# Satellite Image Alignment, Differencing and Segmentation

Research Practice

Submitted in partial fulfillment of the requirements of

**BITS G540, Research Practice**

By

**Rishikesh S**

ID No- 2016H103016G

Under the supervision of

**Prof. Neena Goveas**

**Associate Professor, Dept. of CS & IS**

BITS PILANI K K Birla Goa Campus



**Birla Institute of Technology and Science, Pilani**

K. K. Birla Goa Campus

DateMay 2017

# **ABSTRACT**

There are a wide variety of applications that require identification of the changes that has happened on the surface of the earth, eg: gas pipeline obstruction, environmental changes, urban development rate etc. These applications require identifying the particular area in which the change has actually happened. Processing the whole image and identifying all the objects in the image will be computationally costly if we have a huge imageset. But if we identify parts of each image where the change has actually happened, we can identify those objects in that area alone. My Research practice involves coming up with a method to do the same.

# **CONTENTS**

[**ABSTRACT** 2](#_Toc481360977)

[**BASIC DIFFERENCE** 4](#_Toc481360979)

[**THRESHOLDING** 6](#_Toc481360980)

[**IMAGE ALIGNMENT** 8](#_Toc481360981)

[1. **Translation** 8](#_Toc481360982)

[2. **Euclidean** 8](#_Toc481360983)

[3. **Affine** 8](#_Toc481360984)

[4. **Homography** 8](#_Toc481360985)

[**CLAHE (Contrast Limited Adaptive Histogram Equalization)** 11](#_Toc481360986)

[**CONTOUR AND AREA BASED FILTERING** 13](#_Toc481360987)

[**SEGMENTATION:** 15](#_Toc481360988)

[**PERCENTAGE CHANGE CALCULATION**: 16](#_Toc481360989)

[**COMPLETE FLOW** 17](#_Toc481360990)

[**CONCLUSION** 19](#_Toc481360991)

[**REFERENCES** 20](#_Toc481360992)

# **BASIC DIFFERENCE**

* I took 2 satellite images from the internet and tried to find the difference between the two using basic pixel-wise subtraction.
* Cv2.subtract computes the difference between each element in two arrays.

1. **import** numpy as np
2. **import** cv2
4. #read two source images in grayscale
5. img1 = cv2.imread('a.jpg',0)
6. img2 = cv2.imread('b.jpg',0)
8. #get the dimensions of the source
9. h, w = img2.shape
11. #calculate what disappeared
12. disappeared = cv2.subtract(img1, img2)
14. #calculate what appeared
15. appeared = cv2.subtract(img2, img1)
17. #write result image
18. cv2.imwrite('disappeared.jpg', disappeared)
19. cv2.imwrite('appeared.jpg', appeared)

The difference calculated by the above code is incredibly crude and can roughly show what has appeared or disappeared. But this cannot be used for identifying the objects in the image as it doesn’t isolate or segment it anyway.

Source Images:  Appeared and Disappeared:

b

# **THRESHOLDING**

* In the previous result, we saw that the resultant images were very crude to get any good information out of it. To filter out unwanted components from the images, I used Gaussian Blur and OTSU’s Thresholding from the cv2 library.
* The Gaussian Blur reduces the noise from the images and the Otsu’s thresholding reduces a gray-level image to a binary image.

