**Introduction to Image Processing and Computer**

**Vision**

Filip Zawadka

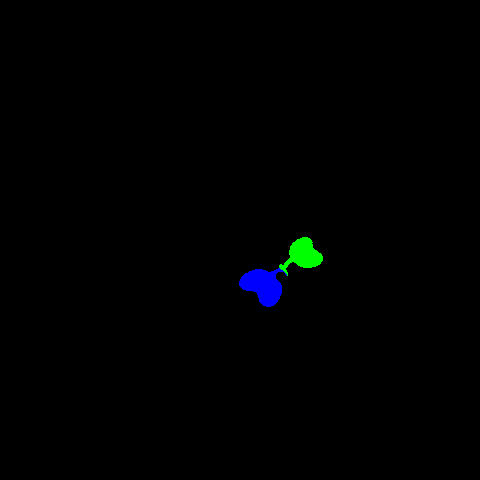
Project 1:

Plant Segmentation and Labeling

1. Introduction
2. Step by step algorithm description of
   1. Masks
   2. Colouring
3. Results assessment of
   1. Masks
   2. Colouring
4. Code
5. Final remarks and bibliography
6. **Introduction**

We are given 900 pictures documenting the growth of different plants and our goal is to colour each leaf of the given plant with a different colour. 

We are also given a file with coloured leaves which we can compare with our results.



We start with computing the masks of all the leaves. The main idea is to find an adequate hsv threshold and then clear the image from the noise.

Unfortunately clearing the mask turned out to be a bit harder then just using opening and closing after a proper threshold was selected, because there were some bigger elements of the image that were same colour as the plant.

After dealing with this problem, my idea for labelling was to use the watershed algorithm which is defined as:

„In the study of [image processing](https://en.wikipedia.org/wiki/Image_processing), a **watershed** is a transformation defined on a [grayscale](https://en.wikipedia.org/wiki/Grayscale) image. The name refers metaphorically to a geological [*watershed*](https://en.wikipedia.org/wiki/Drainage_divide), or drainage divide, which separates adjacent drainage basins. The watershed transformation treats the image it operates upon like a topographic map, with the brightness of each point representing its height, and finds the lines that run along the tops of ridges.” (wikipedia)

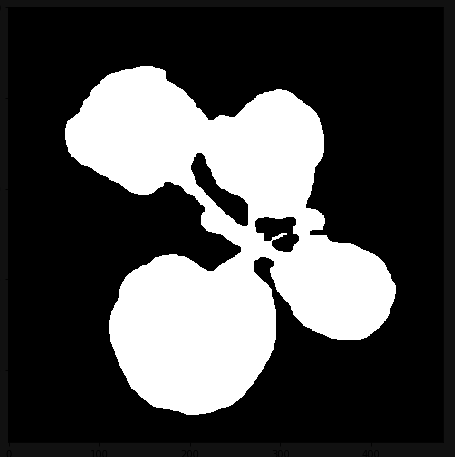
Simply, it helps us to find boundaries of objects and divide them into components even if they are connected or, in some cases, overlap each other.

For example in an image like this one:

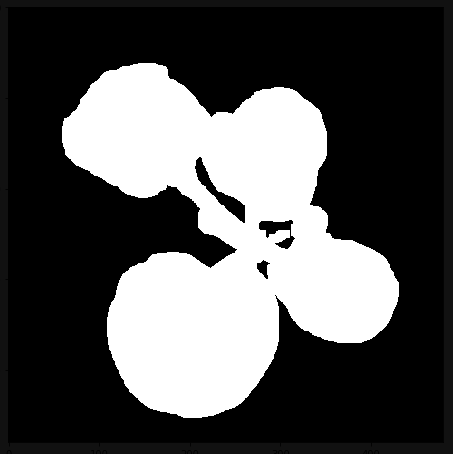
(I chose a one with leaves that touch each other)



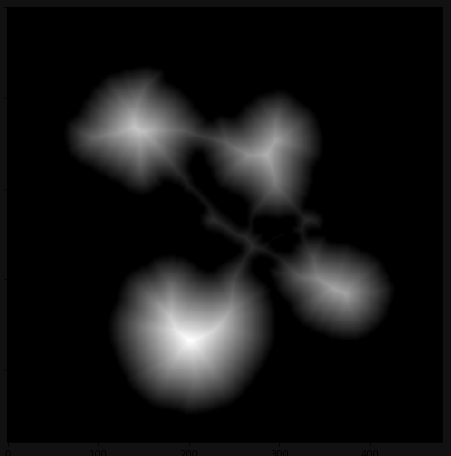
After computing a correct mask:



We can see that the white area in the image would be treated as an one object and we want to avoid this. So the first part is to use dilate to find the sure background. This will just expand the nonzero part of the image, which will leave us with a mask that is wider then our image and thus contains it.



