Adedayo Adeyemi:317296

Introduction to Image Processing and Computer Vision

Leaf Segmentation

Table of Contents

1. Introduction

Image segmentation is the aspect of computer vision that deals with the partitioning of an image into smaller multiple segments or group of pixels that are assigned to specific labels; the pixels having the same labels share certain characteristics. Image segmentation is important cause it can be used to detect edge, line, and boundaries within an image to aid further image analysis and instance or object detection/description.

1.1 problem definition

1.1.1 Task description

In this project, I will try to segment images of leaves on white background. The input will be images of leave s to be segmented and the expected output are the correctly segmented images. Below are sample images of the leaves that will be used in this project.

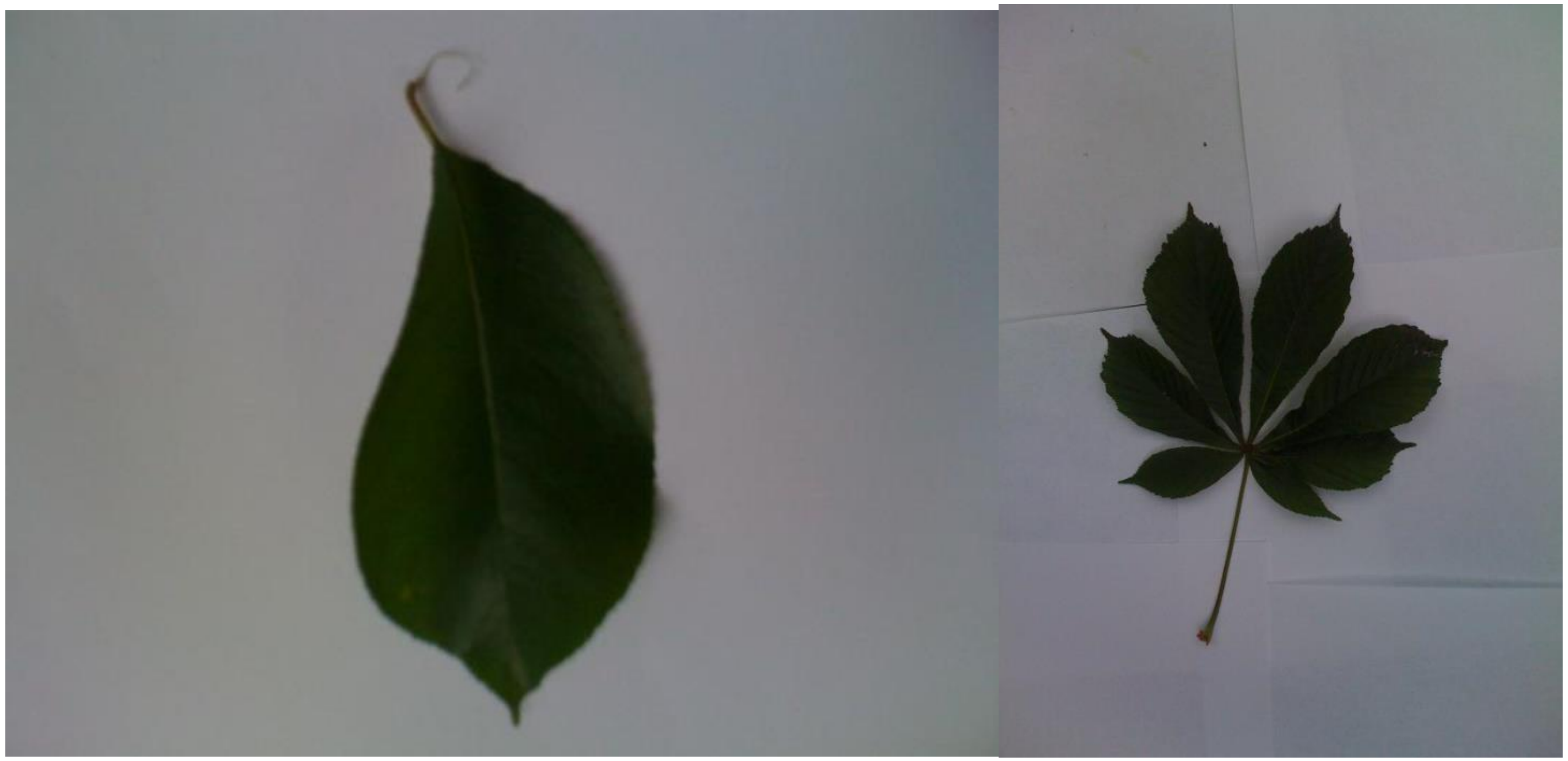


Figure 1: Sample leaf

1.1.2 Dataset description

The dataset is a publicly available dataset: Leafsnap Field dataset. There are 300 images available of which 150 images already produced good result segmentation using state of the art segmentation methods while the remaining 150 were not successfully segmented.
The leaves contained in the data set have different shapes, color intensity and texture as seen in the image above Figure 1: Sample leaf.

