_BGR2GRAY)
plt.imshow(gray_crld_img)
```

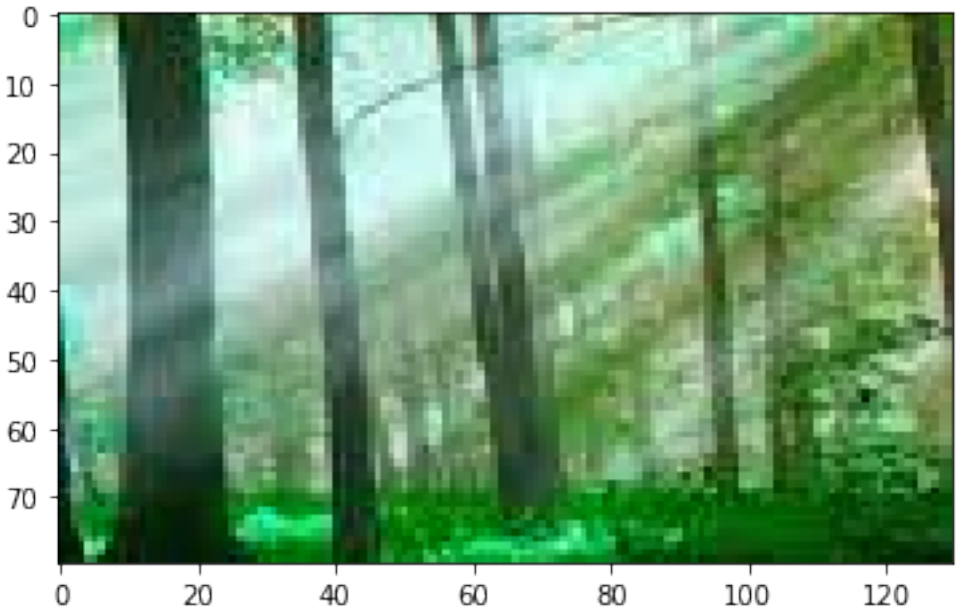
Out[4]: <matplotlib.image.AxesImage at 0x7fe3f27a6400>



1.4 1.3 Convert Range of Image to another color

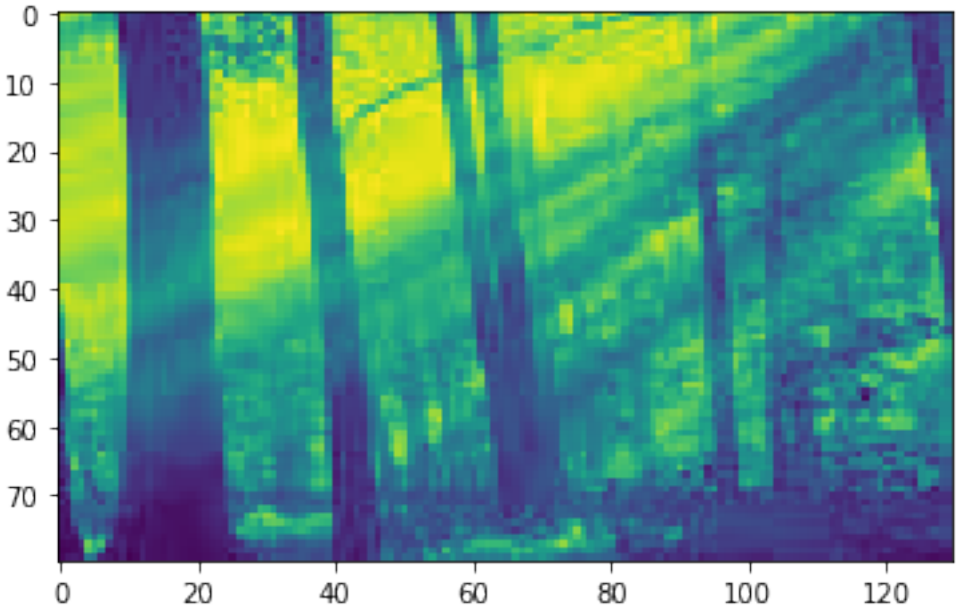
In [5]:
```
# you can use numpy indecies to take a region of image
# see to above pic, to take a region of it:
#the hieght interval is specified in the first index,
# and the width interval is specified in the second index
roi = clrd_img[40:120, 160:290]

# you can also choose a specific pixel to change its value as follows:
# px = img[100, 100]
plt.imshow(roi)
```

Out[5]:
```
<matplotlib.image.AxesImage at 0x7fe3bad765f8>
```




```
In [6]: gray_crld_roi = cv2.cvtColor(roi, cv2.COLOR_BGR2GRAY)
        plt.imshow(gray_crld_roi)
```

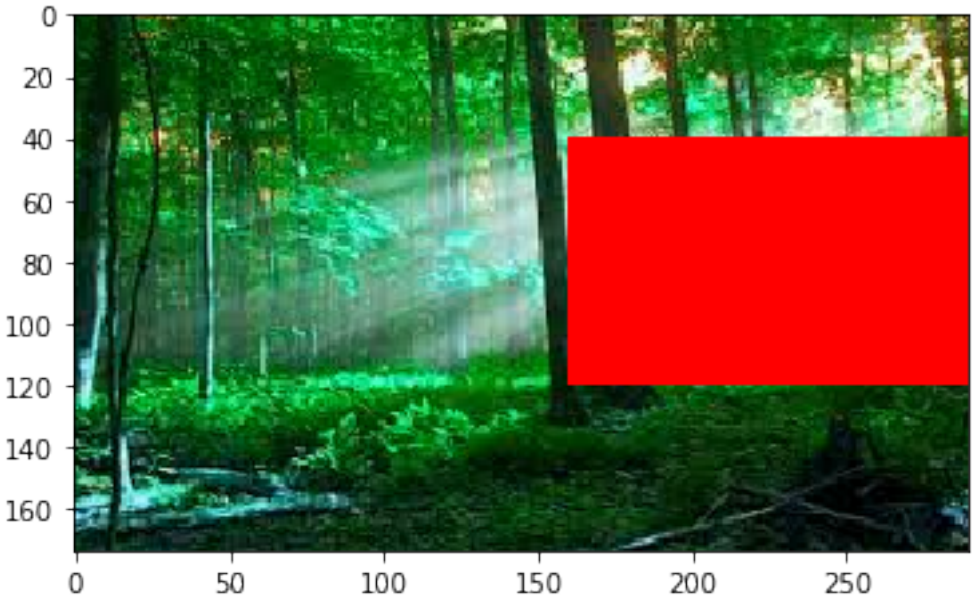
```
Out[6]: <matplotlib.image.AxesImage at 0x7fe3bacd6e48>
```



You can also change this part to any color you like. for example to *red*.

