A Novel Python Video Processing Program for Identifying and Quantification of Soil Cracks in Real Time

Vaibhav Khandare Dilip [1,2] , He Huang [1], Ankit Garg [1]* Peng Lin[1] and Guoxiong Mei[3]

1 Department of Civil and Environmental Engineering, Shantou University, Shantou, Guangdong, 515041, China; ankit@stu.edu.cn
2 Department of Chemical Engineering, Indian Institute of Technology, Gandhinagar, Gujarat, 382024, India; dilip.khandare@iitgn.ac.in
3 College of Civil Engineering and Architecture, Guangxi University, China; meiguox@163.com
***** Correspondence: ankit@stu.edu.cn; Tel.: +86 15007542863

Abstract: The presence of soil cracks can lead to higher rainfall water infiltration, which will in turn cause instability in ground engineering infrastructure. Maintenance of ground infrastructure is hence essential, and such cracks are often identified through manual observations, that are laborious and not easily quantifiable. One of the ways could be to develop an automated program that can capture cracks through video processing. The objective of this study is to develop a simple video processing technique for soil crack sorting and its quantification in real-time. The new program automates the sorting of a region of cracks according to the given boundaries and quantifies the number of cracks in that region. A stepwise strategy is demonstrated to efficiently sort cracks and compute the crack intensity factor of real-time video with a negligible delay. Python script takes a frame from a video and analyzes it. It first automatically extracts the required portion of a video and computes cracks in this region. It also identifies the frames which have cracks and gives corresponding time and location to the administrator. Such a program can be useful in the future for analyzing videos obtained from UAV monitoring of the large area and, thus, identify vulnerable areas for maintenance.

Keywords: Cracks quantification, Region Isolation, Automation, Cost-effective, Python script

1. Introduction

Soil cracks arise because of the dryness of soil. Soil cracks are related to surface moisture content; hence it helps to understand the atmosphere–earth surface interaction. Surface moisture content is vital to understand atmosphere-earth surface interaction [1,2]. Atmosphere –earth surface interaction refers to heat exchange between the atmosphere and soil surface, evaporation, and cloud formation [3]. Cracking of soil in man-made slope can lead to higher infiltration of water during rainfall and hence, any instability of slopes [4]. Further, the presence of cracks in agricultural land can also lead to excess percolation of water below root depths. This will make it difficult for the shallower roots of crops to access water from deep underground. It is of utmost importance for any engineers or agriculturists to access cracking in the ground.

Most of the studies [5,6] focus on crack intensity factor (CIF) quantification using either image processing program (i.e., Image J) or in-house built PYTHON script. CIF symbolizes the total area of cracks in a given area of soil. A program built by Anangsha et al. [5] converts an image to grayscale. Further, noise is removed by a bilateral filter to preserve edges. Image is thresholded followed by calculation of CIF. The program is useful for a small set of images, however, it will be laborious and time-consuming for a large set of images, that are likely required for monitoring of large areas (man-made slopes and agricultural farms). It will be difficult for authority (through manual identification) to identify vulnerable areas (i.e., one of them is cracking) in a short time for any decision making. Recently, Unmanned Aerial Vehicles (UAVs) [7] have been deployed for conducting monitoring of large areas but it is still a cumbersome task to analyze output, which is informed of high-resolution videos (in large scale). It is reasonable to assume that a program that can capture and identify areas or images (containing cracks) from a video is required first before any quantification. Manual processing of large regions captured by drones would not be practical.

In literature, several video processing programs have been built in PYTHON for Thermal infrared imaging of geothermal environments by UAV [7], Drone video capture – a new method in precision agriculture [8], etc. Such programs in the PYTHON programming language uses libraries such as OpenCV, NumPy, etc. Uses of such libraries can be useful in building a novel PYTHON program for identifying and quantification of soil cracks in real-time.