2. Literature Review

According to Buoncompagni, Maio and Lepetit, shape is the main element for leaves recognition due to the different color and partial or absence of textures with respect to leaves belonging to the same species (Buoncompagni, Maio, & Lepetit, 2015). Therefore, segmentation of leaves is very important in the process of leaves recognition.
One of the major challenges of leaf segmentation is the pixel level precision needed in differentiating global leaves shapes and highlighting leaves boundary structures.

According to them, other methods such as image binarization with a fixed threshold and methods based on mean shift / K means clustering have been used for leaf segmentation in supervised and unsupervised settings respectively with good results. However, they stated that the recent applications of leaf recognition in loose settings which uses the maximization algorithm do not work well with images having shadows and light reflections. Therefore, they propose a mew method of training pixel wise classifiers that learns to filter according to the foreground and background regions of leaves images then using the EM algorithm for segmenting the images (Buoncompagni, Maio, & Lepetit, 2015).

3. Solution Description

Since the leaves in the dataset are green and the background white, the mask of the images was obtained by removing everything that is not green from the image.

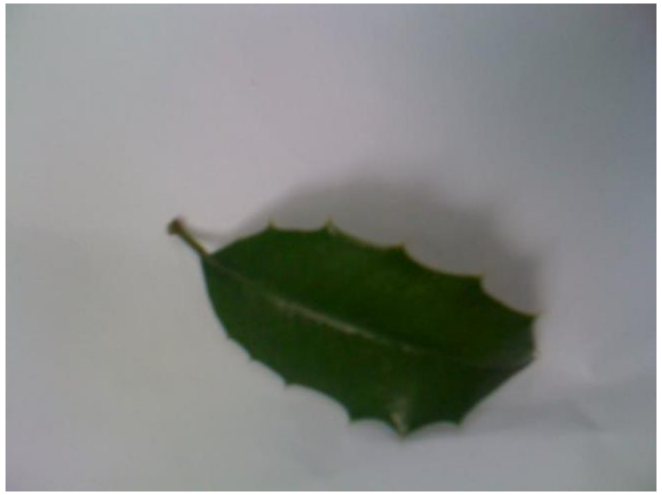


Figure 2:sample leaf

In selecting the shades of green, the RGB image was converted to HSV and multiple trials were made in order to choose the right shade of green manually and with the help of a color shades generator (https://mdigi.tools/color-shades/#808080). The generator gives the percentage of the HSV in this format : hsv(0, 0%, 2%). In order to get the numpy array format, I converted it manually by calculating the number that gives the percentage using the Hue range [0,179], Saturation range is the Saturation range in HSV which is [0,255] and the value range is the Value range in HSV which is [0,255].

```
# remove all colors except of particular shades of green
hsv = cv2.cvtColor(image , cv2.COLOR_BGR2HSV)
lower_green = np.array ([30 , 30, 10])
upper_green = np.array ([110, 255 , 130])
mask = cv2.inRange(hsv , lower_green , upper_green)
```

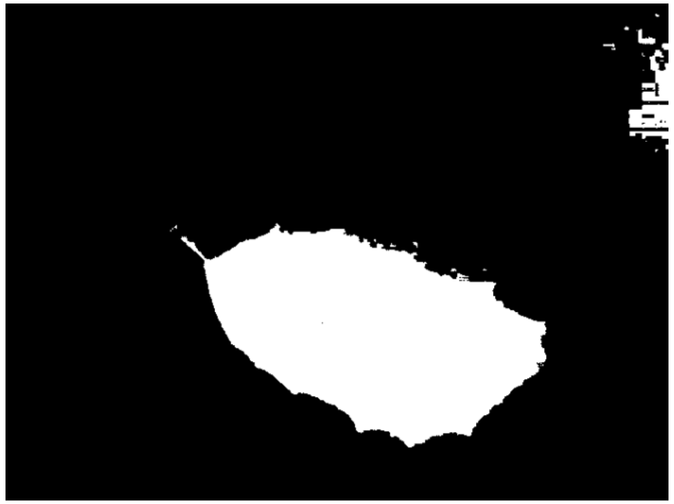


Figure 3:Leaf mask after applying HSV

I use median blur to remove the background noise from the image then apply thresholding to get the binary image for contouring.

```
#denoise blur image
    mask = cv2.medianBlur(mask, 7)

    #threshold the image
    ret,thresh = cv2.threshold(mask, 127, 255, 0, cv2.THRESH_BINARY)
```