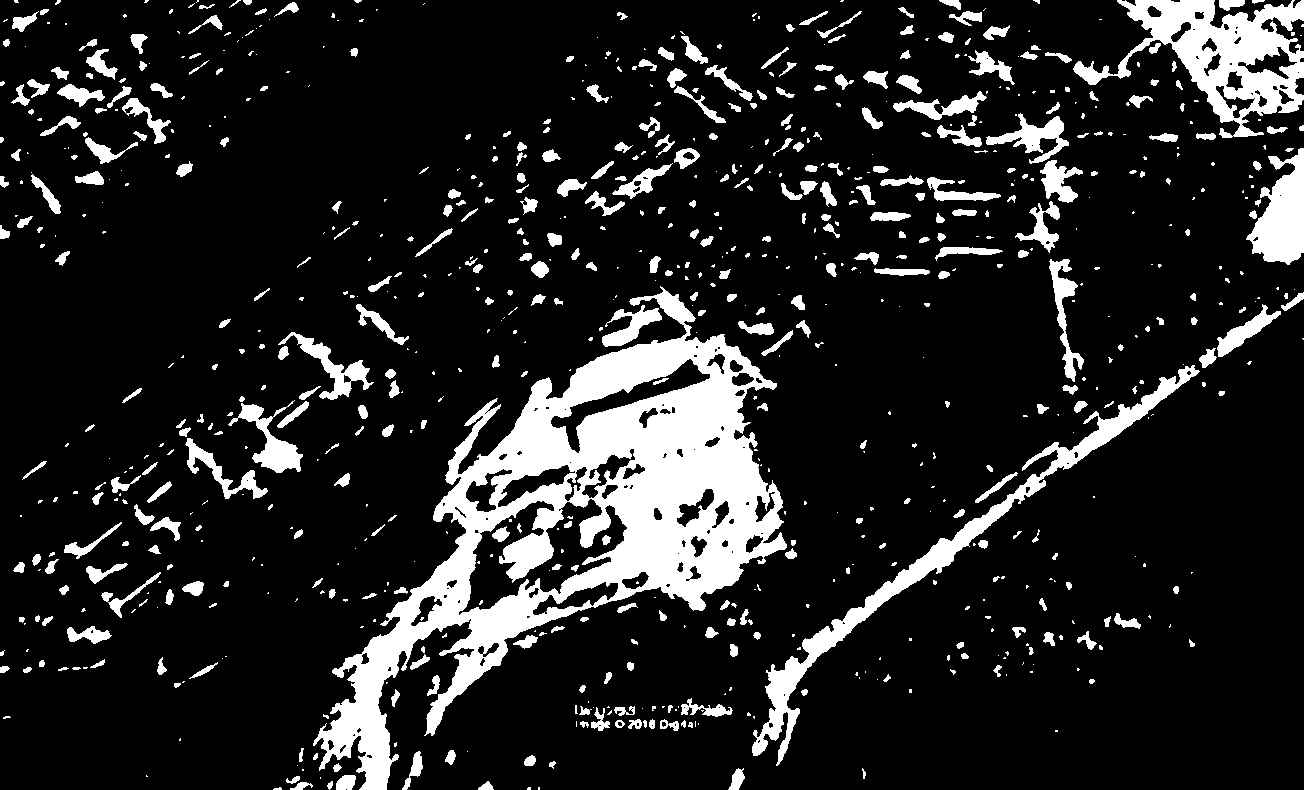
1. #otsu's thresholding with Gaussian blur for noise reduction
2. **import** numpy as np
3. **import** cv2
5. #get both images in grayscale
6. img1 = cv2.imread('a.jpg',0)
7. img2 = cv2.imread('b.jpg',0)
9. #get dimensions
10. h, w = img2.shape
12. #create a zeroed out image of same dimensions
13. res1 = np.zeros((h,w,1), np.uint8)
15. #find difference b/w 1st and 2nd image and vice versa
16. disappear = cv2.subtract(img1, img2)
17. appear = cv2.subtract(img2, img1)
19. #set threshold values
20. thresh = 70
21. maxValue = 255
23. # apply gaussian blur and otsu's threshold
24. blur1 = cv2.GaussianBlur(disappear,(5,5),0)
25. ret,disappear = cv2.threshold(blur1,0,255,cv2.THRESH\_BINARY+cv2.THRESH\_OTSU)
26. blur2 = cv2.GaussianBlur(appear,(5,5),0)
27. ret,appear = cv2.threshold(blur2,0,255,cv2.THRESH\_BINARY+cv2.THRESH\_OTSU)
29. #results
30. cv2.imwrite('disappear.jpg', disappear)
31. cv2.imwrite('appeared.jpg', appear)