This study aims to (i) create a novel a python program for sorting of region and quantification of soil cracks in real-time and also (ii) automation of sorting of a region of soil containing cracks. The program is written on a high-level Python language. This novel Python script can be potentially useful for analyzing the output of large ground infrastructure (agriculture, compacted man-made slopes, etc) captured using UAV. It uses contours method for sorting of region and quantification of soil cracks (i.e., crack area).

2. Materials and Methods

2.1. New Methodology for Sorting of Grids and Counting the Number of Cracks and Computing Their Respective Area

Figure 1 shows the flowchart for the sorting of grids and counting the number of soil cracks and computing their respective area. From the uppermost text box to the lowermost text box in figure 1 shows the stepwise breakdown of the methodology used in this code. This Python script is developed to automate the sorting of cracks and computing number of cracks. This python script is customized for the joining of cracks nearby cracks with negligible change in the area. The stepwise breakdown of each step of the novel Python script in a visual form is represented in figure 2. Computation time for python script is reduced by resizing the original video's frame to one fourth.

2.2. Capturing Frame and Noise Removal

2.2.1. Capturing Frame From a Video and Creating Kernel for Further Use

Capturing a Frame from Video

A-frame (figure 2a) is captured from the video by OpenCV built-in module cv2.VideoCapture (“video name”). It takes a successive frame from video using cv2.VideoCapture (“video name”).read(). It gives two output values, such as frame and truth value of the frame. This module helps in the systematic handling of frames.

Resizing of Frame to One-Fourth of the Original Size

Size of frame is reduced to one fourth using module cv2.resize(frame,(frame.shape[1]//2,frame.shape[0]//2)). It helps in reducing the computational time, i.e., reduces the delay between input and output frame. Time taken for one frame with resizing and without showing intermediates are 0.094764 seconds. Time taken for one frame without resizing and without showing intermediates are 0.234554 seconds.

Creating Kernels of Different Shapes and Size

Three Kernel of different sizes and shape is created. The first kernel is 4*4 square kernel by NumPy module np.ones((4,4),np.uint8), the second is 11*11 square kernel by NumPy module np.ones((11,11),np.uint8), and the third kernel is 11*11 circular kernel by OpenCV module cv2.getStructuringElement(cv2.MORPH_ELLIPSE,(11,11))

Creating Blank White Image for Background

A black blank image is created using module np.zeros_like(frame). It makes a matrix with size and dimensions equal to the frame, but it reduces all BGR value to zero. The above black image is converted into white by taking a negative black image. For a negative image, the ‘ ~ ‘ symbol is applied before the image.

2.2.2. Removing Noise of the Frame

For noise removal, cv2.pyrMeanShiftFiltering(frame,1,50) inbuilt module is used for removing background noise and smoothing the texture of the soil, as shown in this frame (figure 2b). pyrMeanShiftFiltering takes two input values one is a distance, and the other is a maximum value. A pixel is taken from the frame. A neighborhood of size of distance (input value) is created. From this neighborhood, pixel values are taken. The difference between neighborhood pixel BGR value and initial pixel BGR values is compared with the maximum limit (input value). If the difference is less then maximum limit, then BGR pixel values are changed to mean BGR value in that neighborhood.

2.3. Applying a Large Gaussian Filter on a Copy of the Frame.

A large Gaussian filter is applied to a copy of the frame by using a module cv2.GaussianBlur(frame, (75,75), 75) to obtain an image (figure 2c). It smoothens the edges of the grids. It makes all the pixel values of the frame near to the soil pixel values. It works on a Gaussian function.

2.4. Subtraction of Frames to Get an Image of Grids

The above two images are subtracted in this step using module cv2.subtract(frame, blur) to obtain an image (figure 2d). The first image is smoothed by pyrMeanShiftFiltering, in which random noise is reduced. While another image is from Gaussian blur, in which a soiled image is obtained. The subtraction of the above image s gives an image of grids. This grids image helps in the isolation of grids. In the frame, the soil region is segmented by the white ropes. Soil pixel value lies near to [[[103 113 123]]] and white rope value lies near to [[[252 242 237]]]. Gaussian blurred image all pixel value lies near to soil pixel. Hence, during subtraction soil pixel value goes to zero.