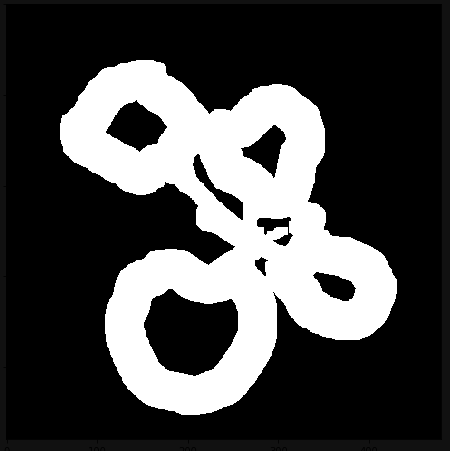
The second part is to use a distance transform algorithm, which just gives higher values to pixels that are further from the edges, so they appear brighter when the image is displayed:



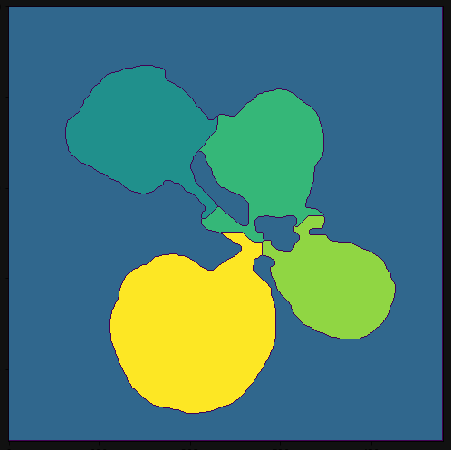
So after simple thresholding we can get the sure foreground image, which is the mask containing pixels from highest values, so the ones the furthest from the edges and thus the ones we want to use as our base for each individual component:



After that we need the unknown region, so the region between sure background and sure foreground which the watershed algorithm will add to one or the other component. So the unknown region is just sure background – sure foreground:



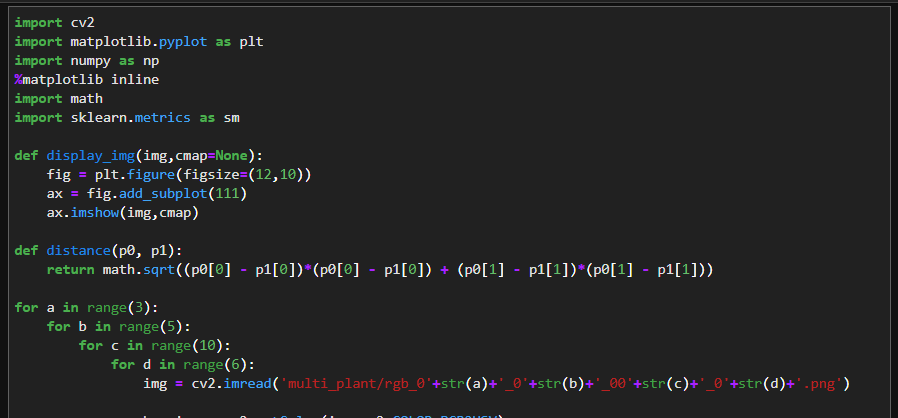
After that we mark each of the connected component and then use the watershed algorithm:



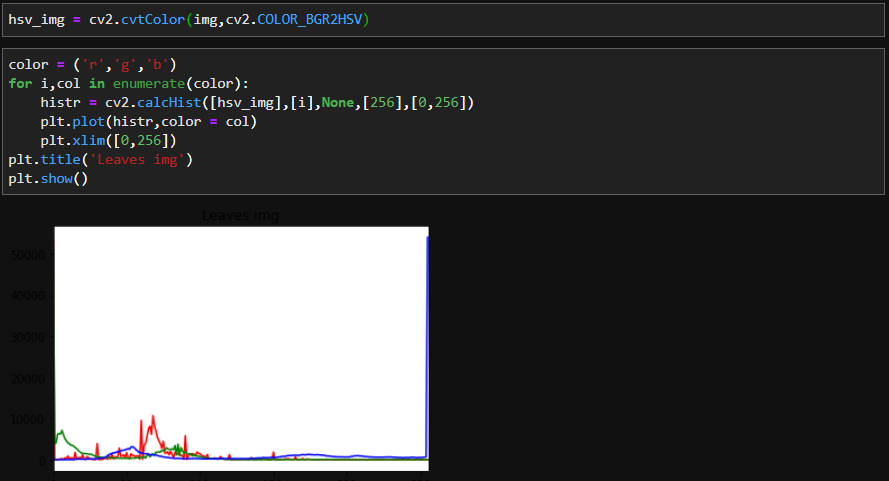
Which just leaves us with convering the image to proper shape and colours.

2.1. Step by step algorithm description of masks

First we import all of the needed libraries and define helping functions for display images and computing the distances on in the pictures. Then we read each consecutive image one by one using for loops:

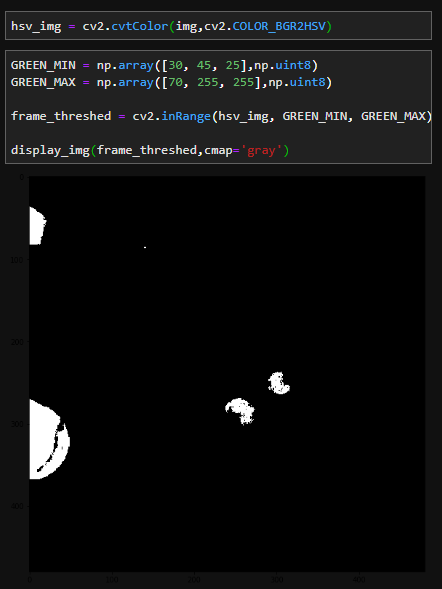
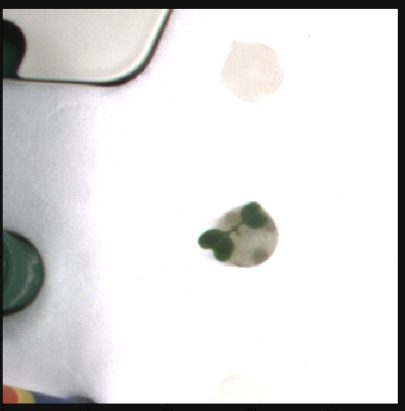


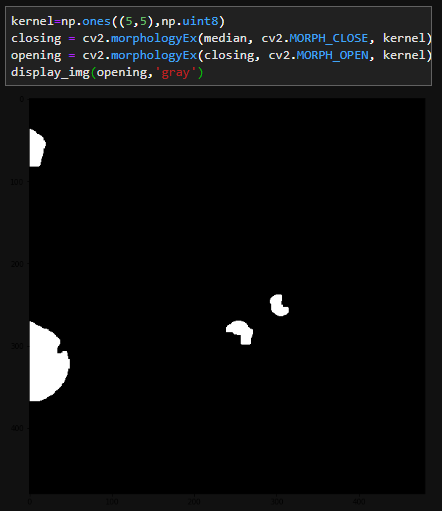
We want to find a correct hsv threshold so to help ourselves we display a histogram of one of the plants in its final stages of growth so the largest percentage of the colour coverage is from the plant:



Using the histogram and a method of trial and error we guess the mask. I will use the first image as an example.

After choosing the mask we apply it to the hsv colour coded image, and after that use a blur to simplify it and then opening and closing to get rid of the smaller noise.