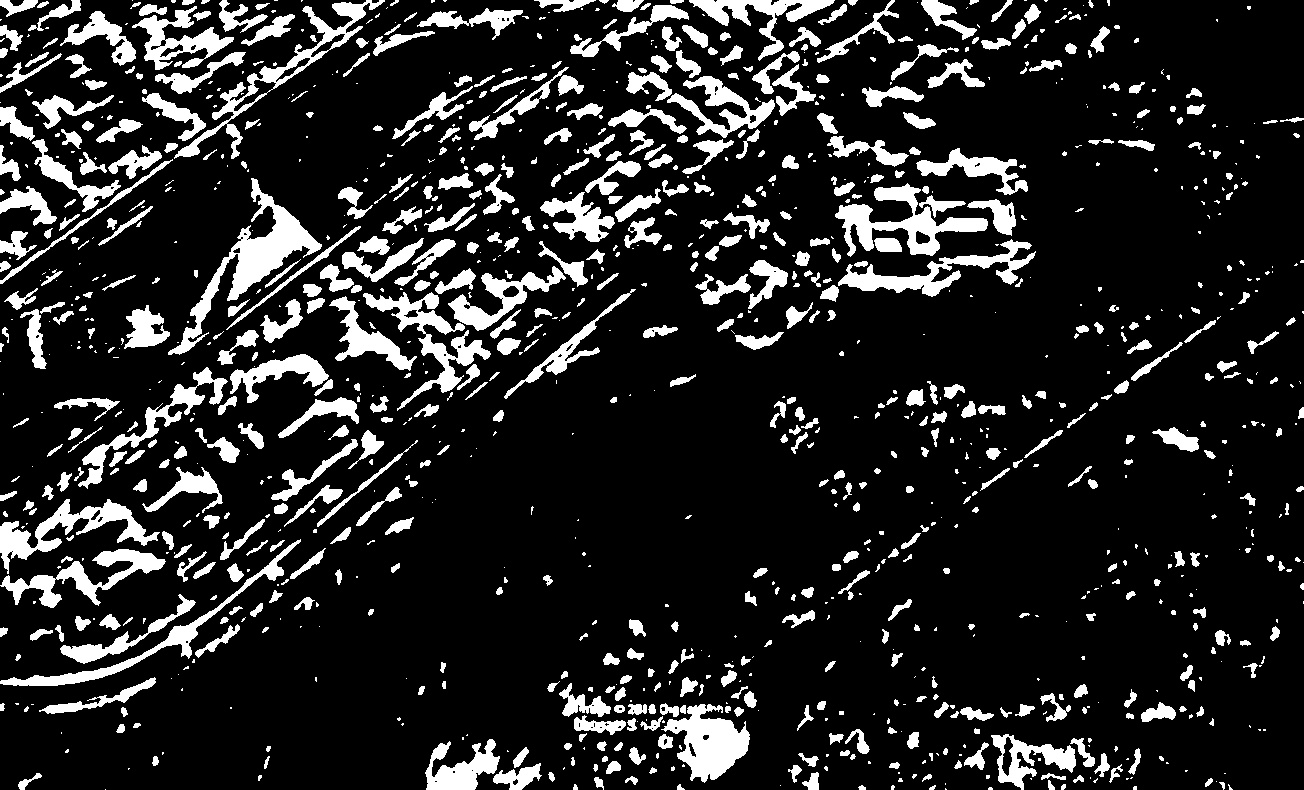
The results show that it has much filtered information in the form of a binary image which is much better that just using subtraction alone. But it still includes even minor differences like shadows and brightness levels in the images. I also haven’t segmented the portionswhich changed to be used for identification.

**SOURCE IMAGES:**

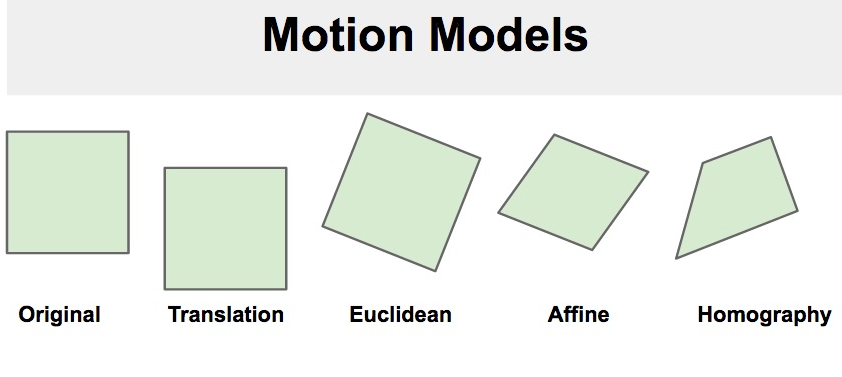
Appeared and Disappeared:





# **IMAGE ALIGNMENT**

* One important problem I identified while experimenting was that if the source images are out of alignment the differencing will give wrong results.
* The images may differ due to the following transformations:



## Translation:

One or more of the source images may be shifted to a different position compared to the other. The first image is be shifted by (x , y) to obtain the second image.

## Euclidean:

One or more of the source images can be shifted and rotated compared to the other. The first image is a rotated by an angle and shifted by (x,y) to get second image.

## Affine:

One or more of the source images can be shifted, rotated and sheared compared to the other. The transform has 6 parameters. For example, when a square undergoes an Affine transformation, parallel lines remain parallel, but lines meeting at right angles no longer remain orthogonal.