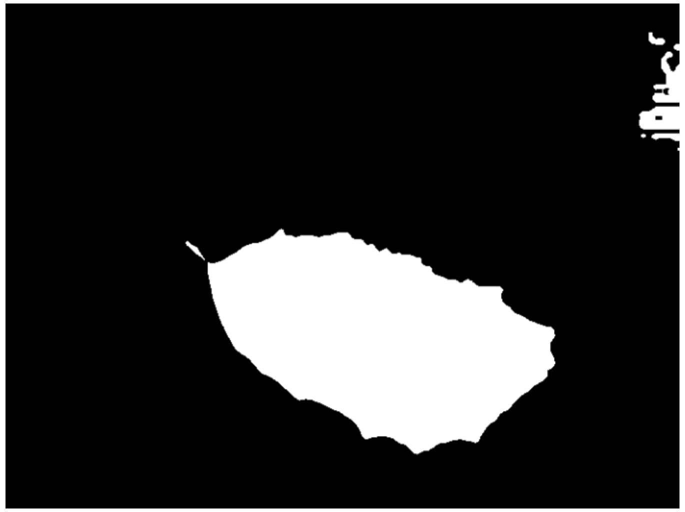


Figure 4: leaf after removing noise and thresholding

I performed morphological transformations of closing on the image to reduce the holes (background colors) in the foreground(leaf) region of the image. While the erosion and dilation of the closed image removes the stem of the leaves.

```
# construct a closing kernel and apply it to the thresholded image
    kernel = cv2.getStructuringElement(cv2.MORPH_RECT, (21, 7))
    closed = cv2.morphologyEx(thresh, cv2.MORPH_CLOSE, kernel)

    # perform a series of erosions and dilations
    closed_eroded= cv2.erode(closed, None, iterations = 4)
    closed_eroded= cv2.dilate(closed_eroded, None, iterations = 4)
```

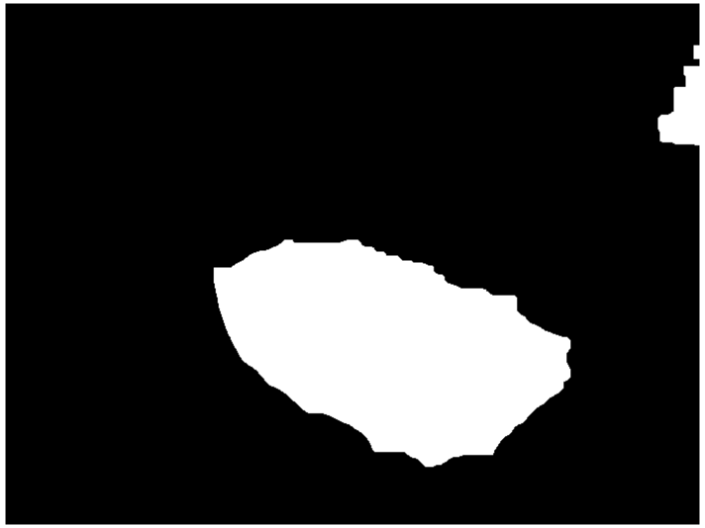


Figure 5: Leaf after morphological transformations

Then I detect the contour in the image and choose the one that is in the center of the image in order to get the leaf part that we are interested in. If there is only one contour detected then there is no for the center of the image detection.

```
# find the contours in the thresholded image, then sort the contours
    # by their area, keeping only the largest one
    countour_image = cv2.findContours(closed_eroded.copy(), cv2.RETR_EXTERNAL,cv2.CHAIN_APPROX_SIMPLE)
    countour_image  = countour_image[0]
    if len(countour_image) == 1:
      c = sorted(countour_image, key = cv2.contourArea, reverse = True)[0]
    else:
      min=image.shape[0]
      for i in range(len(countour_image)):
          if cv2.contourArea(countour_image[i]) > 700:
              Moments = cv2.moments(countour_image[i])
              cont_X = int(Moments["m10"] / M["m00"])
              cont_Y = int(Moments["m01"] / M["m00"])
              Z = sqrt((cont_X - image.shape[1]/2)**2 + (cont_Y - image.shape[0]/2)**2)
              if min > Z:
                  min=Z
```

```
            c=countour_image[i]
```