```
In [7]: clrd_img_with_red = clrd_img.copy()
        clrd_img_with_red[40:120, 160:290] = [255, 0, 0]
        plt.imshow(clrd_img_with_red)
```

```
Out[7]: <matplotlib.image.AxesImage at 0x7fe3bacc0978>
```



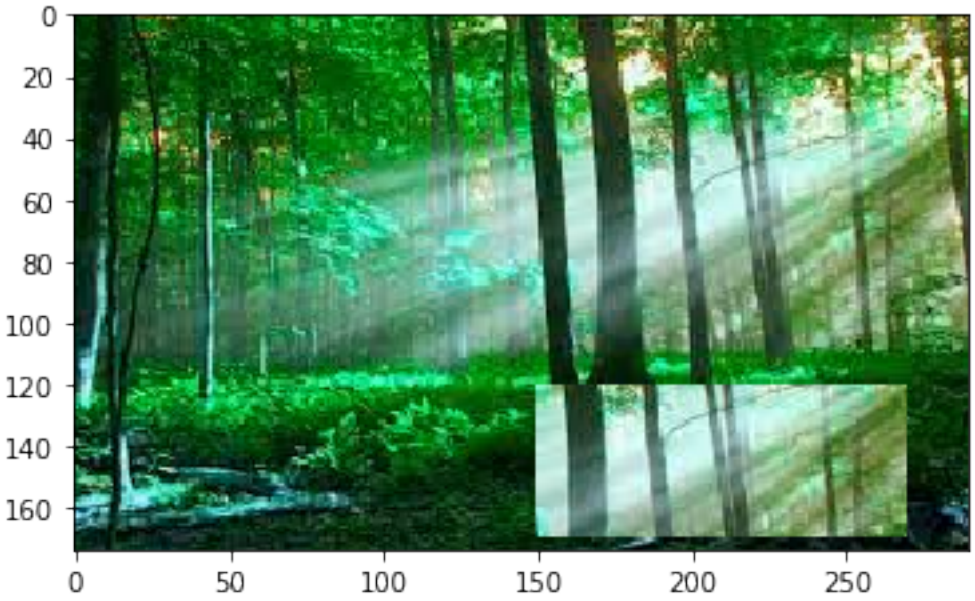
you can move parts of image to other locations as well. look to the following code:

```
In [8]: plt.imshow(clrd_img)
```

```
Out[8]: <matplotlib.image.AxesImage at 0x7fe3bac1feb8>
```

```
In [9]: # keep in mind that the size of each region should be identical.
        # that is img[a:b, c:d] = img[a':b', c':d'] where:
        # |a-b| = |a'-b'| and |c-d| = |c'-d'|
        clrd_img_duplicated_parts = clrd_img.copy()
        clrd_img_duplicated_parts[120:170, 150:270] = clrd_img[40:90, 160:280]
        plt.imshow(clrd_img_duplicated_parts)
```

```
Out[9]: <matplotlib.image.AxesImage at 0x7fe3bac08c50>
```



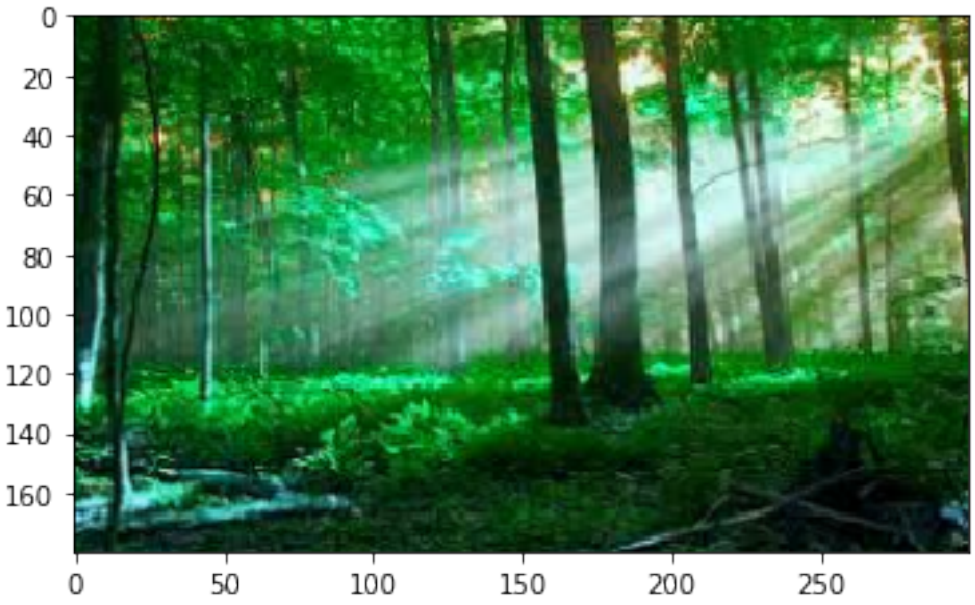
1.5 1.4 Images Geometric Transformations

1.5.1 1.4.1 Image Scaling

OpenCV comes with the method cv2.resize() to resizes images. This resizing includes scaling the image up or down as desired.

```
In [10]: clrd_img_resized = clrd_img.copy()
         clrd_img_resized = cv2.resize(clrd_img_resized,(300,180), fx=2, fy=2)
         # the first parameter is the image.
         # the second paramter is used to define the length of the axis to show
         #       if it is the same as the original image, no scaling will be done,
         #       if it is (0,0), the image will be shown as default with doubled axis scales
         #       you can change the value to see the effect.
         # the third parameter is the scaling regarding the x-axis
         # the fourth paramter is the scaling regarding the y-axis
         plt.imshow(clrd_img_resized)
```

```
Out[10]: <matplotlib.image.AxesImage at 0x7fe3bab72518>
```

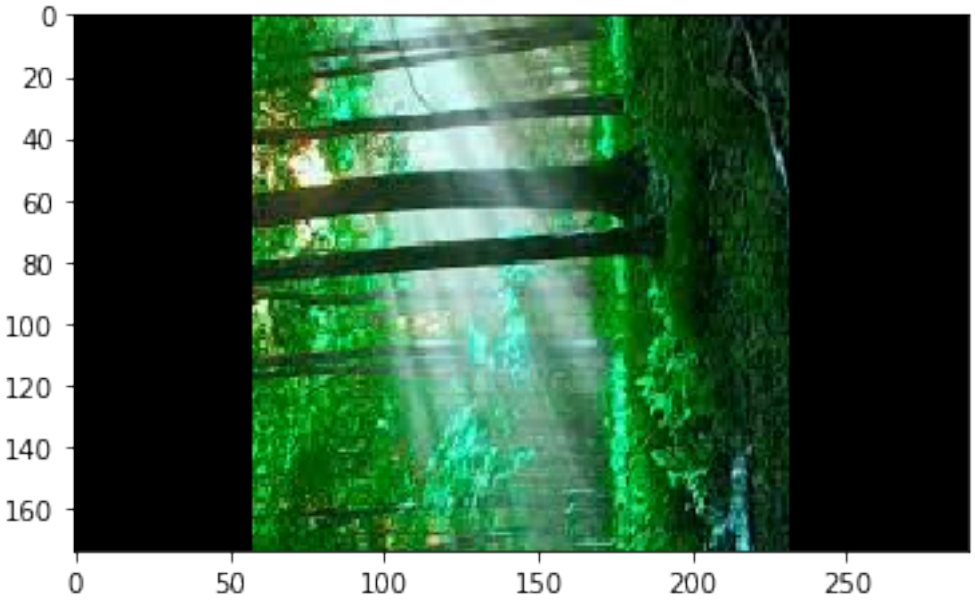


1.5.2 1.4.2 Image rotation

Image rotation is a little bit tricky. To do it, you must first find the rotation matrix then feed it to a function called warpAffine(). These functions are all in the OpenCV