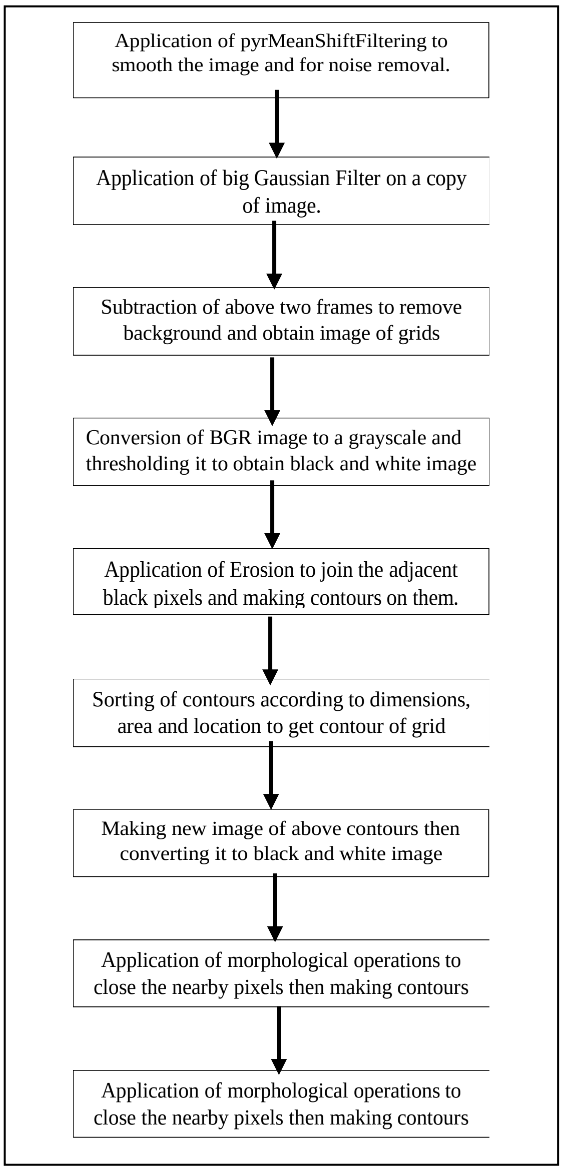
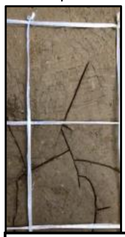

Application of pyrMeanShiftFiltering to smooth the image and for noise removal.
Application of big Gaussian Filter on a copy of image.
Subtraction of above two frames to remove background and obtain image of grids
Conversion of BGR image to a grayscale and thresholding it to obtain black and white image
Application of Erosion to join the adjacent black pixels and making contours on them.
Sorting of contours according to dimensions, area and location to get contour of grid
Making new image of above contours then converting it to black and white image
Application of morphological operations to close the nearby pixels then making contours
Application of morphological operations to close the nearby pixels then making contours

Figure 1. Flow Diagram of the Novel Python script for sorting of regions and quantification of cracks and their respective area

2.5. Grayscaling and Thresholding Image of Grids

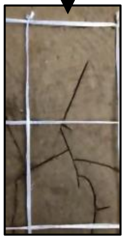
The above image is converted into a grayscale image (figure 2e) by using module cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY). During grayscale conversion, the light intensity is conserved. The digital image is converted into a gray color. Cracks are dark in color (i.e., gray value is less than 50). After the conversion of an image into grayscale, inverse threshold by using OpenCV module cv2.threshold(gray,50,255, cv2. THRESH_BINARY_INV) to obtain an image (figure 2f). Threshold requires two parameters, such as a limiting value and a final value. Grayscale pixel values range from 0-255. If the pixel value of grayscale is higher than the limiting value of the threshold, the pixel is converted into a given final value, while the rest of the pixel is converted to zero.

```
# Capturing Frame of a Video

cap = cv2.VideoCapture('08.mp4')

_,frame = cap.read()    # First frame of Video
```