Finally, a rotated bounding box is computed and drawn around the segmented image.

```
# compute the rotated bounding box of the contour
rect = cv2.minAreaRect(c)
box = cv2.boxPoints(rect)
box = np.int0(box)
# draw a bounding box arounded the detected leaf and display the
   # image
   box_image = image.copy()
   cv2.drawContours(box_image, [box], -1, (0, 255, 0), 3)
   return box_image,mask
```

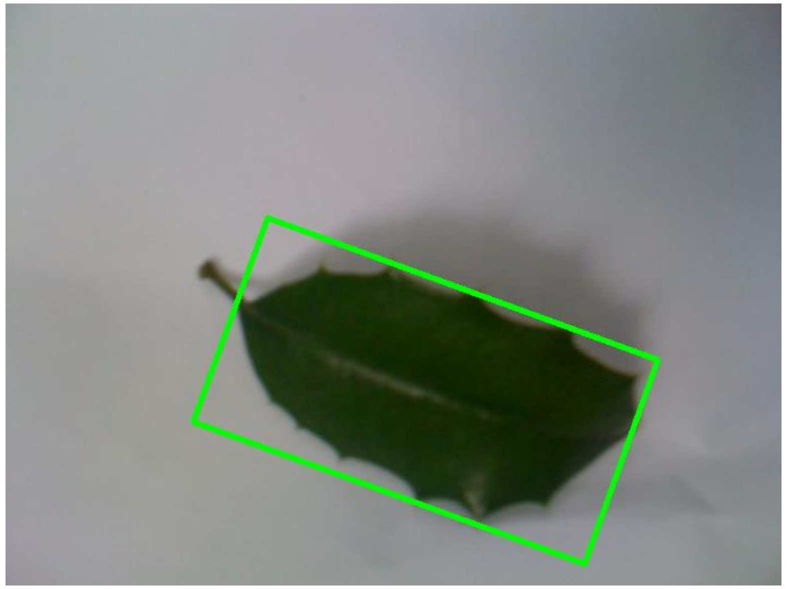


Figure 6: Bounding Box

4. Results

Mean IoU of 75% and Mean dice of 81% was achieved from the leaf dataset 1
Mean IoU of 56% and Mean dice of 64% was achieved from the leaf dataset 2

Max IoU of 99% and Mean dice of 99% was achieved from the leaf dataset 1
Max IoU of 99% and Mean dice of 99% was achieved from the leaf dataset 2

For the leave dataset 1 below are the results:

```
********** Jaccard index ***************


Mean IoU: 0.7506748232056236
Min IoU: 13001283354584.png - 0.0021333333333333334
Max IoU: 13292245855389.png - 0.9970032108455227

 ********** Dice coefficient ***************

Mean Dice: 0.811180431041741
Min Dice: 13001283354584.png - 0.004257583821181479
Max Dice: 13292245855389.png - 0.9984993568672288
```