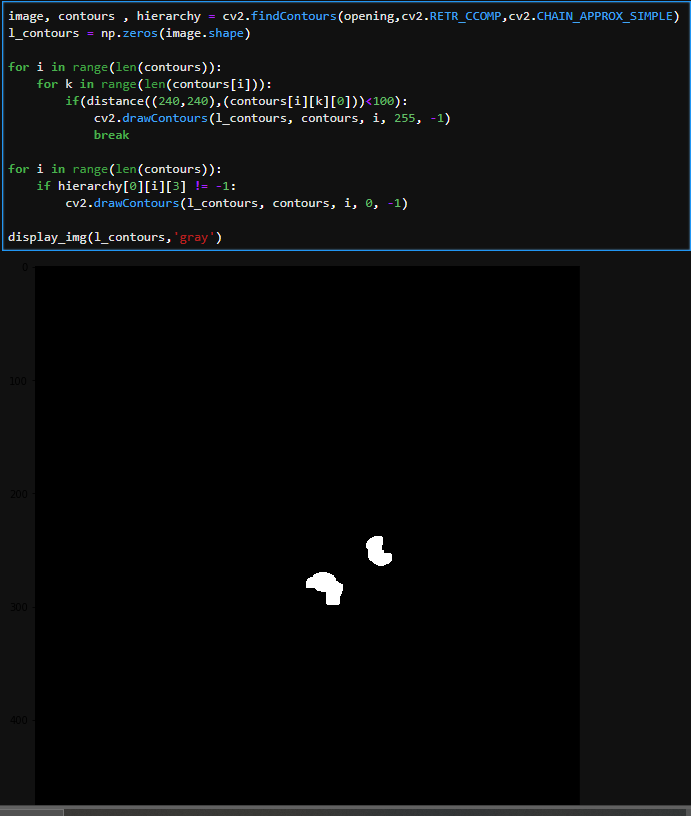




To get rid of the bigger noise we divide the image into contours and leave only those that have a point in the range of 100 from the middle so like the ones closer to the middle or crossing the red circle in this image:



Which gives our mask.

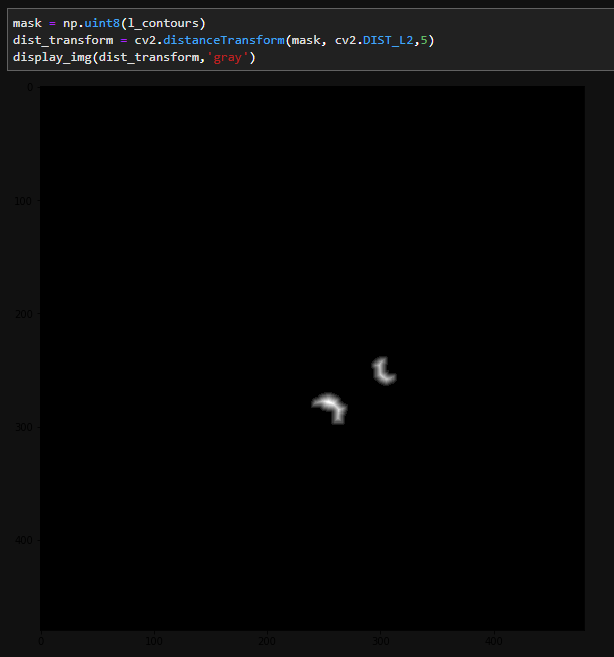


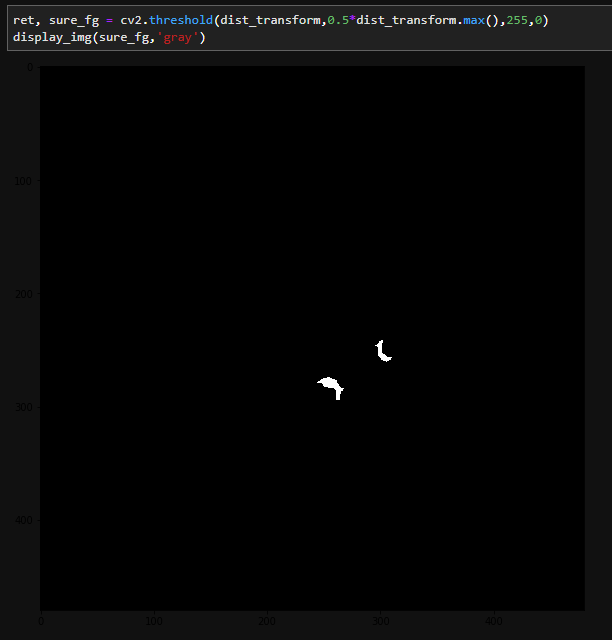
2.2. Step by step algorithm description of mask colouring

First we compute the sure background:



Next we want to find the sure foreground:





Then we search for the unknow region:



Now we start labelling. We add one to all labels so that sure background is not 0, but 1. Then we mark the region of unknown with zero.