(a) **Input image**

```
# Noise removal and smoothing texture

frame = cv2.pyrMeanShiftFiltering(frame,1,50)
```

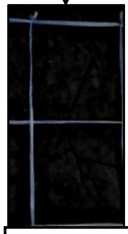


(b) Noise removal

```
# Applying large Gaussian Blur

blur = cv2.GaussianBlur(frame, (75,75), 75)
```

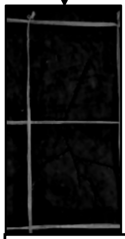
(c) Blurred image

```
# Subtracting above two images

diff = cv2.subtract(frame,blur)
```



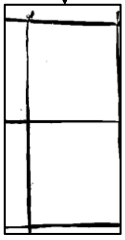
(d) Difference image

```
# Grayscale image

gray = cv2.cvtColor(diff, cv2.COLOR_BGR2GRAY)
```



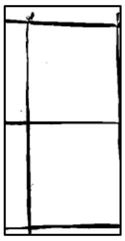
(e) Grayscale image

```
# Inverse Threshold Image

ot, thresh = cv2.threshold(gray, 50, 255, cv2.THRESH_BINARY_INV)
```



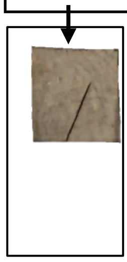
(f) Threshold image

```
# Erode Image

erode = cv2.erode(thresh,kernel,1)
```



(g) Erode image

```
# Image of soil patches

contours,_ = cv2.findContours(erode, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_NONE)

# After sorting

cv2.drawContours(imgo, contours, index, (255, 255, 255), -1)

# After Grascaling and thresholding above contour image

ymask = thresho>0    # Creating mask of contour for soil patch

final = np.zeros_like(frame,np.uint8)

final[ymask] =frame[ymask]

final[~ymask] = img[~ymask]
```



(h) Soil patch image

```
# Grayscaling soil patch

greyf = cv2.cvtColor(final,cv2.COLOR_BGR2GRAY)
```

(i) Grayscale image

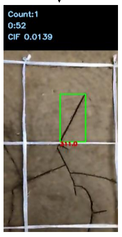
```
# Inverse Thresholding
_,threshf = cv2.threshold(greyf,60,255,cv2.THRESH_BINARY_INV)
```

(j) Threshold image

```
# Closing Morphological operation
threshf = cv2.morphologyEx(threshf, cv2.MORPH_CLOSE, kernelcircle)
```

(k) Closing image

```
# Displaying Contours
contours,_ = cv2.findContours(threshf, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_NONE)
  # After sorting, Displaying highlighting the crack
cv2.rectangle(fout,(x,y),(x+w,y+h),(0,255,0),2)
cv2.putText(fout, "{}".format(cv2.contourArea(contour)), (x,y+h+10), fonths, 0.45,(0,6,206),2)
cv2.putText(fout, "Count:{}".format(noc), (10,30), fonths , 1, (250, 206, 135) ,2)
```



(l) Output image

Figure 2. Steps involved in sorting of region and quantification of cracks and computing area of crack

2.6. Using Morphological Operation to Join the Nearby Black Pixels

On the above black and white image, the morphological operation is applied by using module cv2.erode(thresh, kernel,1). The image after erode morphological operation is (figure 2g). In erode, a black pixel near the edges expands according to the Kernel Size. It joins the nearby black pixels. In some areas, black pixels are not continuous. Erode connects the nearby pixels and help in a black-edged square formation. In this black square, white contours cover the part of the soil. For this erosion, a square kernel of one having dimension 4*4 is used.