Good result: good lighning and mask and bounding box

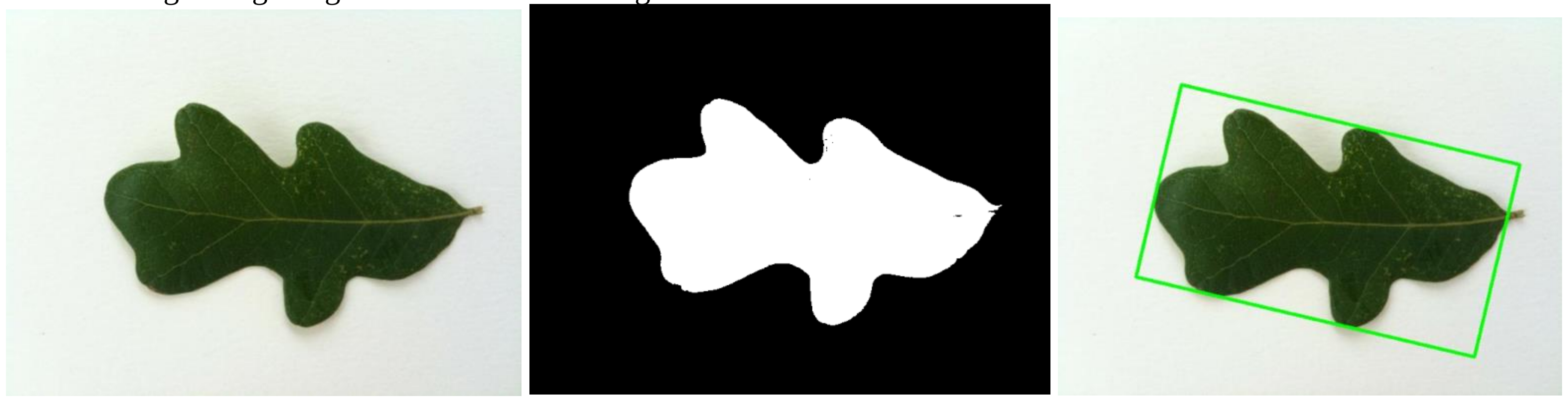


Figure 7: Good result leaf set 1

Bad result:

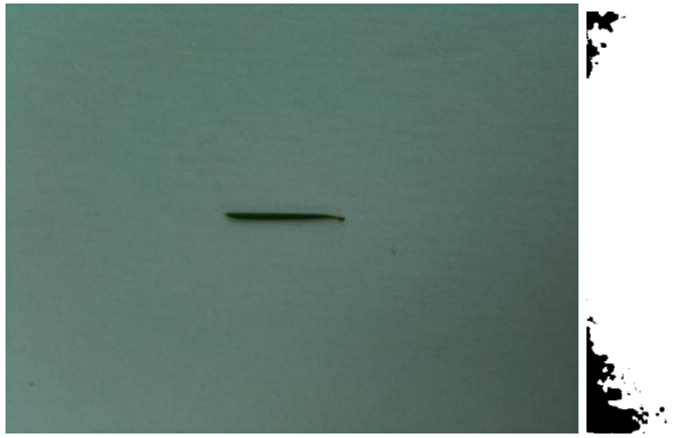
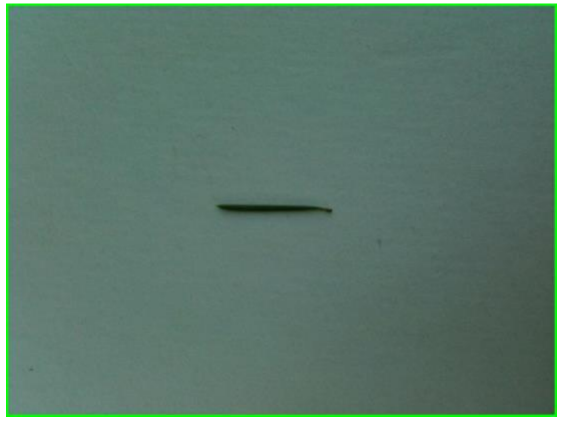


Figure 8:: Bad result leaf set 1



For the leave dataset 2 below are the results:

```
********** Jaccard index ***************

Mean IoU: 0.5686498491876589
Min IoU: 13002223738081.png - 0.0016201055467394915
Max IoU: 13291804059300.png - 0.9907632212186936

********** Dice coefficient ***************

Mean Dice: 0.6438509268692695
Min Dice: 13002223738081.png - 0.0032349701004756658
Max Dice: 13291804059300.png - 0.9953601821236924
```

Example of good segmentation: With 99% accuracy. Here it is noticed that the image has good lightning, no shadows therefore it is easy to segment.