Next we apply watershed algorithm to find markers.

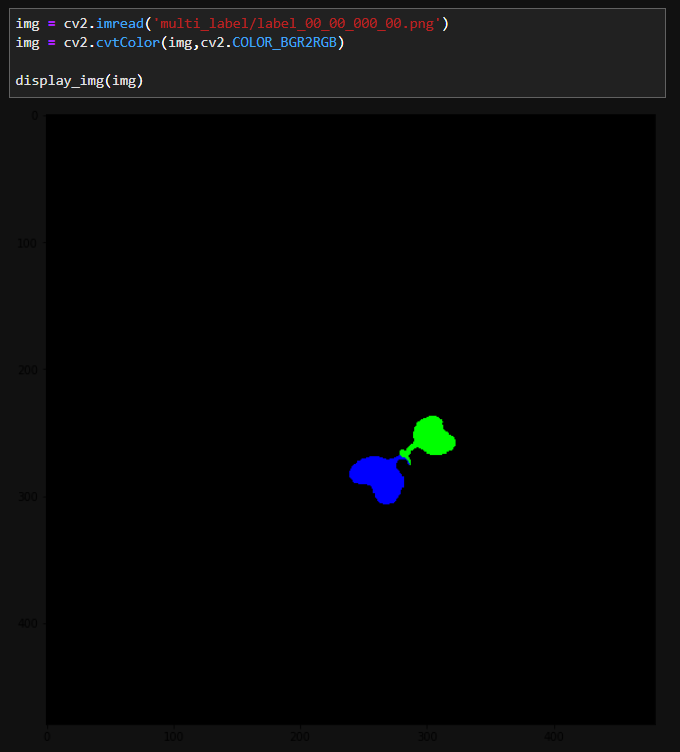


Lastly, we want to change marked image to a coloured one, with colours corresponding to the ones in the folder labelled image.

I used the image as if it was in bgr coding, so to display it in rgb we need a conversion.

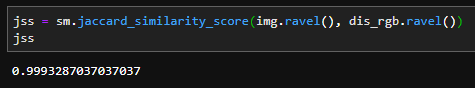


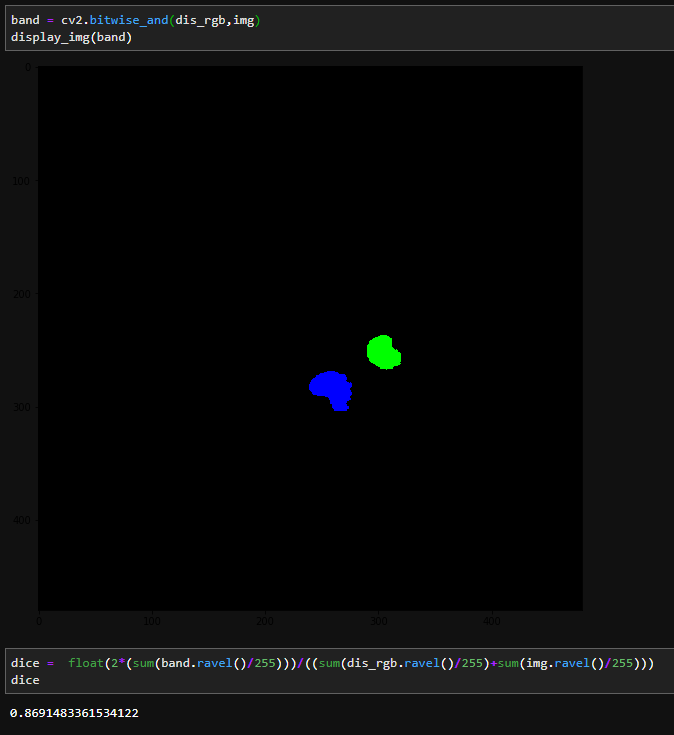
Given image to compare:



**3.Results assessment**

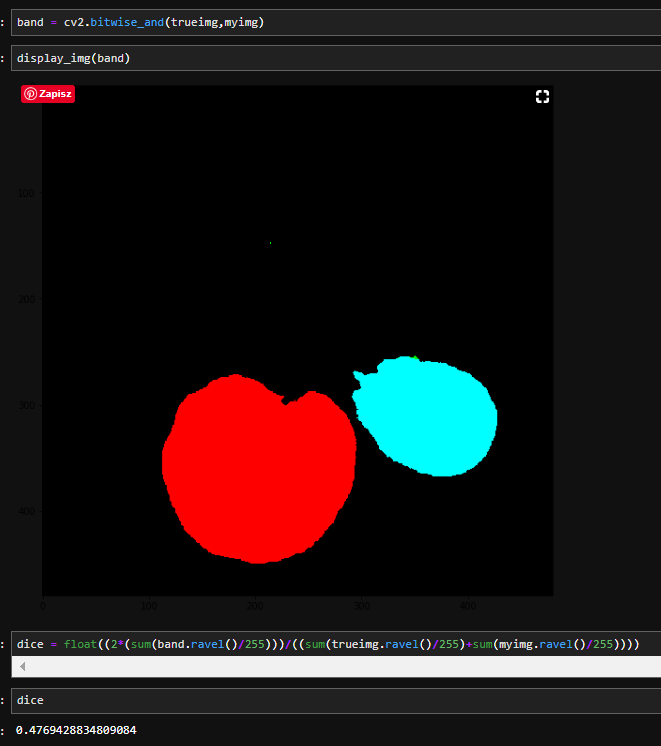
As we saw in the last paragraph our resulting image is very similar to the one we were aiming to achieve so the error came out small





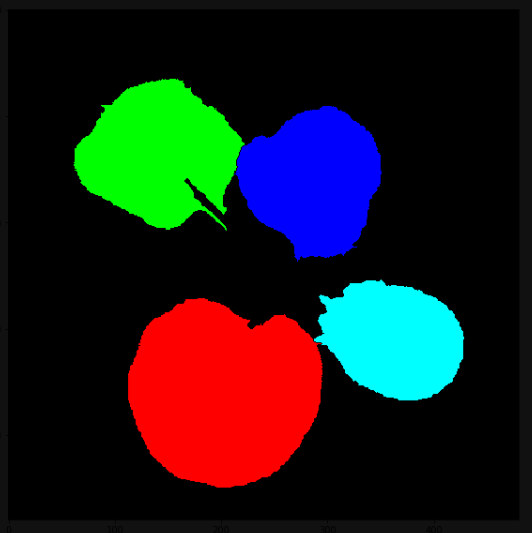
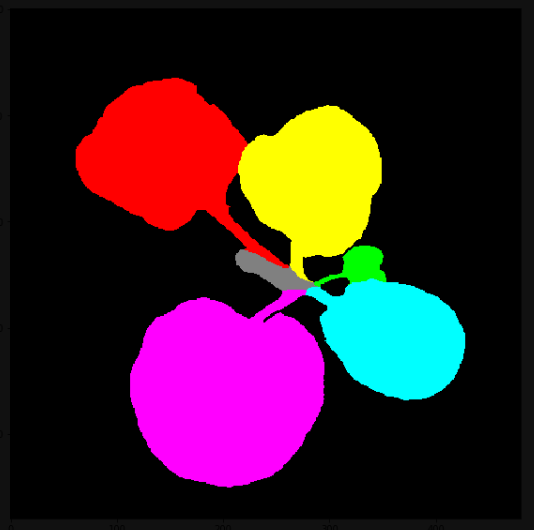
Unfortunately the mean value of all the similarity scores isn’t that good. The Jaccard Similarity Score is equal to 0.9419510979295274, but the Dice Similarity Score is equal to only 0.37539793779928937.