2.7. Sorting of Contours According to Their Location, Area, and Dimensions of the Bounding Box of Contours

Contours are created on the above morphological operation by using module, contours,_ = cv2. findContours(erode, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_NONE). Contours are simple to close curves, which encloses the white pixel and are separated by black pixels. The bounding box is created on contours buy cv2.boundingRect(contour) module. The bounding box gives vertices of contour. The area enclosed by contour is obtained from the cv2.contourArea module. Logical reasoning based on width, height, location, and the area is used for sorting of contours.

2.8. Creating New Image of the Above Sorted Contour

Series of steps are used for creating new steps.

Contours are drawn on the new black image. A black image is created using a module np.ones ((frame.shape[0],frame.shape[1],3),np.uint8). It creates a black image of the same size and type.

Contours are drawn on this image using module cv2.drawContours (imgo, contours, index, (255, 255, 255), -1). It makes a white patch of soil on the above black image.

Image is converted into grayscale then thresholded to get a black and white image. Using same module such as cv2.cvtColor(imgo, cv2.COLOR_BGR2GRAY) and cv2.threshold(imgo, 100,255,cv2.THR ESH_BINARY) respectively.

A mask is created from the above thresholded image. A new black image is created using module np.zeros_like(frame, np.uint8). After that, the mask is applied to the above black image to obtain an image of figure 2h. The first frame mask is applied to the black image using the final[ymask] = frame[ymask]. On the same image, a negative mask is applied to the white image using final[~ymask] = img[~ymask]

2.9. Grayscaling and Thresholding Above Soil Patches Image

Image is converted into grayscale image (figure 2i) using cv2.cvtColor(final, cv2.COLOR_BGR2GRAY). Above Grayscale image is thresholded using module cv2.threshold(greyf,60,255, cv2.THRESH_BINARY_INV) to obtain an image (figure 2j). An inverse threshold is used because cracks have value lies below 50. The inverse threshold converts all the pixels below 60 to 255 of the Grayscale image.

2.10. Applying Morphological Operation

The morphological operation of closing is applied using module cv2.morphologyEx(threshf, cv2.MORPH_CLOSE, kernelcircle) to obtain (figure 2k). Closing Morphological operation is a combination of dilation and erosion. First, the image is dilated using the kernel. After that image is eroded using the same kernel. It joins the cracks, which lie in the kernel range. A circular kernel of size 11*11 is used for achieving the purpose of closing. The circular kernel instead of the square is used because the cracks are random. By this, nearby cracks are joined without any significant increase in area.

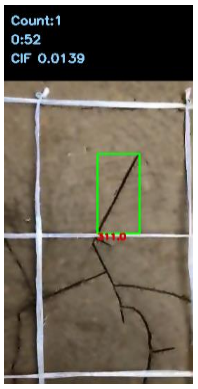
2.11. Creating Contours and Sorting Them

On the above binary image contours are created using module cv2.findContours(threshf, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_NONE). After that, bounding Rectangle i.e., vertices and contour area values are extracted using modules cv2.boundingRect(contour) and cv2.contourArea(contour) respectively. Logical reasoning based on contour area and dimensions are applied to sort the contours. Bounding Rectangle and contourArea is displayed on the output frame (figure 2l) using module cv2.rectangle(fout, (x, y), (x+w,y+h), (0,255,0),2) and cv2.putText(fout, "{}".format(cv2.contourArea(contour)), (x,y+h+10), fonths, 0.45, (0, 6, 206) ,2) respectively. CIF value is calculated by the ratio of the total number of white pixel values in the last threshold (figure 2j) and the mask applied on the frame.