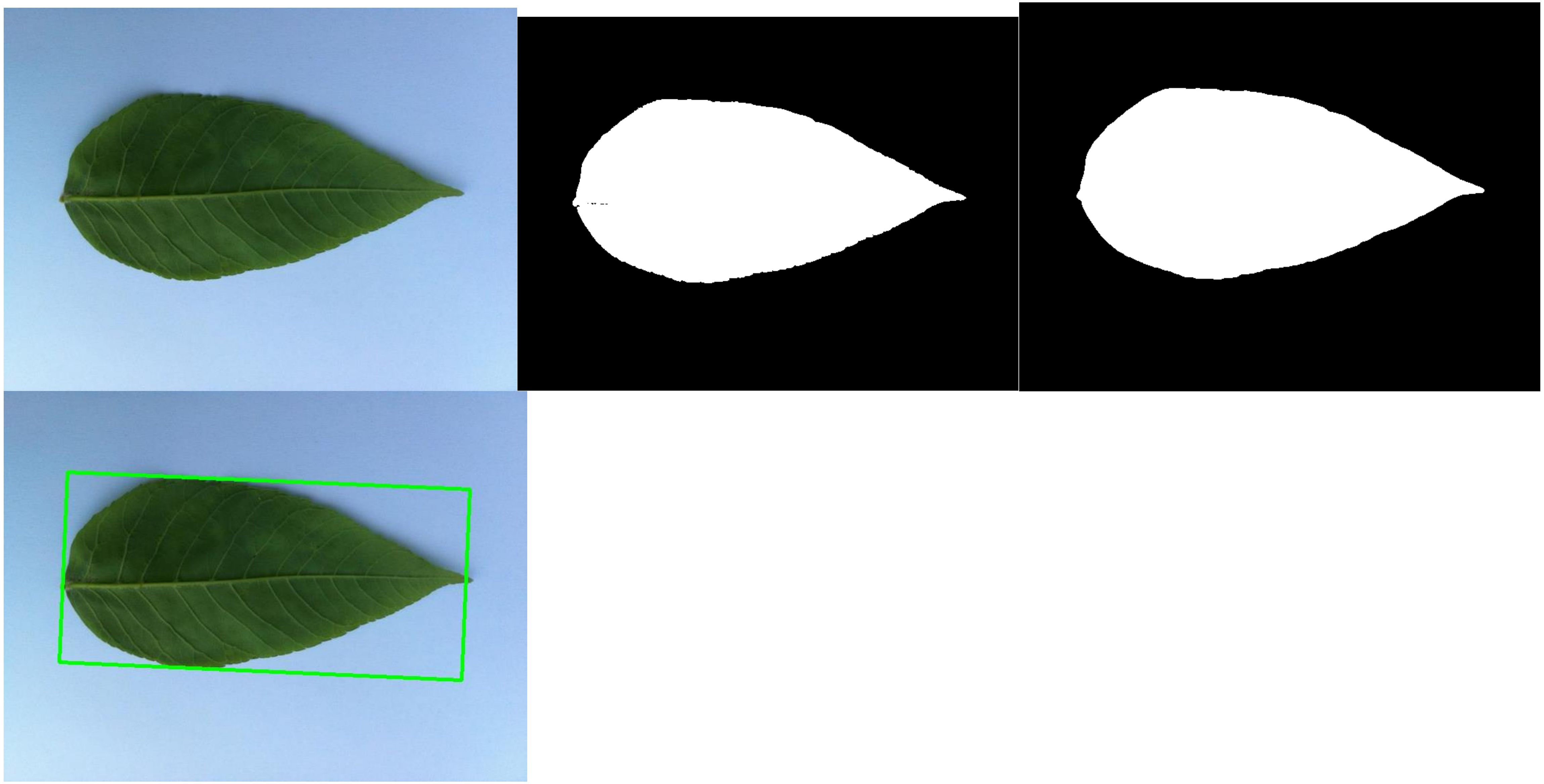


Figure 9:: Good example leaf set 2

Sample of bad result: as we can see the leaf is very little and bad lightning, therefore it was ad to segment.