## Homography:

Any type of 2D transformation of the 2D images. This transform has 8 parameters. For example, a square when transformed using a Homography can change to any quadrilateral.

In order to realign the images I used an implementation of the Enchanced Correlation Coefficient (ECC) [1]. It has been integrated into openCV. It is independent of brightness and contrast variations and it is computationally feasible to align the images using ECC.

1. **import** numpy as np
2. **import** cv2
4. **def** align():
5. im1 = cv2.imread('before.jpg')
6. im2 = cv2.imread('after.jpg')
8. # Convert images to grayscale
9. im1\_gray = cv2.cvtColor(im1,cv2.COLOR\_BGR2GRAY)
10. im2\_gray = cv2.cvtColor(im2,cv2.COLOR\_BGR2GRAY)
12. # Find size of image1
13. sz = im1.shape
15. # Define the motion model
16. warp\_mode = cv2.MOTION\_TRANSLATION
18. # Define matrices and initialize the matrix to identity
19. **if** warp\_mode == cv2.MOTION\_HOMOGRAPHY :
20. warp\_matrix = np.eye(3, 3, dtype=np.float32)
21. **else** :
22. warp\_matrix = np.eye(2, 3, dtype=np.float32)
24. #  number of iterations.
25. number\_of\_iterations = 5000;
27. # Specify the threshold of the increment two iterations
28. termination\_eps = 1e-10;
30. # termination criteria
31. criteria = (cv2.TERM\_CRITERIA\_EPS | cv2.TERM\_CRITERIA\_COUNT, number\_of\_iterations,termination\_eps)
33. #ECC algorithm.
34. (cc, warp\_matrix) = cv2.findTransformECC (im1\_gray,im2\_gray,warp\_matrix, warp\_mode, criteria)
36. **if** warp\_mode == cv2.MOTION\_HOMOGRAPHY :
37. # Use warpPerspective for Homography
38. im2\_aligned = cv2.warpPerspective (im2, warp\_matrix, (sz[1],sz[0]), flags=cv2.INTER\_LINEAR + cv2.WARP\_INVERSE\_MAP)
39. **else** :
40. # Use warpAffine for Translation, Euclidean and Affine
41. im2\_aligned = cv2.warpAffine(im2, warp\_matrix, (sz[1],sz[0]), flags=cv2.INTER\_LINEAR + cv2.WARP\_INVERSE\_MAP);
43. # Show final results
45. cv2.imwrite('aligned\_A.jpg', im1)
46. cv2.imwrite('aligned\_B.jpg', im2\_aligned)

**SOURCE IMAGES:**  

**ALIGNED:**



# **CLAHE (Contrast Limited Adaptive Histogram Equalization)**

Satellite images are often of poor quality and the contrast may not be good in these images. In order to fix this we can use CLAHE (Contrast Limited Adaptive Histogram Equalization).

1. **import** numpy as np
2. **import** cv2
4. **def** clahe():
5. img1 = cv2.imread('aligned\_A.jpg',0)
6. img2 = cv2.imread('aligned\_B.jpg',0)
7. # create a CLAHE object (Arguments are optional).
8. clahe = cv2.createCLAHE(clipLimit=2.0, tileGridSize=(8,8))
10. cl1 = clahe.apply(img1)
11. cv2.imwrite('clahe\_A.jpg',cl1)
13. cl2 = clahe.apply(img2)
14. cv2.imwrite('clahe\_B.jpg',cl2)

Histogram Equalization increases the global [contrast](https://en.wikipedia.org/wiki/Contrast_(vision)) of images. Areas of lower local contrast gains a higher contrast. One disadvantage of is Histogram Equalization that due to over-brightness, we might lost some information from the image. In order to overcome this we use CLAHE. Image is divided into small blocks called "tiles". Then each of these blocks are histogram equalized. So in a small area, histogram would confine to a small region. If noise is present, it will be amplified. To avoid this, **contrast limiting** is applied. If any histogram bin is above the specified contrast limit, those pixels are clipped and distributed uniformly to other bins before applying histogram equalization.

**NOTE:**

When CLAHE is applied as a part of this implementation it includes even minor changes like disappearance of grass from an area. If we don’t require such fine grained details, it is better to leave this out. In case we want fine grained results we can include this in our results. I have left this out of the program flow as an optional add on.