```
In [11]: clrd_img_rotated = clrd_img.copy()
         rows,cols = clrd_img_rotated.shape[0], clrd_img_rotated.shape[1]

         angle = 90
         M = cv2.getRotationMatrix2D((cols/2, rows/2), angle, 1)
         clrd_img_rotated = cv2.warpAffine(clrd_img_rotated, M, (cols, rows))
         plt.imshow(clrd_img_rotated)
```

```
Out[11]: <matplotlib.image.AxesImage at 0x7fe3baad7668>
```



To be honest, I only get to know how to change the angle, but other parameters are not even clear in the documentation. I may read more about it later.

1.6 1.5 Image Thresholding

1.6.1 1.5.1 simple thresholding

Image Thresholding is a very important operation. It is used, I guess and you will probably agree with me after while, by many commercial applications such as CamScanner.

The idea is very simple. For a given image, Image thresholding is to change all pixels of the image having value greater that threshold to a given fixed value. The same thing is applied to the pixels below this threshold.

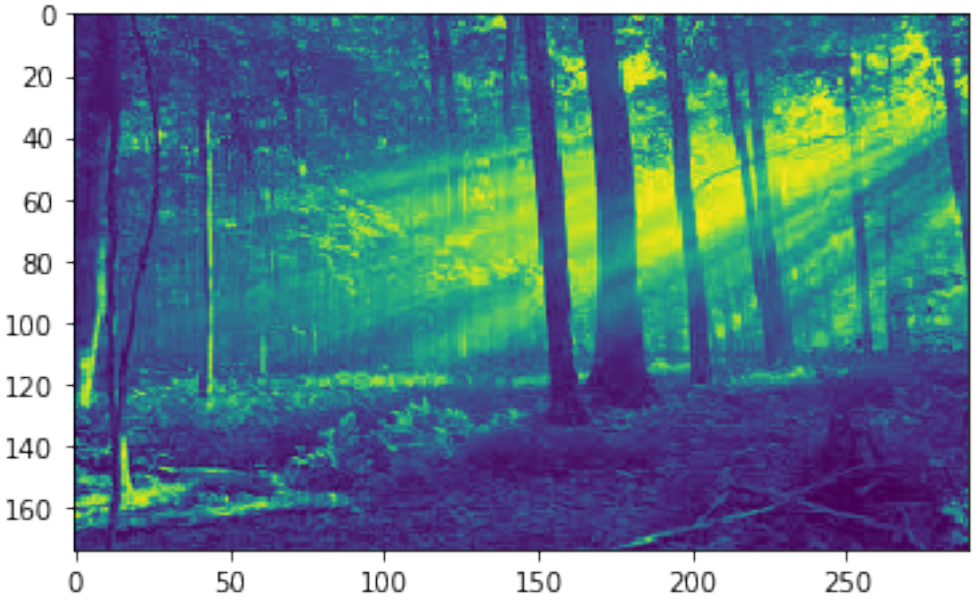
We may apply this on the above colorful images but they will not depict its crucial use. I will apply it on one of them and apply it, then, on another image to show it importance.

It is important to know that this operation is applied to a gray scaled images to know why, values of gray scaled images are of type numpy arrays to represent the values of RGB, but pixel values of gray scaled images are just numbers of black intensity between 0,255 inclusive.

```
In [12]: clrd_img_thresholded = clrd_img.copy()
         clrd_img_thresholded = cv2.cvtColor(clrd_img_thresholded, cv2.COLOR_BGR2GRAY)
         ret, threshold1 = cv2.threshold(clrd_img_thresholded, 125, 50, cv2.THRESH_BINARY)
         # first parameter is the image to be thresholded.
         # second paramter is the threshold by which pixels are calssified.
         # third parameter is the maxVal to which pixels exceeding the threshold are converted
         # fourth paramter is the threshold type. Better to take a look at their differences i
         # https://docs.opencv.org/2.4/doc/tutorials/imgproc/threshold/threshold.html

In [13]: print('original image')
         plt.imshow(clrd_img_thresholded)
```

```
original image
```

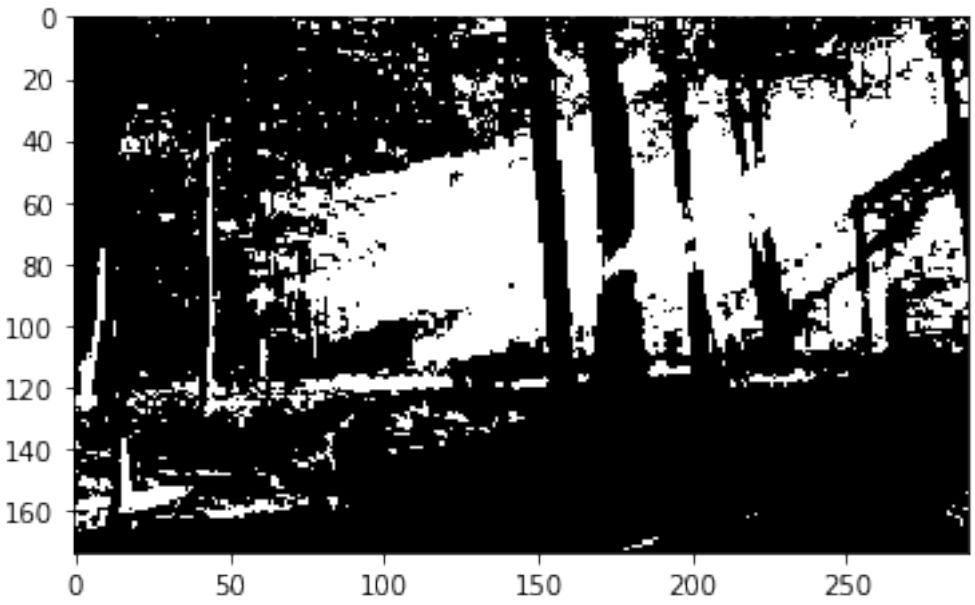
```
Out[13]: <matplotlib.image.AxesImage at 0x7fe3baabe470>
```




```
In [14]: print('thresholded image')
         plt.imshow(threshold1, cmap='gray')
```