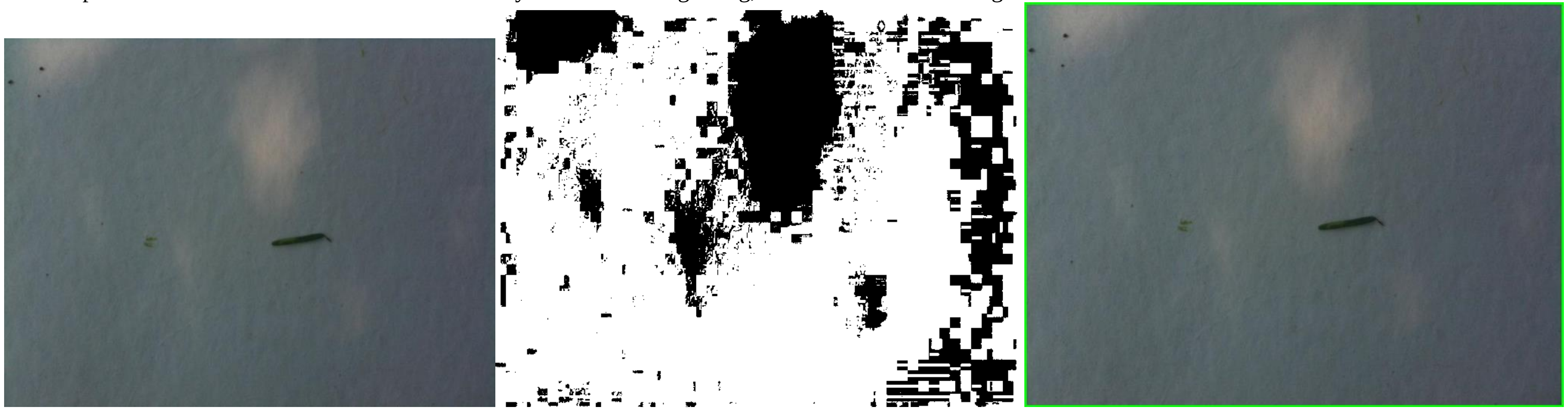


Figure 10: Bad example leaf set 2

5. CONCLUSION:

In conclusion the results were good for leaves that did not have stems, are visible and have good lightning. However, the results were bad for leaves with bad lightning, not visible leaves and leaves with greenish background since the mask was created based on the green image.

