**Supplement 1: Python Image processing script for vegetation cover**

Emmett Snyder <emmett.snyder@mail.utoronto.ca>

October 2019

This script calculates total percent cover of greenhouse containers or trays.

It won't work with circular pots, which would require the use of Hough circle detection.

My brief experience with this has taught me that circle detection with opencv is a messy business

and should be avoided at all costs.

The workflow for this script is:

1. open image as color

2. duplicate image; set red and blue channels to 0

3. convert green channel image to grayscale

4. threshold using Otsu's algorithm; manually adjust if necessary

5. Detect rectangle (container)

6. Apply mask to image

7. Count number of pixels in masked image

8. Count number of white pixels in masked image

10. Save thresholded image

12. Write image paths and total cover to spreadsheet

"""

from os import path, listdir

import cv2 as cv

import numpy as np

import csv

openCv\_version = cv.\_\_version\_\_

class IMG(object):

"""An object to wrap up all the thresholding functions Assumes that we are dealing with an RGB image/"""

def \_\_init\_\_(self, imagefp,

#reference

outputTS): """validate image

file imagefp: pointer to image file

outputTS: pointer to output file (thresholded image)"""

#validate that file exists

if not path.exists(imagefp):

raise IOError("File <%(fp)s> DNE" %{'fp': imagefp})

#if not path.exists(reference):

# raise IOError("File <%(fp)s> DNE" %{'fp': referenc})

self.ORIG = cv.imread(imagefp, cv.IMREAD\_COLOR)

self.IMAGE = self.ORIG.copy()#read image as color

self.IMAGE =cv.cvtColor(self.IMAGE, cv.COLOR\_BGR2HSV)

cv.namedWindow("Green", cv.WINDOW\_NORMAL)

cv.resizeWindow("Green", 750, 750)

cv.namedWindow("Mask", cv.WINDOW\_NORMAL)

cv.resizeWindow("Mask", 750, 750)

cv.namedWindow("All", cv.WINDOW\_NORMAL)

cv.resizeWindow("All", 750, 750)

#cv.namedWindow("con", cv.WINDOW\_NORMAL)

#cv.resizeWindow("con", 750, 750)

otsuval, grn, grn2 = self.getGreenChannel()

#con = self.findContainer(self.ORIG)

#cv.imshow("con", con)

#cv.imshow("Green", grn)

#cv.imshow("Mask", grn2)

#cv.imshow("All", self.ORIG)

#cv.waitKey(0)

#cv.imwrite(outputTS, grn)

#print(grn)

cc = self.get\_cc(grn)

self.row = [otsuval, cc[0], cc[1], cc[2], imagefp, outputTS, path.split(imagefp)[-1]] #otsu threshold, image size, white pixels, cover, output thresholded image

#cv.waitKey(0)

#cv.destroyAllWindows()

#self.findContainer(self.ORIG)

#ts = self.threshold()

#self.thresholded = ts["image"]

#self.thresholdLevel = ts["threshold"]

#self.total = self.countPixels()

#self.white = self.countWhite()

def getGreenChannel(self):

"""Convert the red and blue channels to 0"""

grn = self.IMAGE.copy()

mask = cv.inRange(grn, (0, 25,25), (90, 255, 255))#90-100 seem to do the trick

#mask = cv.GaussianBlur(mask, (11,11), 25)

newMask = mask > 0

grn = np.zeros\_like(self.IMAGE, np.uint8)

grn[newMask] = self.IMAGE[newMask]

hsv\_channels = np.uint8

#grn = cv.cvtColor(grn, cv.COLOR\_BGR2GRAY)

h, s, v = cv.split(grn) #split hsv into channels

h = cv.GaussianBlur(v, (5,5), 5)

ret,binary = cv.threshold(h, 0, 255, cv.THRESH\_BINARY)#+cv.THRESH\_OTSU)#use otsu's algorithm

return(ret, binary, grn)

def findContainer(self, image):

"""Find the container borders. See: https://blog.ayoungprogrammer.com/2013/04/tutorial-detecting-multiple-rectangles.html/

image: image to find container borders

workflow:

1. smooth with gaussian blur to remove noise

2. Threshold image

gray = cv.GaussianBlur(image, (51,51), 1000)

gray = cv.cvtColor(gray, cv.COLOR\_BGR2GRAY)

ret, ts = cv.threshold(gray, 75, 100, cv.THRESH\_BINARY\_INV)

\_, contours, hierarch = cv.findContours(ts, cv.RETR\_LIST ,cv.CHAIN\_APPROX\_SIMPLE)

cv.drawContours(image,contours, -1, (0,0,255),-1)

return(image)

#ret1, contours, blah =cv.findContours(ts, cv.RETR\_TREE, cv.CHAIN\_APPROX\_SIMPLE)

#for cnt in contours:

# approx = cv.approxPolyDP(cnt, 0.01\*cv.arcLength(cnt, True), True)

# cv.drawContours(image, [approx], 0, (0), 5)

# x = approx.ravel()[0]

# y = approx.ravel()[1]

#cv.namedWindow("shapes", cv.WINDOW\_NORMAL)

#cv.resizeWindow("shapes", 750, 750)

#cv.imshow("shapes",image)

#cv.waitKey(0)

#cv.destroyAllWindows()

def save(self):

"""Automate the thresholding process and save image"""

cv.imwrite(self.output, self.thresholded)

def get\_cc(self, image):

"""Get the container cover"""

im = self.countPixels(image)

wht = self.countWhite(image)

return(im, wht, (1.0\*wht/im\*100))

@staticmethod

def countPixels(image):

"""Count the total number of pixels in the image and return it"""

ht, wd = image.shape #need a third object

return(ht\*wd)

@staticmethod

def countWhite(image):

"""Count the number of white pixels in the image"""

return (cv.countNonZero(image))

if \_\_name\_\_ == "\_\_main\_\_":

"""

Disclaimer: Worked on my computer. Directories will need to change if script is reused.

Also you'll need to get openCV installed which is not always a trivial task.

"""

dr = "" #input file directory

odr = "" #output file directory

csvfile = "" #results csv file

with open(csvfile, "w+") as csvf:

writer = csv.writer(csvf)

writer.writerow(["otsu threshold", "image size(px)", "white pixels (px)", "cover (%)", "original image", "thresholded image", "image name", "container", "treatment"])

files = listdir(dr)

for fp in files:

if (path.splitext(fp)[-1] != ".jpg"):

pass

else:

ts = path.join(odr, "ts-" + fp)

I = IMG(path.join(dr, fp), ts)

with open(csvfile, "a") as csvf:

writer = csv.writer(csvf)

writer.writerow(I.row)

#break