```
thresholded image
```

```
Out[14]: <matplotlib.image.AxesImage at 0x7fe3baa19e48>
```

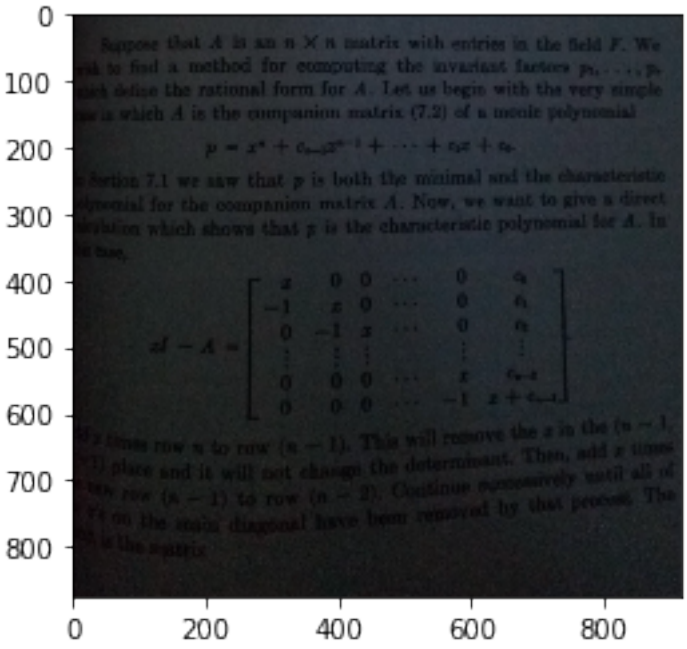


Matplotlib Library is not good at showing grayscaled images. Actually, it shows all white scales as only one color. the White color.Better to move to cv2.imshow() in the following code:

```
In [15]: cv2.imshow('test',threshold1)
         cv2.waitKey(0)
         cv2.destroyAllWindows()
```

To give a better example showing how useful is this technique, I will upload an image of a textual paper. The paper is not clear. See it below:

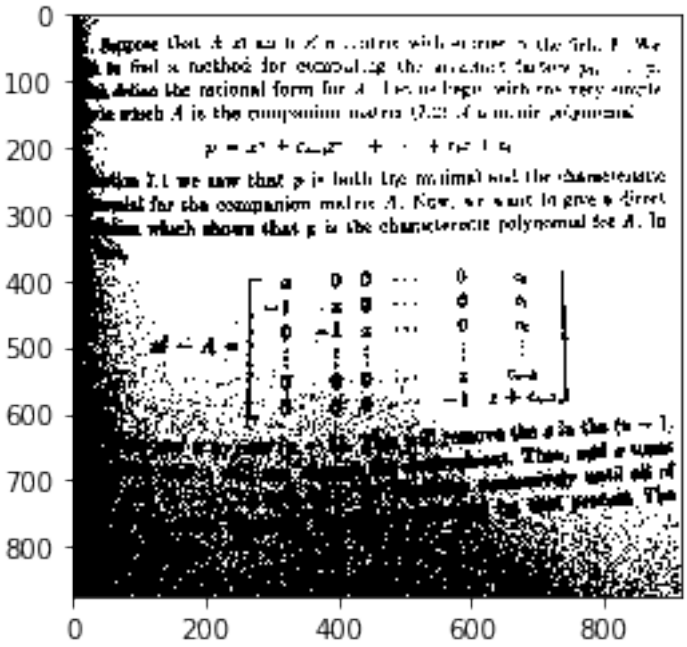
```
In [16]: paper_img = cv2.imread('paper_text.jpg')
         plt.imshow(paper_img)

Out[16]: <matplotlib.image.AxesImage at 0x7fe3baa02748>
```



Let us apply thresholding to this image.

since the image is not clear, a low threshold should be okay to distinguish textual pixels from white ones.

```
In [17]: paper_img_grayscale = cv2.cvtColor(paper_img, cv2.COLOR_BGR2GRAY)
         ret, thresholded2 = cv2.cv2.threshold(paper_img_grayscale, 30, 255, cv2.THRESH_BINARY)

In [18]: plt.imshow(thresholded2, cmap='gray')

Out[18]: <matplotlib.image.AxesImage at 0x7fe3b9b0bba8>
```



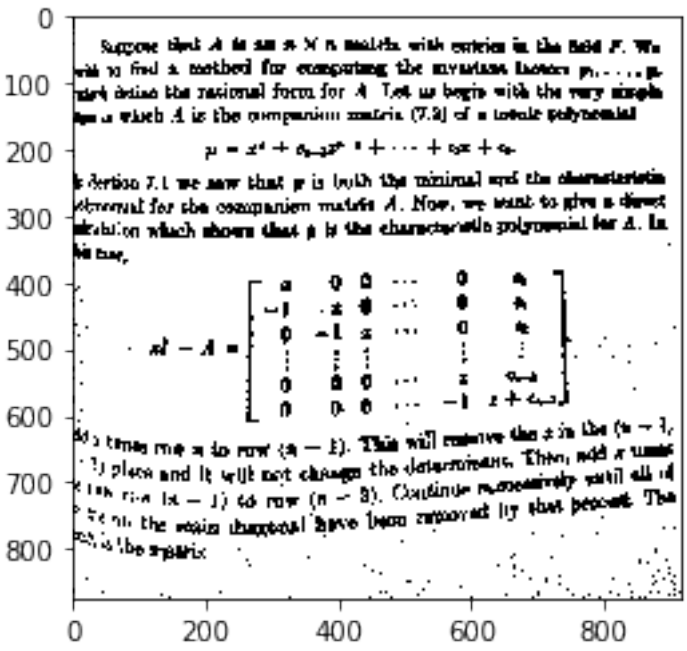
as you can see, The image is converted to black and white image where textual pixels are separated. However, there are some regions of the image where black color is destructing the text. In this case, we will use another type of thresholding explained in the next section.

1.6.2 1.5.2 adaptive thresholding

In adaptive thresholding, the algorithm check for smaller regions so that the result is optimized. In that case, the algorithm finds different thresholds for different regions of the image depending on the lightening of the area and other conditions. As its counterpart, it has two types: - cv2.ADAPTIVE_THRESH_MEAN_C - cv2.ADAPTIVE_THRESH_GAUSSIAN_C

The first type takes the mean of the selected region to be the threshold while the threshold in the second type is the weighted sum of pixels values in the selected area and weights are a gaussian window.