**SOURCE IMAGE:** **AFTER CLAHE:**



# **CONTOUR AND AREA BASED FILTERING**

* Currently the flow of the program is that, It first aligns the two source images with ECC as seen earlier, then the difference is found by subtraction and then Otsu’s threshold is applied.
* Now that we have the difference at hand we need to contourize the result.
* This will give us a vector of the contours where the changes have happened and along with interesting properties like area of the contour which I used for refinement.
* Now, Based on the threshold area, the I only filter out the small insignificant contours.

1. **import** cv2
2. **import** numpy as np
3. **import** os
5. **def** refine2():
6. threshold\_area=3000.0
7. image\_src = cv2.imread("appeared.jpg")
8. im1 = cv2.imread('aligned\_A.jpg')
9. im2 = cv2.imread('aligned\_B.jpg')

12. gray = cv2.cvtColor(image\_src, cv2.COLOR\_BGR2GRAY)
13. ret, gray = cv2.threshold(gray, 250, 255,0)
15. image, contours, hierarchy = cv2.findContours(gray, cv2.RETR\_LIST, cv2.CHAIN\_APPROX\_SIMPLE)
16. mask = np.zeros(image\_src.shape, np.uint8)
17. cnts = sorted(contours, key=cv2.contourArea)
18. i=0
19. **for** c **in** cnts:
20. area=cv2.contourArea(c)
21. **if** area > threshold\_area:
22. cv2.drawContours(mask, [c], -1, (255,255,255), -1)

25. cv2.imwrite("refined.jpg", mask)

**NOTE:** The threshold area may vary for different images, so we might need to change it a bit to get good results.

**SOURCE IMAGES:**  

**COUNTOURISED AND AREA FILTERED**





# **SEGMENTATION:**

* Now that I have the Area based filtered contour, I can use this information to segment the original image.
* I first get the dimensions and the coordinates of the contours. Then using this I create new images that are segmented from the changed image.
* These images can be used for Identification. [4]

1. **import** cv2
2. **import** numpy as np

5. **def** segment():
6. threshold\_area=3000.0
7. image\_src = cv2.imread("refined.jpg")
8. im1 = cv2.imread('aligned\_A.jpg')
9. im2 = cv2.imread('aligned\_B.jpg')

12. gray = cv2.cvtColor(image\_src, cv2.COLOR\_BGR2GRAY)
13. ret, gray = cv2.threshold(gray, 250, 255,0)
15. image, contours, hierarchy = cv2.findContours(gray, cv2.RETR\_LIST, cv2.CHAIN\_APPROX\_SIMPLE)
16. mask = np.zeros(image\_src.shape, np.uint8)
17. cnts = sorted(contours, key=cv2.contourArea)
18. i=0
19. **for** c **in** cnts:
20. area=cv2.contourArea(c)
21. **if** area > threshold\_area:
22. x,y,w,h = cv2.boundingRect(c)
24. strin=str(i)+".jpg"
25. i=i+1
27. cv2.imwrite(strin, im2[y:y+h,x:x+w])