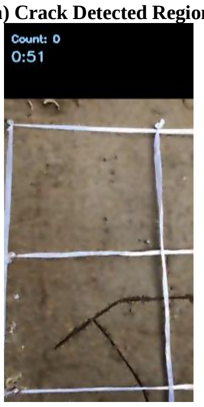
3. Results

3.1. Segregation of Frames from Videos Containing Ground with and without Cracks

Present Novel python code automates the segregation of frames. It takes many frames of video in a second. It instantaneously segregates the region containing cracks. It takes every frame from videos and gives a period of time that contains cracks as shown in figure 3a and figure 3b. The above portion of each frame gives the necessary details of the frame. In figure 3a, topline in frames gives the details about the soil region. It shows details related to the quantification of cracks. Second-line shows the time of the video. It tells the administrator about the time at which the cracked frame is detected. Python script can be adapted to save the crack frame with the required details to save the time required for rewinding the many videos and get to a particular time to give the frame number of the video. If cracks are not present it shows it gives output to the administrator by Count=0 as shown in image figure 3b. In this image, there are cracks beneath the center soil patch of the image. However, it does not detect that cracks and shows the efficient sorting of regions. Manual sorting uses a rectangle to crop the soil region. Soil’s region divider may not be a perfect rectangle. Manual sorting will sort region with some boundaries and other areas. However, this novel script cuts only soil patches.



(a) Crack Detected Region



(b) Crack undetected region

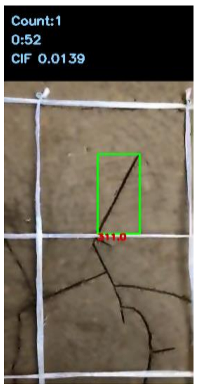
Figure 3. Segregation of soil patches containing cracks

3.2. Manual Verification of Results with Results of This Newly Developed Novel Python Script Using Experiments.

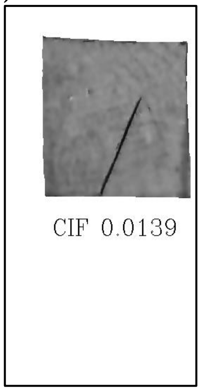
The output values of soil crack area from the novel Python Script are compared with those from manual processing of image. Manual values are calculated by image processing. In this method, the image is converted into a binary form, then the CIF value is calculated using program similar to that of Anangsha et al. [5]. The visual comparison between the manual value and this present code value is given in figure 4a and figure 4b. Along with the original frame figure 4a contains the number of cracks, CIF value of soil, dimension, and area of cracks with time. figure 4b has the above same information without any time. As shown in these figures, this novel python script gives similar CIF value as it given by manual method. Also, the crack bounding box and dimensions are the same.

4. Discussion

Based on the results in figures 3 and 4, it can be concluded that the novel PYTHON script captures frames containing soil cracks from a video and also quantify number, area and CIF automatically. This study is a first step towards automation of quantification of soil cracks from a video. Further studies are required to integrate such programs in UAVs for capturing soil cracking in large areas.



(a) Present code CIF value



(b) Manual CIF value

Figure 4. Comparison of CIF value of this novel code with Manual value

5. Conclusions

A novel Python Script was demonstrated in this present study to automate the sorting of region and quantification of soil cracks in real time from a video. The novel python script considers the entire portion of the frame and automates the sorting of the soil region. After sorting, it gives the number of cracks and their respective area. This python script is very efficient and swift; the time difference between input and output for a frame is one-tenth of a second. This novel script can be adapted in UAVs monitoring and quantification of the cracks in large ground areas including agricultural drought areas, man-made slopes etc. This python script can be used for the maintenance system for large regions administration using.

Supplementary Materials: The following are available online at https://stumail-my.sharepoint.cn/:v:/g/personal/18hhuang2_stu_edu_cn/EQliOHSZ5mdKoy5P9-MGvtcBrCgantVZtBIJs1KO8aGFEQ?e=ChPpCn Video S1: Original video.