```
In [19]: threshold3 = cv2.adaptiveThreshold(paper_img_grayscale,
                                            255, cv2.ADAPTIVE_THRESH_MEAN_C,
                                            cv2.THRESH_BINARY, 201, 7)
         # the first parameter is the image.
         # the second parameter is the maxVal to which a pixel is converted.
         # the third paramter is the type of the adaptive thresholding.
         # the fourth paramter is the type of the simple thresholding.
         # the fifth type is the selected region size. more bigger this value more adaptive th
         # the sixth parameter is just a constant C which is subtracted from the mean or weigh
         plt.imshow(threshold3, cmap='gray')

Out[19]: <matplotlib.image.AxesImage at 0x7fe3b9af6358>
```



By looking at this image, you may notice why thresholding is useful and why I said that CamScanner app is based on this idea.

1.7 1.6 Image Arithmetics

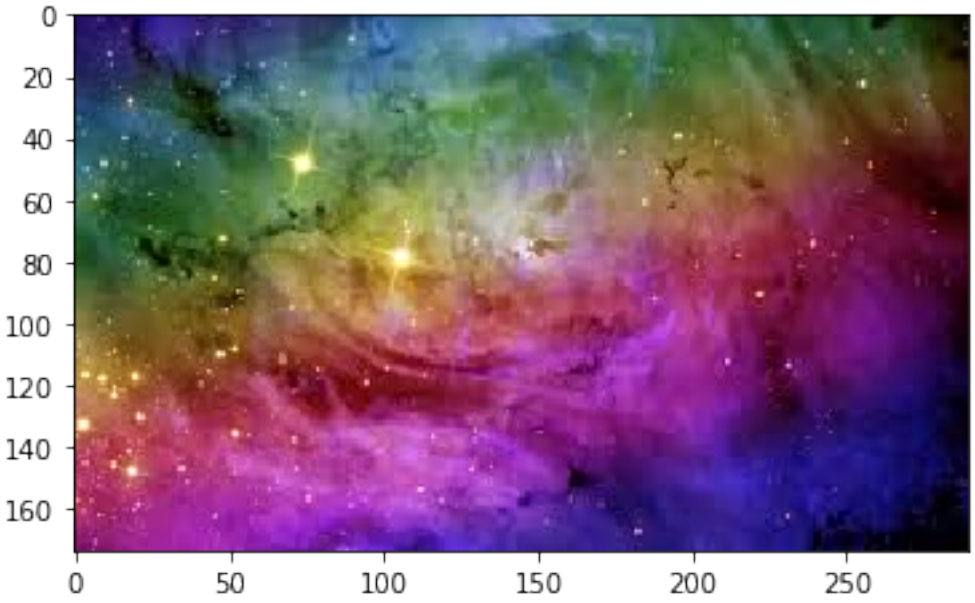
To understand this part, we need two images. For that purpose, I am going to upload another image. **Make sure that both images are of identical size**

```
In [20]: clrd_img.shape
```

```
Out[20]: (174, 290, 3)
```

```
In [21]: clrd_img2 = cv2.imread('colorful_image2.jpeg')
         plt.imshow(clrd_img2)
```

```
Out[21]: <matplotlib.image.AxesImage at 0x7fe3b9a59668>
```

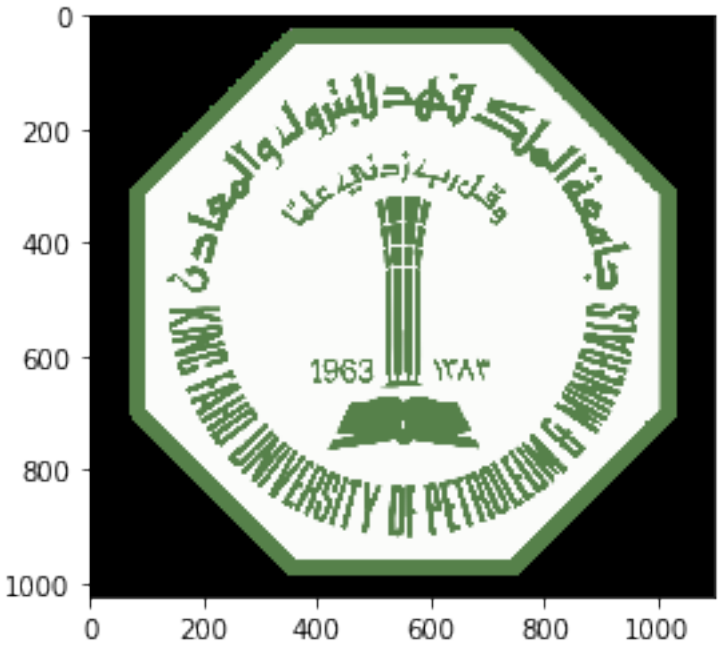


The logo, basically, is without background, you may preview it using opencv to double check. **However, cv2 adds a black background to the image!!** This is the original image: kfupm logo

Moreover, better to have a logo image. I am going to use my university logo :)

```
In [22]: kfupm_logo = cv2.imread('kfupm_logo.png')
         plt.imshow(kfupm_logo)
```

```
Out[22]: <matplotlib.figure.AxesImage at 0x7fe3b9a37f28>
```

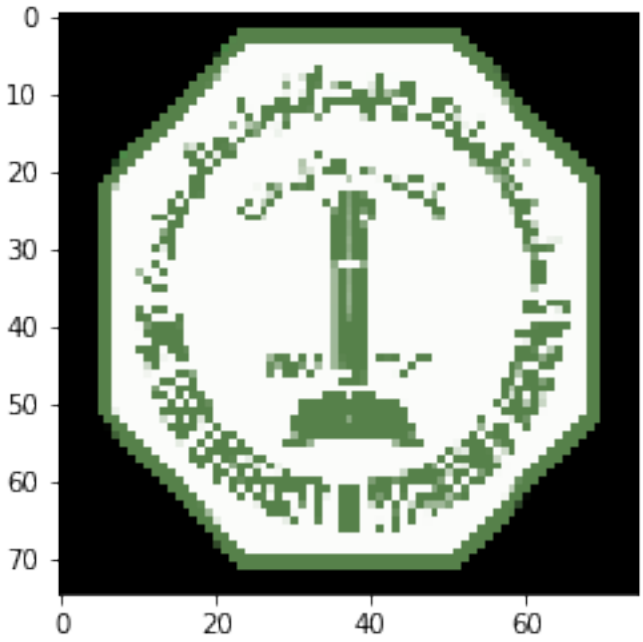


```
In [114]: cv2.imshow('kfupm logo',kfupm_logo)
          cv2.waitKey(0)
          cv2.destroyAllWindows()
```

The image is too large to be fit in another image. Better to resize it.

NOTICE THAT matplotlib plt function will not show the image properly. Better to use cv2.imshow()

```
In [23]: kfupm_logo_resized = cv2.resize(kfupm_logo,(75,75))
         plt.imshow(kfupm_logo_resized)
         # cv2.imshow('kfupm logo resized',kfupm_logo_resized)
         # cv2.waitKey(0)
         # cv2.destroyAllWindows()