# **PERCENTAGE CHANGE CALCULATION:**

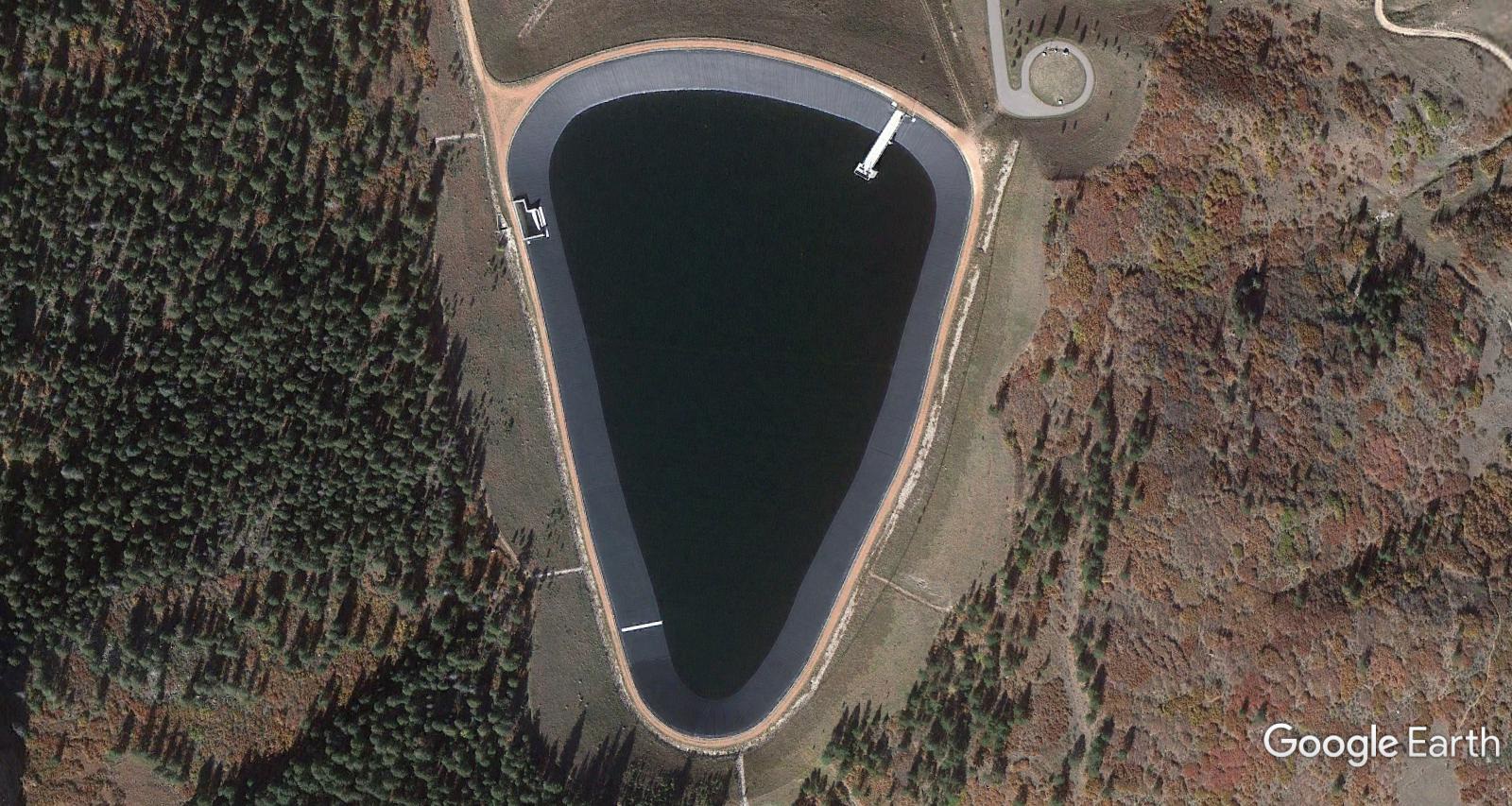
* Based on the area filtered contour binary image we can calculate the percentage changes in the following way. The cv2.countNonZero from the OpenCV 3 package is used to achieve this [5].
* It counts the number of pixels in the image which are not balck. Since ours is a binary image. We can use this function to count the white pixels in the white portions which represent th changes. Then calculate the percentage of change.

1. **from** \_\_future\_\_ **import** division
2. **import** cv2
3. **import** numpy as np