Author Contributions: Conceptualization, Ankit Garg.; methodology, Ankit Garg and Vaibhav Khandare; software, Ankit Garg and Vaibhav Khandare; validation, Ankit Garg and Vaibhav Khandare.; formal analysis, Ankit Garg and Vaibhav Khandare; investigation, Ankit Garg and Vaibhav Khandare; resources, Ankit Garg, Vaibhav Khandare and He Huang; data curation, Ankit Garg and Vaibhav Khandare; writing—original draft preparation, Ankit Garg and Vaibhav Khandare; writing—review and editing, Ankit Garg and He Huang; visualization, Ankit Garg and He Huang; supervision, Ankit Garg; project administration, Ankit Garg and He Huang; funding acquisition, Ankit Garg. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by China National Natural Science Foundation (NSFC) Youth Project, grant number 41907252

Acknowledgments: N/A

Conflicts of Interest: The authors declare no conflict of interest.

Appendix A

Python Code:

```
import cv2, numpy as np #Importing Libraries


cap = cv2.VideoCapture('08.mp4') #Capturing frame of video
_,frame = cap.read() #taking first frame
frame = cv2.resize(frame,(frame.shape[1]//2,frame.shape[0]//2)) #reducing size to half
#Creating Kernels
kernel = np.ones((4,4),np.uint8)
kernel5 = np.ones((11,11),np.uint8)
kernelcircle = cv2.getStructuringElement(cv2.MORPH_ELLIPSE,(11,11)) #circular Kernel
#Creating white backgroung
img = np.zeros_like(frame)
img = ~img
while cap.isOpened():
    com = frame
    cv2.imshow("Fig. 1a",frame)
    #Step S1
    frame = cv2.pyrMeanShiftFiltering(frame,1,50) #Reducing Noise
    cv2.imshow("Fig. 2a",frame)
    #Step S2
    blur = cv2.GaussianBlur(frame, (75,75), 75) #Extracting color of soil
    cv2.imshow("Fig. 3a",blur)
    fout = frame.copy() #Saving a copu of frame
    #Step S3
    diff = cv2.subtract(frame,blur) #Creating grid image
    cv2.imshow("Fig. 4a",diff)
    #Step S4
    gray = cv2.cvtColor(diff, cv2.COLOR_BGR2GRAY) #Grayscale conversion
    cv2.imshow("Fig. 5a",gray)
    ot, thresh = cv2.threshold(gray, 50, 255, cv2.THRESH_BINARY_INV) #Black and White conversion
    cv2.imshow("Fig. 6a",thresh)
    #Step S5
```

```
 erode = cv2.erode(thresh,kernel,1) #Joining nearby white pixels.
    cv2.imshow("Fig. 7a",erode)
    contours,_ = cv2.findContours(erode, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_NONE)
    #Step S6
    i = -1
    cs = []
    for contour in contours:
        i = i + 1
        (x, y, w, h) = cv2.boundingRect(contour)
        #Sorting of contours
        if w > 180 and (450 >h > 180) and x!=0 and y!=0 and x+w
!=erode.shape[1] and y+h !=erode.shape[0]:
            cs.append(i)
            cv2.rectangle(com, (x,y),(x+w,y+h), (0,255,0),2)
    #Step S7
    imgo = np.ones((frame.shape[0],frame.shape[1],3),np.uint8)
    for index in cs:
        cv2.drawContours(imgo, contours, index, (255, 255, 255), -1)
    imgo = cv2.cvtColor(imgo,cv2.COLOR_BGR2GRAY)
    _,thresho = cv2.threshold(imgo,100,255,cv2.THRESH_BINARY)
#     thresho = cv2.erode(thresho,kernel5,1)
    ymask = thresho > 0
    final = np.zeros_like(frame, np.uint8)
    final[ymask] = frame[ymask] #final image having required portion
    final[~ymask] = img[~ymask]
    cv2.imshow("Fig. 8a",final)
    greyf = cv2.cvtColor(final,cv2.COLOR_BGR2GRAY)
    cv2.imshow("Fig. 9a",greyf)
    _,threshf                                                    =
cv2.threshold(greyf,60,255,cv2.THRESH_BINARY_INV)
    cv2.imshow("Fig. 10a",threshf)
    #Step S8
    threshf = cv2.morphologyEx(threshf, cv2.MORPH_CLOSE,
kernelcircle) #dilation + ersoiom
    cv2.imshow("Fig. 11a",threshf)
    contours,_ = cv2.findContours(threshf, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_NONE)
    output = final.copy()
    #Step S9
    noc = 0
    fonths = cv2.FNT_HERSHEY_SIMPLEX
    for contour in contours:
        if cv2.contourArea(contour)>50:
            (x, y, w, h) = cv2.boundingRect(contour)
            # Sorting of cracks
            if (w > 16 and h < 190) or w >1:
                noc +=1
                cv2.rectangle(fout, (x,y),(x+w,y+h), (0,255,0),2)
                cv2.putText(fout,
"{}".format(cv2.contourArea(contour)), (x,y+h+10), fonths, 0.45, (0,
6, 206) ,2)
    cv2.putText(fout, "Count:{}".format(noc), (10,30), fonths , 1,
(250, 206, 135) ,2)
    cv2.imshow("Fig. 12a",fout)
    ret, frame = cap.read()
    if frame is None:
        break
    frame = cv2.resize(frame,(frame.shape[1]//2,frame.shape[0]//2))
    if cv2.waitKey(0) == 27:
        break
cv2.destroyAllWindows()
cap.release()
```