6. Source code:

```
# import the necessary packages
import glob
import numpy as np
import cv2
import os
from math import sqrt


def save_image(save_path,image, name,):
    """Function for saving the image"""
    name = os.path.basename(name)
    cv2.imwrite(save_path + '/' + name, image)

def create_mask_and_box(path):
    """Fuction to create mask and bounding box"""
      # load the image
    image = cv2.imread(path,cv2.IMREAD_COLOR)

    # remove all colors except of particular shades of green
    hsv = cv2.cvtColor(image , cv2.COLOR_BGR2HSV)
    lower_green = np.array ([30 , 30, 10])
    upper_green = np.array ([110, 255 , 130])
    mask = cv2.inRange(hsv , lower_green , upper_green)

    #denoise blur image
    mask = cv2.medianBlur(mask, 7)

    #threshold the image
    ret,thresh = cv2.threshold(mask, 127, 255, 0, cv2.THRESH_BINARY)

    # construct a closing kernel and apply it to the thresholded image
    kernel = cv2.getStructuringElement(cv2.MORPH_RECT, (21, 7))
    closed = cv2.morphologyEx(thresh, cv2.MORPH_CLOSE, kernel)
```

```
# perform a series of erosions and dilations
closed_eroded= cv2.erode(closed, None, iterations = 4)
closed_eroded= cv2.dilate(closed_eroded, None, iterations = 4)

# find the contours in the thresholded image, then sort the contours
# by their area, keeping only the largest one
countour_image = cv2.findContours(closed_eroded.copy(), cv2.RETR_EXTERNAL,cv2.CHAIN_APPROX_SIMPLE)
countour_image  = countour_image[0]
if len(countour_image) == 1:
  c = sorted(countour_image, key = cv2.contourArea, reverse = True)[0]
else:
  min=image.shape[0]
  for i in range(len(countour_image)):
      if cv2.contourArea(countour_image[i]) > 500:
          Moments = cv2.moments(countour_image[i])
          cont_X = int(Moments["m10"] / Moments["m00"])
          cont_Y = int(Moments["m01"] / Moments ["m00"])
          Z = sqrt((cont_X - image.shape[1]/2)**2 + (cont_Y - image.shape[0]/2)**2)
          if min > Z:
              min=Z
              c = countour_image[i]



rect_image = cv2.minAreaRect(c)
box = cv2.boxPoints(rect_image)
box = np.int0(box)


# draw a bounding box arounded the detected leaf and display the
# image
box_image = image.copy()
cv2.drawContours(box_image, [box], -1, (0, 255, 0), 3)
return box_image,mask
```

```
def read_write_path():
    """ Function for creating and saving masks and bounding boxes for each leaf image"""
    read_leaf_path = 'drive/MyDrive/Colab Notebooks/leaf_dataset/leaves_testing_set_1/color_images/'
    write_leaf_masks_path = 'drive/MyDrive/Colab Notebooks/leaf_dataset/solution/masks'
    write_leaf_box_path = 'drive/MyDrive/Colab Notebooks/leaf_dataset/solution/boxes'

    files = [f for f in glob.glob(read_leaf_path + "**/*.png", recursive=True)]
    for f in files:
        box_image,mask_image = create_mask_and_box(f)
        save_image(write_leaf_masks_path,mask_image, f)
        save_image(write_leaf_box_path,box_image, f.replace('rgb', 'box'))
        name = os.path.basename(f)

    print("All files saved")
    cv2.waitKey()




def check_result():
    """Function for chcecking if our predicted masks is similar to sample masks"""
    print("************** checking masks ****************")

    sample_mask_image_path= 'drive/MyDrive/Colab Notebooks/leaf_dataset/leaves_testing_set_1/ground_truth'
    my_leaf_masks_path = 'drive/MyDrive/Colab Notebooks/leaf_dataset/solution/masks'

    masks_files = [f for f in glob.glob(sample_mask_image_path + "**/*.png", recursive=True)]
    mean_IoU = 0
    mean_Dice = 0
    min_IoU = 1
```

```
max_IoU = 0
min_Dice = 1
max_Dice = 0
results = []

for f in masks_files:
    name = os.path.basename(f)
    my_mask = cv2.imread(my_leaf_masks_path + '/' + name)[:, :, 0]
    sample_mask = cv2.imread(sample_mask_image_path + '/' + name)[:, :, 0]

    predicted_mask_binary = my_mask.flatten().astype(np.bool)
    true_mask_binary = sample_mask.flatten().astype(np.bool)

    overlap = np.logical_and(true_mask_binary, predicted_mask_binary)
    union = np.logical_or(true_mask_binary, predicted_mask_binary)
    IOU = overlap.sum() / float(union.sum())

    Dice = (2.0 * overlap.sum()) / (
        np.sum(predicted_mask_binary) + np.sum(true_mask_binary)
    )
    results.append((name, IOU, Dice))
    #print(name + ' :  IoU: ' + str(IOU) + '   Dice: ' + str(Dice))
    mean_IoU += IOU
    mean_Dice += Dice
    if IOU > max_IoU:
        max_IoU = IOU
        nameMaxIoU = os.path.basename(f)
    if IOU < min_IoU:
        min_IoU = IOU
        nameMinIoU = os.path.basename(f)
    if Dice > max_Dice:
        max_Dice = Dice
        nameMaxDice = os.path.basename(f)
    if Dice < min_Dice:
```

```python
            min_Dice = Dice
            nameMinDice = os.path.basename(f)

    mean_IoU /= len(masks_files)
    mean_Dice /= len(masks_files)
    print("\n ********** Jaccard index *************** \n")
    print("Mean IoU: " + str(mean_IoU))
    print("Min IoU: " + nameMinIoU + ' - ' + str(min_IoU))
    print("Max IoU: " + nameMaxIoU + ' - ' + str(max_IoU))
    print("\n ********** Dice coefficient *************** \n")
    print("Mean Dice: " + str(mean_Dice))
    print("Min Dice: " + nameMinDice + ' - ' + str(min_Dice))
    print("Max Dice: " + nameMaxDice + ' - ' + str(max_Dice) + "\n")
    with open('results.txt', 'w') as f:
        f.write("\n ************** checking masks ***************** \n ")
        f.write(" Filename:        IoU per leaf score:        IoU score:      Dice per leaf score:        Dice score:\n")
        for item in results:
            f.write("%s\n" % str(item))
        f.write("\n%s\n\n" % "\n ********** Jaccard index *************** \n")
        f.write("%s" % "Mean IoU: " + str(mean_IoU) + "\n")
        f.write("%s" % "Min IoU: " + nameMinIoU + ' - ' + str(min_IoU) + "\n")
        f.write("%s" % "Max IoU: " + nameMaxIoU + ' - ' + str(max_IoU) + "\n")
        f.write("\n%s\n\n" % "\n ********** Dice coefficient *************** \n")
        f.write("%s" % "Mean Dice: " + str(mean_Dice) + "\n")
        f.write("%s" % "Min Dice: " + nameMinDice + ' - ' + str(min_Dice) + "\n")
        f.write("%s" % "Max Dice: " + nameMaxDice + ' - ' + str(max_Dice) + "\n")
    cv2.waitKey()



def main():
    print("Computer Vision project 1 - Leaf")

    #Main code:
```

```
    read_write_path()
    check_result()
    print("Thank you. Goodbye")




main()
```

Works Cited

Buoncompagni, S., Maio, D., & Lepetit, V. (2015). Leaf Segmentation under Loosely Controlled Conditions. In M. W. Xianghua Xie (Ed.), *British Machine Vision Conference 2015* (pp. 133.1-133.12). BMVA Press.

Rosebrock, A. (2016, February 1). *OpenCV center of contour*. Retrieved December, 2021 from pyimagesearch: https://www.pyimagesearch.com/2016/02/01/opencv-center-of-contour/

https://learnopencv.com/simple-background-estimation-in-videos-using-opencv-c-python/