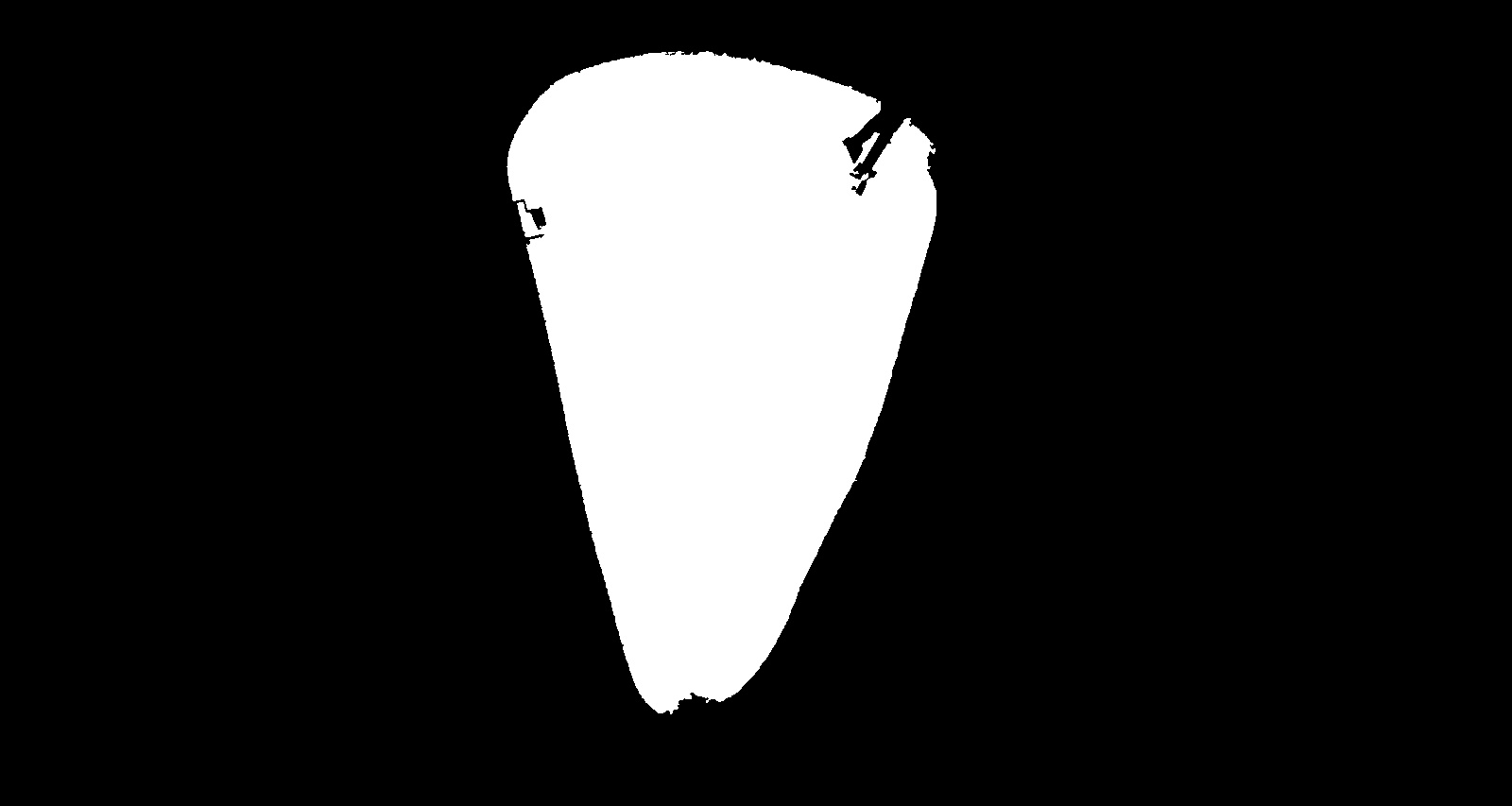
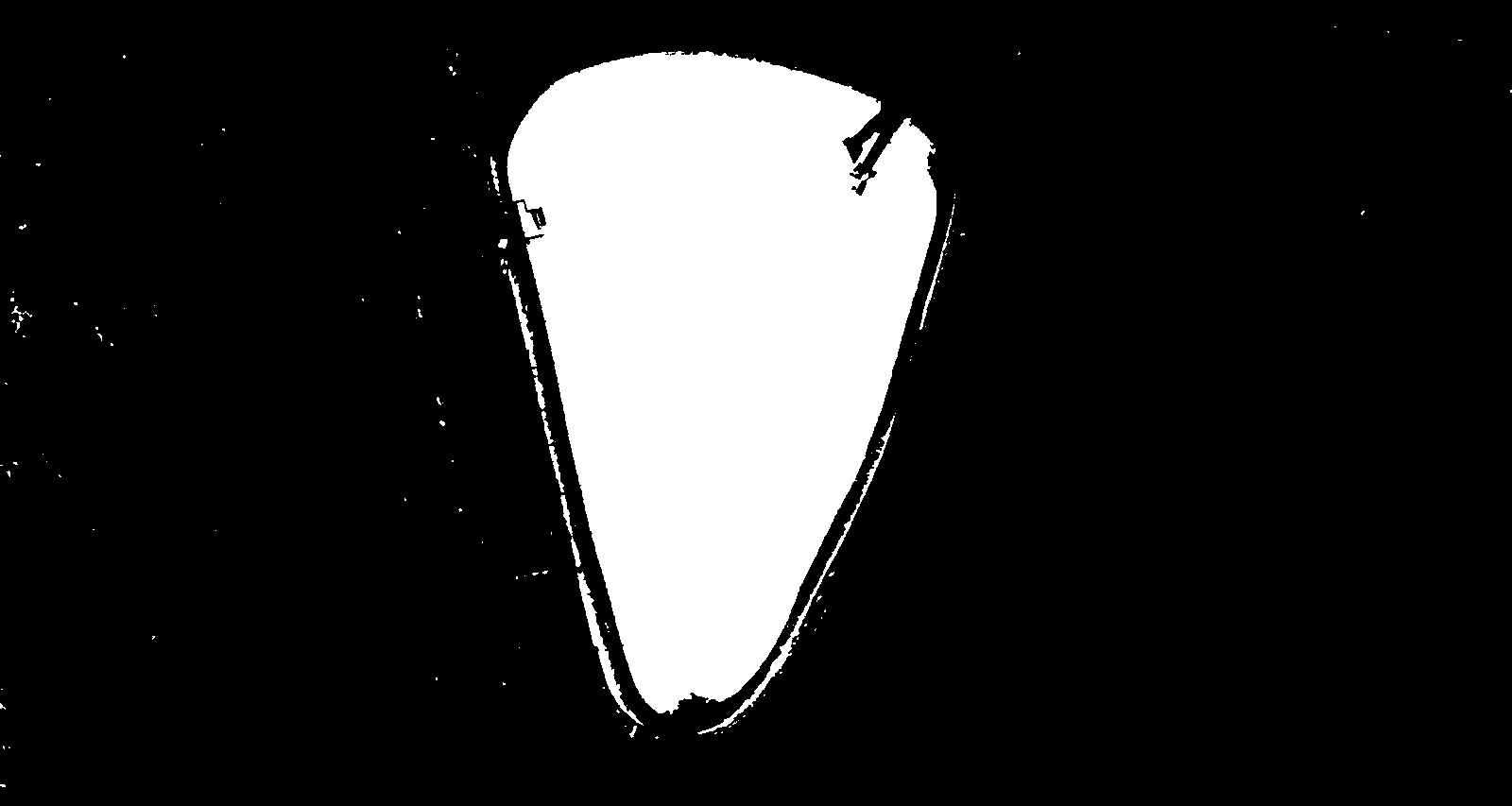
6. # read image
7. img = cv2.imread('refined.jpg',0)
9. # get the number of pixels which are not black
10. count=cv2.countNonZero(img)
12. #get the size of the image
13. size = img.size
15. # calculate the percentage of pixels which are white
16. percent=count/size\*100
18. # print result
19. **print** "percent:", percent