The reason for this might be explained using this example. We calculate the similarity score for the last stage of growth of the same plant:



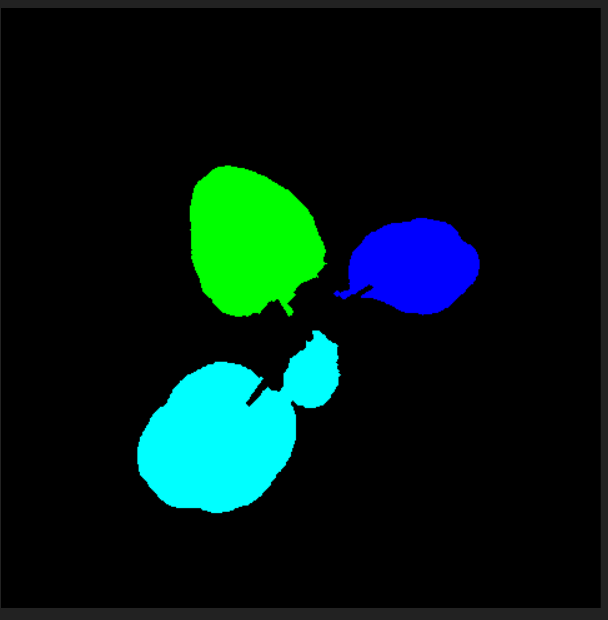
BitwiseAND matrix

We can see that bitwiseAND matrix (here band) only shows two leaves. If we look at our coloured image and the given one :

While the leaves are drawn pretty accuratly the colours don’t match and the band matrix shows only the ones which are of the same colour or a have the same compound one. Here we see the light blue and red, because it’s a component of pink.

Another problem that accured was that some leaves that were growing too near to eachother were treated as one and threrefore coloured with the same colour as in the picture below.



**4. Code**

# Filip Zawadka 290569

# Project 1

# Plant Segmentation and Labeling

import cv2

import matplotlib.pyplot as plt

import numpy as np

%matplotlib inline #only for my compiler

import math

import sklearn.metrics as sm

def display\_img(img,cmap=None):

    fig = plt.figure(figsize=(12,10))

    ax = fig.add\_subplot(111)

    ax.imshow(img,cmap)

def distance(p0, p1):

    return math.sqrt((p0[0] - p1[0])\*(p0[0] - p1[0]) + (p0[1] - p1[1])\*(p0[1] - p1[1]))

for a in range(3):

    for b in range(5):

        for c in range(10):

            for d in range(6):

                img = cv2.imread('multi\_plant/rgb\_0'+str(a)+'\_0'+str(b)+'\_00'+str(c)+'\_0'+str(d)+'.png')

                hsv\_img = cv2.cvtColor(img,cv2.COLOR\_BGR2HSV)

                GREEN\_MIN = np.array([55, 45, 25],np.uint8)

                GREEN\_MAX = np.array([75, 255, 255],np.uint8)

                frame\_threshed = cv2.inRange(hsv\_img, GREEN\_MIN, GREEN\_MAX)

                kernel=np.ones((5,5),np.uint8)

                median = cv2.medianBlur(frame\_threshed,5)

                closing = cv2.morphologyEx(median, cv2.MORPH\_CLOSE, kernel)

                opening = cv2.morphologyEx(closing, cv2.MORPH\_OPEN, kernel)

                image, contours , hierarchy = cv2.findContours(opening,cv2.RETR\_CCOMP,cv2.CHAIN\_APPROX\_SIMPLE)

                # Set up empty array

                l\_contours = np.zeros(image.shape)

                # For every entry in contours

                for i in range(len(contours)):

                    for k in range(len(contours[i])):

                        if(distance((240,240),(contours[i][k][0]))<100):

                            cv2.drawContours(l\_contours, contours, i, 255, -1)

                            break

 #Now draw all of the inside contours in case of overlaping leaves

                for i in range(len(contours)):

                    if hierarchy[0][i][3] != -1:

                        cv2.drawContours(l\_contours, contours, i, 0, -1)

                mask = np.uint8(l\_contours)

                cv2.imwrite ('mask/rgb\_0'+str(a)+'\_0'+str(b)+'\_00'+str(c)+'\_0'+str(d)+'.png',mask)

                sure\_bg = cv2.dilate(l\_contours,kernel,iterations=3)

                dist\_transform = cv2.distanceTransform(mask, cv2.DIST\_L2,5)

                ret, sure\_fg = cv2.threshold(dist\_transform,0.5\*dist\_transform.max(),255,0)

                sure\_fg = np.uint8(sure\_fg)

                sure\_bg = np.uint8(sure\_bg)

                unknown = cv2.subtract(sure\_bg,sure\_fg)