Out[23]: <matplotlib.image.AxesImage at 0x7fe3b999aa58>
```

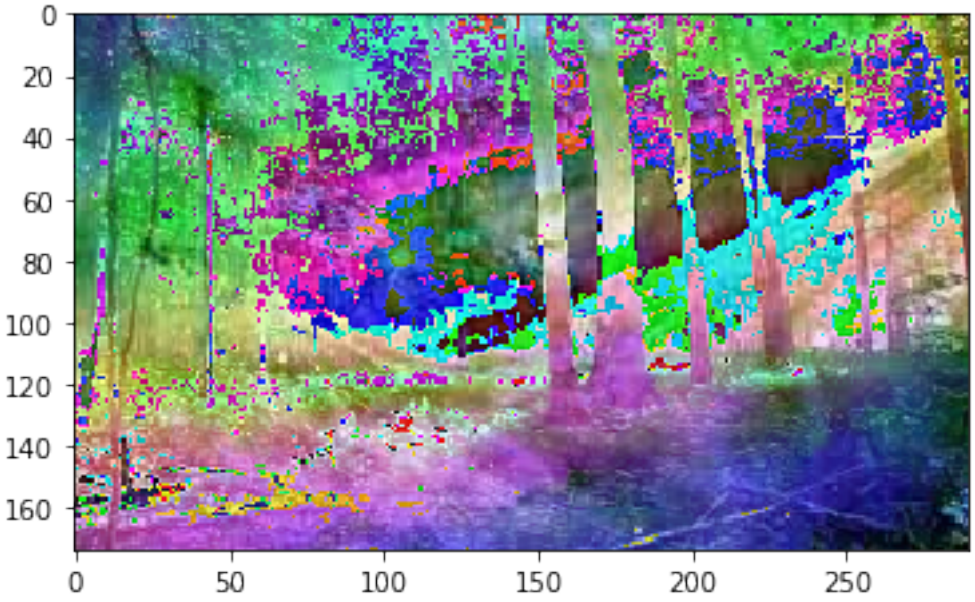


1.7.1 1.6.1 Image Addition

1.6.1.1 absolute addition We can interpret both images using only simple addition of images!! Images are implemented using numpy arrays. Therefore, you can apply this operation just as any two numpy arrays.

what do you think the image will look like?? See the it below:

```
In [24]: two_images = clrd_img+clrd_img2
         plt.imshow(two_images)

Out[24]: <matplotlib.image.AxesImage at 0x7fe3b86a3400>
```

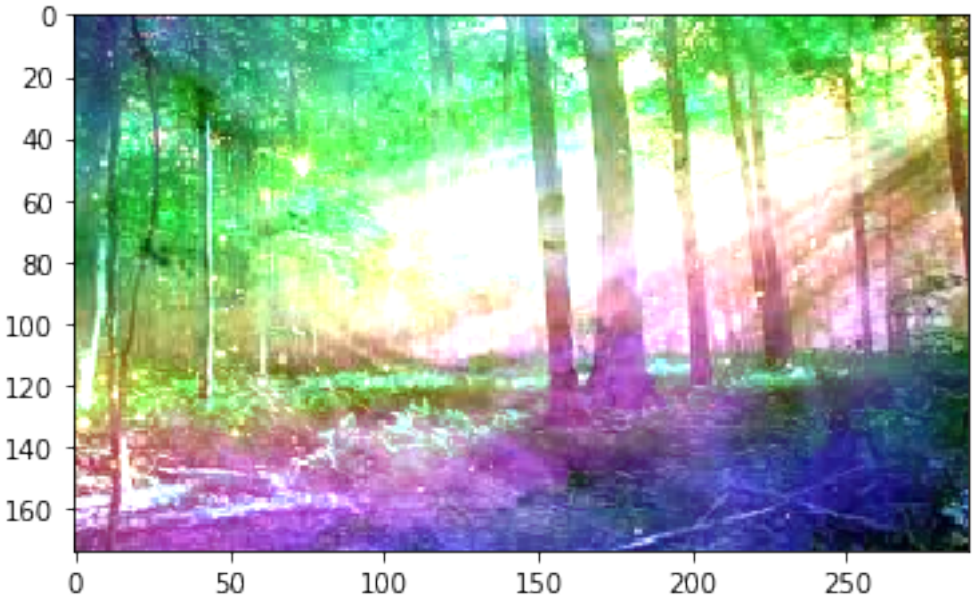


What a bad interpretaion :)) But, What happened?? This image is the result of adding all pexils of both images!! The operation is in mod 256. That is, c = a+b (mod 256) where c is a pexil in the new resuted image a,b are pexils in the images to add.

This resulting image suggests that we shall think of another way if we want to merge two images. It may work sometimes but may give horrible results just as what we get in our case!!

Another version of add is provided with opencv itself. follow the code below:

```
In [25]: two_images_cv = cv2.add(clrd_img, clrd_img2)
         plt.imshow(two_images_cv)

Out[25]: <matplotlib.image.AxesImage at 0x7fe3b867fd68>
```



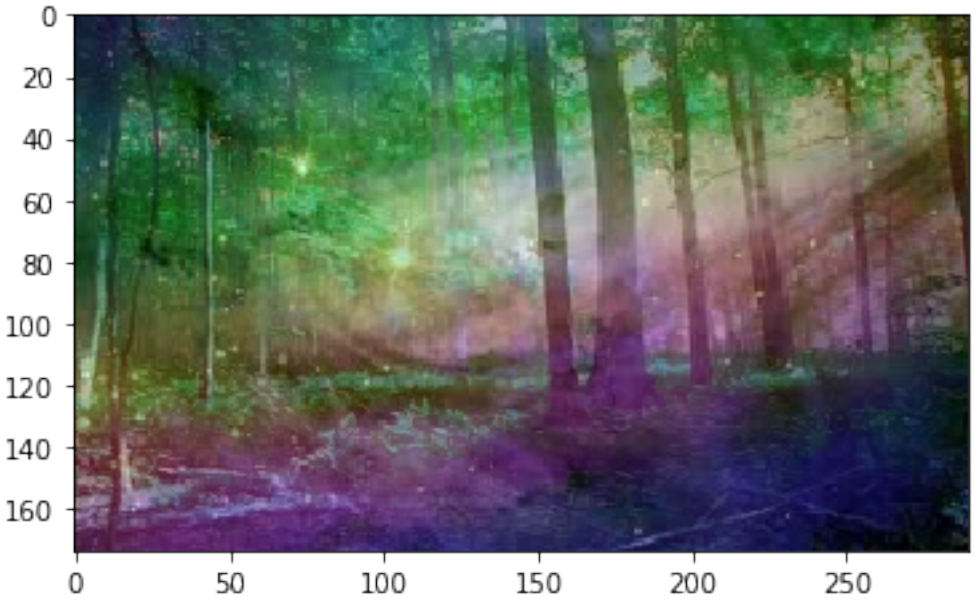
This is a better combination though. Nevertheless, what really happens?? Okay, this image is also the result of adding all pexils of both images!! But if the pexils summatoin is greater than 255, the value of the new pexil is 255.

For example, if a = 255, b = 10, which are pexils in the added images, then pexil c, which is in the resulted image is only 255!!

1.6.1.2 wieghted addition OpenCV still provides another alternative if the previous solutions did not match developers needs. This times, you can combine images based on a predetermined weight. The code is below:

```
In [26]: two_images_weighted = cv2.addWeighted(clrd_img, 0.45, clrd_img2, 0.55, 0)
         plt.imshow(two_images_weighted)
```

```
Out[26]: <matplotlib.image.AxesImage at 0x7fe3b85ed6d8>
```



As a Conclusion, there are three ways, up to what I know, to combine images, a developer may choose the best implementation fitting his needs.

1.7.2 1.6.2 Bitwise Operations

Bitwise operations is a concept well known in almost many leading programming languages. They are the critical part of conditional statements.

This is a list of them: - not: returns 1 if false and 0 otherwise. - and: returns true if both input are true. False otherwise. - or : returns false if both are false. True otherwise. - xor: returns true if input values are different. True otherwise. Python supports such operations and the following code is self explanatory.