References

[1] Deardorff, J. W. A Parameterization of Ground-Surface Moisture Content for Use in Atmospheric Prediction Models, Journal of Applied Meteorology, 1977, 16(11), 1182–1185, https://doi.org/10.1175/1520-0450(1977)016<1182:APOGSM>2.0.CO;2

[2] Gadi V., Singh S., Singhariya M. and Garg A., Modeling soil-plant-water interaction: effects of canopy and root parameters on soil suction and stability of green infrastructure. Engineering Computations, 2018, 35(3), 1543-1566, https://doi.org/10.1108/EC-07-2017-0280

[3] Hong X., Leach M.J. and Raman S., A sensitivity study of convective cloud formation by vegetation forcing with different atmospheric conditions. Journal of Applied Meteorology, 1995, 34(9), 2008-2028, https://doi.org/10.1175/1520%200450(1995)034%3C2008:assocc%3E2.0.co;2

[4] Garg A., Jason L. C., and Ng C. W. W., Field study on influence of root characteristics on soil suction distribution in slopes vegetated with Cynodon dactylon and Schefflera heptaphylla. Earth Surface Processes and Landforms, 2015, 40(12), 1631-1643, https://doi.org/10.1002/esp.3743

[5] Anangsha A., Gadi V. K., Bordoloi S., Kothapalli S. K., Sreedeep S., Guoxiong M., andGarg A., A New Autonomous Program Customized for Computing Surface Cracks in an Unsaturated Soil in a 1-D Column, Journal of Testing and Evaluation, 2019, 47(5), 3822–3835, https://doi.org/10.1520/JTE20180609

[6] Bordoloi S., Garg A., Sreedeep S., Lin P., Mei G., Investigation of cracking and water availability of soil-biochar composite synthesized from invasive weed water hyacinth, Bioresource technology, 2018, 263, 665-677, https://doi.org/10.1016/j.biortech.2018.05.011

[7] Nishar A., Richards H., Richards S., Breen D., Robertson J. and Bree B., Thermal infrared imaging of geothermal environments by UAV (unmanned aerial vehicle). Journal of Unmanned Vehicle Systems, 2016, 4(2), 136-145, https://doi.org/10.1139/juvs-2015-0030

[8] Mihajlow R., and Ivanova A., Drone video capture – a new method in precision agriculture. Annual journal of technical university of varna, Bulgaria, 2019, 3(2), 32-38, https://doi.org/10.29114/ajtuv.vol3.iss2.141