                # Marker labelling

                ret, markers = cv2.connectedComponents(sure\_fg)

                # Add one to all labels so that sure background is not 0, but 1

                markers = markers+1

                # Now, mark the region of unknown with zero

                markers[unknown==255] = 0

                markers = cv2.watershed(img,markers)

                #Change the markers to bgr image like in the multi\_label folder

                mask\_rgb = np.zeros((480,480,3),np.uint8)

                for i in range(480):

                    for k in range(480):

                        if markers[i][k] == 2:#g-green

                            mask\_rgb[i][k][1]=255

                        elif markers[i][k] == 3:#b-blue

                            mask\_rgb[i][k][0]=255

                        elif markers[i][k] == 4:#gb-light blue/cyan

                            mask\_rgb[i][k][1]=255

                            mask\_rgb[i][k][0]=255

                        elif markers[i][k] == 5:#r-red

                            mask\_rgb[i][k][2]=255

                        elif markers[i][k] == 6:#rb-violet

                            mask\_rgb[i][k][0]=255

                            mask\_rgb[i][k][2]=255

                        elif markers[i][k] == 7:#rg-yellow

                            mask\_rgb[i][k][2]=255

                            mask\_rgb[i][k][1]=255

                        elif markers[i][k] == 8:#rgb-white

                            mask\_rgb[i][k][0]=255

                            mask\_rgb[i][k][1]=255

                            mask\_rgb[i][k][2]=255

                cv2.imwrite ('rgb\_mask/rgb\_0'+str(a)+'\_0'+str(b)+'\_00'+str(c)+'\_0'+str(d)+'.png',mask\_rgb)

#Computing the Similarity Scores

jss=0 #Mean Jaccard Similarity Score

dice=0 #Mean Sice Similarity Score

dice = float(dice)

for a in range(3):

    for b in range(5):

        for c in range(10):

            for d in range(6):

                img = cv2.imread('multi\_label/label\_0'+str(a)+'\_0'+str(b)+'\_00'+str(c)+'\_0'+str(d)+'.png')

                myimg = cv2.imread('rgb\_mask/rgb\_0'+str(a)+'\_0'+str(b)+'\_00'+str(c)+'\_0'+str(d)+'.png')

                #resize to make sure they are of the correct size

                img.resize((480,480,3))

                myimg.resize((480,480,3))

                jss = jss + (sm.jaccard\_similarity\_score(img.ravel(), myimg.ravel()))

                band = cv2.bitwise\_and(myimg,img)

                dice = dice + float(2\*(sum(band.ravel()/255)))/((sum(img.ravel()/255)+sum(myimg.ravel()/255)))

jss = jss/900

dice = float(dice)/900

print(jss)

print(dice)

**5.Final remarks and bibliography**

My main source of knowledge was “[Python for Computer Vision with OpenCV and Deep Learning” course on  Udemy](https://www.udemy.com/course/python-for-computer-vision-with-opencv-and-deep-learning/learn/lecture/12257458?components=buy_button%2Cdiscount_expiration%2Cgift_this_course%2Cintroduction_asset%2Cpurchase%2Cdeal_badge%2Credeem_coupon#overview) i also used their example on the „coins” image, and I worked with jupiter notebook to compute and display step results as well as the full code.

Some additionals links that were helpful:

<https://docs.scipy.org/doc/numpy/user/quickstart.html>

<https://en.wikipedia.org/wiki/HSL_and_HSV>

<https://en.wikipedia.org/wiki/Gaussian_blur>

<https://www.tutorialspoint.com/dip/concept_of_blurring.htm>

<https://en.wikipedia.org/wiki/Thresholding_(image_processing)>

<https://en.wikipedia.org/wiki/Mathematical_morphology>

<https://homepages.inf.ed.ac.uk/rbf/HIPR2/morops.htm>

<https://en.wikipedia.org/wiki/Watershed_(image_processing)>