```
In [27]: for i in range(1, 15):
             if i%2 == 0 and i%3 == 0:
                 print(i,'is divisible by 6.')
             elif i%2 == 0 or i%3 == 0:
                 print(i, 'is NOT divisible by 6, but divisible by 2 or 3')
             else:
                 print('none of the above')
```

```
none of the above
2 is NOT divisible by 6, but divisible by 2 or 3
3 is NOT divisible by 6, but divisible by 2 or 3
4 is NOT divisible by 6, but divisible by 2 or 3
none of the above
6 is divisible by 6.
none of the above
8 is NOT divisible by 6, but divisible by 2 or 3
```

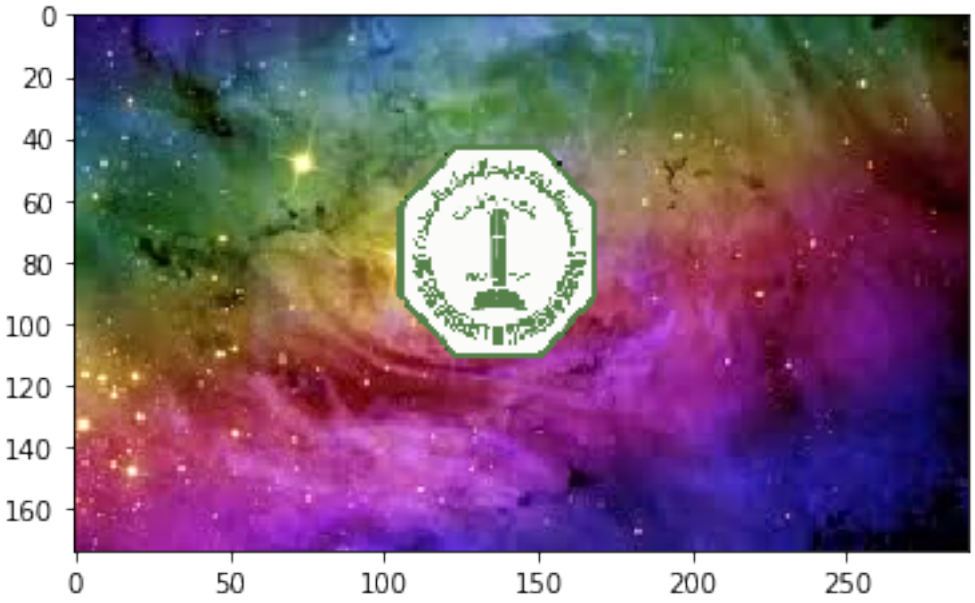
```
9 is NOT divisible by 6, but divisible by 2 or 3
10 is NOT divisible by 6, but divisible by 2 or 3
none of the above
12 is divisible by 6.
none of the above
14 is NOT divisible by 6, but divisible by 2 or 3
```

Interestingly, OpenCV supports such operations on images!! you may have a BitwiseAND operation between two images that returns 1 if two images pixels values are greater than 1 and 0 elsewhere.

This has a great usefulness when adding logos to images. Technically, if the logo is a rectangular shape, then you can add it using roi. However, most logos are non-rectangular, thus, you can add them using bit-wise operations. The following example will clarify this point. In the example, I will add the university logo to one colored image we had.

In [28]:
```
img1 = clrd_img2
img2 = kfupm_logo_resized

# I want to put logo on the center of the image, So I create a ROI
rows,cols,channels = img2.shape
roi = img1[40:40+rows, 100:100+cols]

# Now create a mask of logo and create its inverse mask also
img2gray = cv2.cvtColor(img2,cv2.COLOR_BGR2GRAY)
ret, mask = cv2.threshold(img2gray, 10, 255, cv2.THRESH_BINARY)
mask_inv = cv2.bitwise_not(mask)

# Now black-out the area of logo in ROI
img1_bg = cv2.bitwise_and(roi,roi,mask = mask_inv)

# Take only region of logo from logo image.
img2_fg = cv2.bitwise_and(img2,img2,mask = mask)

# Put logo in ROI and modify the main image
dst = cv2.add(img1_bg,img2_fg)
img1[40:40+rows, 100:100+cols] = dst

plt.imshow(img1)

## the code is taken from:
# http://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_core/py_imag
```

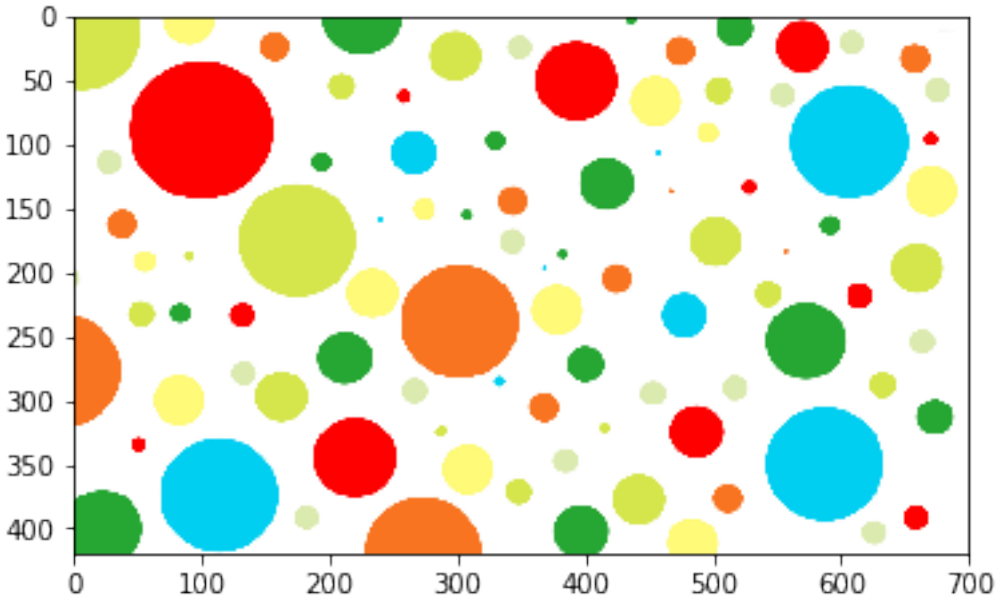
Out[28]:
```
<matplotlib.image.AxesImage at 0x7fe3b85dd438>
```



1.8 1.7 Color Filtering

In this part, we will try to identify certain colors in an image. Let us have this image:

```
In [29]: clrd_crls = cv2.imread('colored_circles.jpg')
         clrd_crls = cv2.resize(clrd_crls, (700, 420) ,fx=0.5, fy=0.5)
         # because matplotlib does wierly here, better to use this code:
         plt.imshow(cv2.cvtColor(clrd_crls.copy(), cv2.COLOR_BGR2RGB))