# **COMPLETE FLOW**

SOURCE IMAGES:



**CONTOURIZED AND FILTERED:**



***Percentage of Change = 16.014***

**SEGMENTATION:**



# **CONCLUSION**

The proposed image processing pipeline can detect changes in a given satellite image which can be used for feeding it to an mage identification pipeline. The proposed pipeline works well even with low quality satellite images. Instead of Directly Satrting to identify each and evry component in the image, this method reduces the overhead by allowing any identification algorithm to identify only components that have changed between the images.

For detection of loss of vegetation cover etc., where the image contrast is not good, I find that incorporation of a contrast adjustment techniques like CLAHE ([Contrast Limited Adaptive Histogram Equalization](https://en.wikipedia.org/wiki/Adaptive_histogram_equalization)) will give a fined grained segmentation [5].

In the future as human settlements starts expanding encroachment detection and loss of vegetation cover will need to be done in an automated manner. I will be working to optimize the image processing pipeline further. Another requirement is to make it available for use to any user. This would require packaging the software and designing customizable features in the GUI.

# **REFERENCES**

1. G. D. Evangelidis and E. Z. Psarakis, "Parametric Image Alignment Using Enhanced Correlation Coefficient Maximization," in IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 30, no. 10, pp. 1858-1865, <http://www.learnopencv.com/image-alignment-ecc-in-opencv-c-python/>, Accessed on 20th March 2017.
2. Vala, HJ & Baxi, Astha (2013). "A review on Otsu image segmentation algorithm", International Journal of Advanced Research in Computer Engineering & Technology (IJARCET) 2 (2): 387. <http://docs.opencv.org/trunk/d7/d4d/tutorial_py_thresholding.html>, Accessed on 20th March 2017.
3. Gaussian Blur: Shapiro, L. G. & Stockman, G. C: "Computer Vision", page 137, 150, Prentice Hall, 2001.
4. Straight Bounding Rectangle from the OpenCV3 Package: <http://docs.opencv.org/trunk/dd/d49/tutorial_py_contour_features.html>, Accessed on 20th March 2017.
5. Counting non black pixels using OpenCV3 Package: <http://docs.opencv.org/2.4/modules/core/doc/operations_on_arrays.html> Accessed on 20th March 2017.
6. K. Zuiderveld, “ Contrast Limited Adaptive Histogram Equalization” in P. Heckbert, Graphics Gems IV, Academic Press 1994.