         # or you can view it using cv2.imshow()

         # cv2.imshow('',clrd_crls)
         # cv2.waitKey(0)
         # cv2.destroyAllWindows()

Out[29]: <matplotlib.image.AxesImage at 0x7fe3b8538e10>
```



Now, we will select only circles with blue colors and view them.

To do so, first we will convert the image to HSV (Hue, Saturation, Value) color space rather than the default color scheme 'RGB'. This scheme 'HSV" converts each color to only one value that is the value of Hue. Thus, we can retrieve this value and mask the image with areas having this color. as follows:

However, how to determine the lower value of a color and its upper value in HSL scheme?

The answer is quoted from stackoverflow:

"This is a common question found in stackoverflow.com. It is very simple and you can use the same function, cv2.cvtColor(). Instead of passing an image, you just pass the BGR values you want. For example, to find the HSV value of Green, try following commands in Python terminal:

`in[1]:green = np.uint8([[[0,255,0 ]]])  hsv_green = cv2.cvtColor(green,cv2.COLOR_BGR2HSV)  print hsv_green out[1]:[[[ 60 255 255]]]`

Now you take [H-10, 100,100] and [H+10, 255, 255] as lower bound and upper bound respectively. Apart from this method, you can use any image editing tools like GIMP or any online converters to find these values, but don't forget to adjust the HSV ranges."

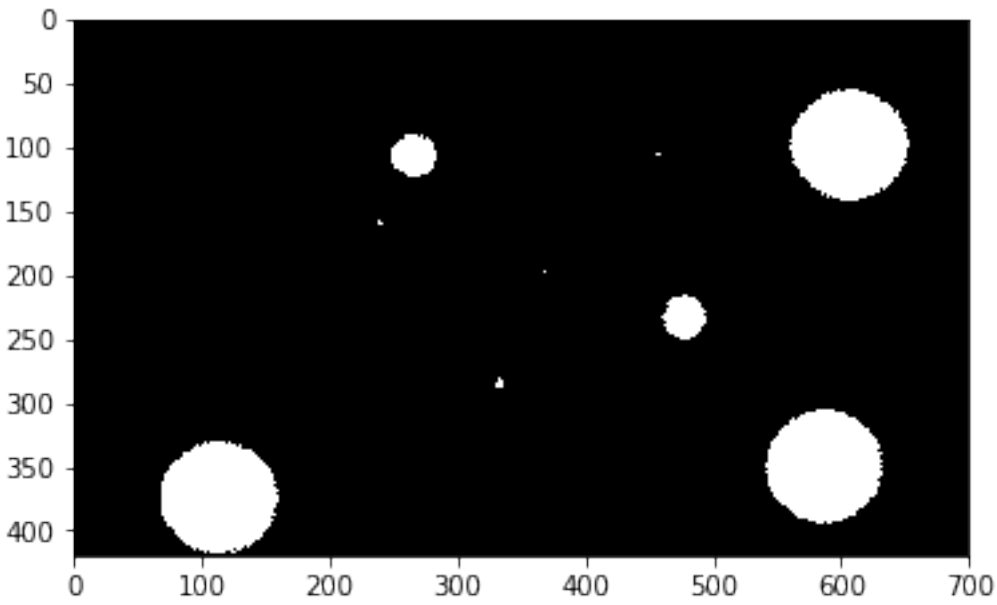
```
In [55]: # I took the value of that color by an add-on
         # REMEMBER, YOU NEED TO PUT IT IN BGR NOT RGB

         my_color = np.uint8([[[241, 207, 0]]])
         hsv_my_color = cv2.cvtColor(my_color,cv2.COLOR_BGR2HSV)
         print(hsv_my_color)
```

```
[[[ 94 255 241]]]
```

```
In [52]: clrd_crls_only_blue = clrd_crls.copy()
```

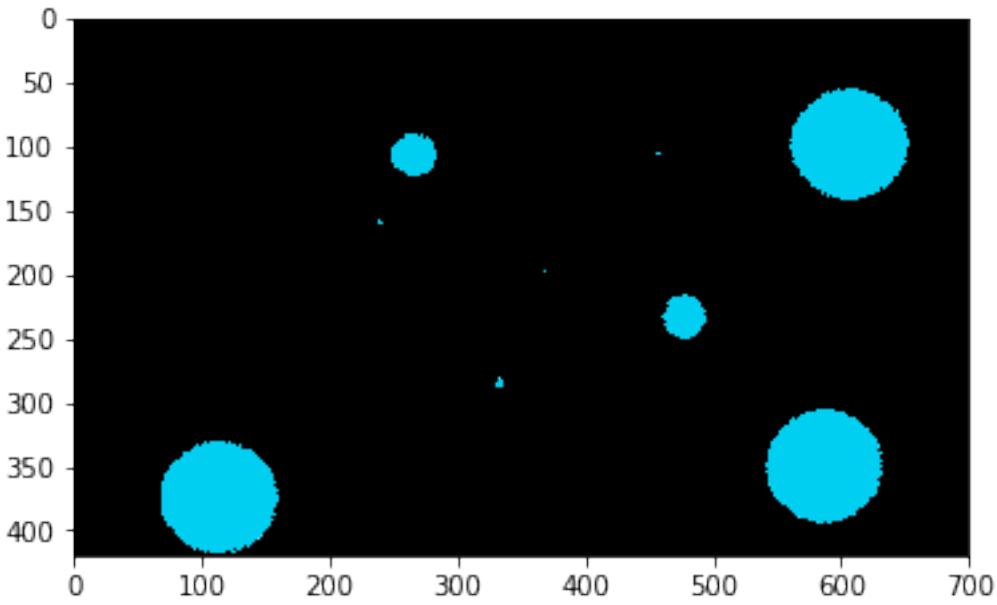
```
# convert to the hsv color map.
hsv = cv2.cvtColor(clrd_crls_only_blue, cv2.COLOR_BGR2HSV)

# extract the values from the method done above.
lower_blue = np.array([84,100,100])
upper_blue = np.array([104, 255, 241])

# the mask of the color
mask = cv2.inRange(hsv, lower_blue, upper_blue)

# the resulted image
res = cv2.bitwise_and(clrd_crls_only_blue, clrd_crls_only_blue, mask = mask)

# print the image
plt.imshow(mask, cmap='gray')
```

Out[52]: <matplotlib.image.AxesImage at 0x7fe3aa4d57b8>



In [53]: `plt.imshow(cv2.cvtColor(res.copy(), cv2.COLOR_BGR2RGB))`

Out[53]: <matplotlib.image.AxesImage at 0x7fe3b85037b8>




```
In [56]: # double check on the image appearnce:
         # cv2.imshow('mask',mask)
         # cv2.imshow('res',res)
         # cv2.waitKey(0)
         # cv2.destroyAllWindows()
```

you may, as an exercise, try to filter another image :).

1.9 1.8 Blurring and Smoothing