Wrocław University
of Science and Technology

Artificial intelligence and Computer Vision Project

Image analysis algorithm for Mars satellite images

February 6, 2021

Prepared by,
Soundarya Srinidhi
245939

Supervised by,
Dr inż. Mateusz Cholewiński

Abstract

In recent years the research on Mars surfaces and the study of the planet as a whole has increased. With years passing by the interest to learn and research more about the composition of the ground surface, that seems to show any slight evidence of life has made scientists and many other science enthusiasts very curious.

With this motivation I decided to develop a project that closely connects to the development of this similar idea by using methods of Computer vision and basic AI to try and implement a strategy to perform automatic shape detection for Mars satellite images. The data of the images collected are a combination of both Mars rover and satellite images.

By performing shape detection, the algorithm identifies regions of interest that need further investigation. It is able to detect regions such as craters, rockets, and sand dunes. After making observations of these regions of interest the user, who is the scientist will be able to extract this information to be able to study the images in much more detail and be able to determine what is unusual about this feature, if incase it is a mars microbe or not.

This project is presented with the conceptual design of the working of the algorithm.
With the use of edge detection and performing many computer vision manipulations on the images, the contours in the image are detected and the areas of interest within the image are labelled for further investigation by scientists.

Table of Contents

Introduction

Object detection is a computer technology which makes uses of Computer vision and image processing methods to detect shapes that belong to particular classes/ characteristics. For example object detection of rocks, cars, humans etc.

With the use of many functions in OpenCV this project includes ways in which shape detection can be performed from scratch to develop the image analysis algorithm. This project has been conducted as part of the Artificial intelligence and Computer vision course. This idea of detecting objects within collected mars images was adopted through the help of various supporting case studies.

Many OpenCV functions have been put to use in the process of developing this project. The skills of using these functions were obtained through different lab coursework over the semester, of which helped to be further modified and applied to this project.

The software can be used in the following ways; it is able to identify regions of sharp changes in pixel intensity which is what helps in obtaining the edges of the images and the algorithm later on works in such a way to find particular regions of unusual shapes that needs further investigation by performing findCountours which is part of the OpenCV library.

Case study of similar projects

As mentioned previously this project's motivation was adopted from various different case studies which developed into a single idea, which was to develop an algorithm that performs image analysis and measurement of certain identified shapes on the mars images.

In this section of the document various case studies are briefly presented which closely relate to the project and ideas that lead to the development of this project.

Originally this project was aimed at developing an algorithm to perform automatic crater detection by making use of the hough transform built in function in OpenCV. Upon continued analysis and experimentation it was realized that it would be much better to be able to develop an algorithm that would perform automated image search in a given image in order to perform image analysis and measurement of particular found objects.

Case study 1: Automated detection of geological landforms on Mars using Convolutional Neural Networks

This project made use of the CNN in order to be able to train the given data set to perform automated detection, which is making use of the machine learning applications.This project was mainly targeted at analysing the mars crater and volcanic landforms. They have made use of a sophisticated approach to obtain their desired results which is SVM. SVM support vector machines which are supervised learning models associated learning algorithms that analyze data for classification and regression analysis.

Upon training the network using the data set of a fixed sample of images, the resulting image is the one with the highest likelihood of the crater being present in the image.

The measures in this algorithm are able to detect positive examples, that is the occurrence of landform/ crater of interest. For example: the given data set with 5 positive points and 100 negative examples has the accuracy of 95%.

As described this project works in a very sophisticated manner, although the idea of being able train the data set to automatically perform detection surely is very promising,although it may not always provide very accurate results.

Case study 2: Automated crater detection

Through this case study, I was able to extract and make applications into my own project with the OpenCV libraries.

This project was aimed to be able to perform automated crater detection with the use of the Hough transform function and collected data such as the ground truth.
The main aim of this algorithm was to solve the problem of manually counting craters on the surface of a planetary body such as mars. This software makes use of edge detection and hough transform to automatically detect craters that are sub-km sizes.

Hough transform is an OpenCV built in function that is used for detecting lines and circles. After the implementation of Canny edge detection they were able to test this on a set of different high quality Mars images. 15 regions were cropped out from the images out of which there were craters detected by making use of a predefined radius of 5 pixels, reason being the image limits the ability to resolve crater features under the edge detection.

Here is a clipping of the analysed data:

Table 1: Classification for detections with ground truth (GT) tests.

	Detected Circle	No Detection
Matching GT Crater	TP	FN
No Matching GT Crater	FP	TN

More information can be obtained from the references regarding this case study.

In my project, the use of edge detection and experimentation with hough transform surely did help to develop the algorithm, due to lack of accuracy and difficulty in prediction of the radius of all the craters increased the complexity to make use of hough transform in the project.

Problem statement

Here the image analysis algorithm shall be looked at much more closely, exploring the main reasons for this idea.

The main goal of this project is to try and find solutions to improve the speed and accuracy for detecting any anomalies in images. It can be very time consuming and tedious to sit down all day long searching for interesting features such as rocks, sand grains, or even try to and regions that could possibly miss the human eye.

BY applying various image analysis techniques in Computer vision using the OpenCV library , features such as edges can be extracted so that the image does not just look like a large brown image, instead it it cover specific edges which would be important for analysis and it is easier for the human eye to be able to differentiate between regions.

Since feature extraction can be made used to extract various results of the detected feature/ region, it can be applied to obtain different information about an image. It has also been made in applications of cancer cell detection and analysis.

The way I made use of image analysis is by automating the detection method. Once there is a specific region of interest detected after performing all the computer vision image analysis techniques, a window is created around the image that specifies the number of points it covers and which can help to make a prediction of the shape of the image and this can be taken into further research and analysis of the detected shape in different regions.

Although this is a very basic algorithm it does the job with respect to this particular project. There are better ways to achieve this by making use of other machine learning algorithms such as YOLO and tensorflow.

Limitations

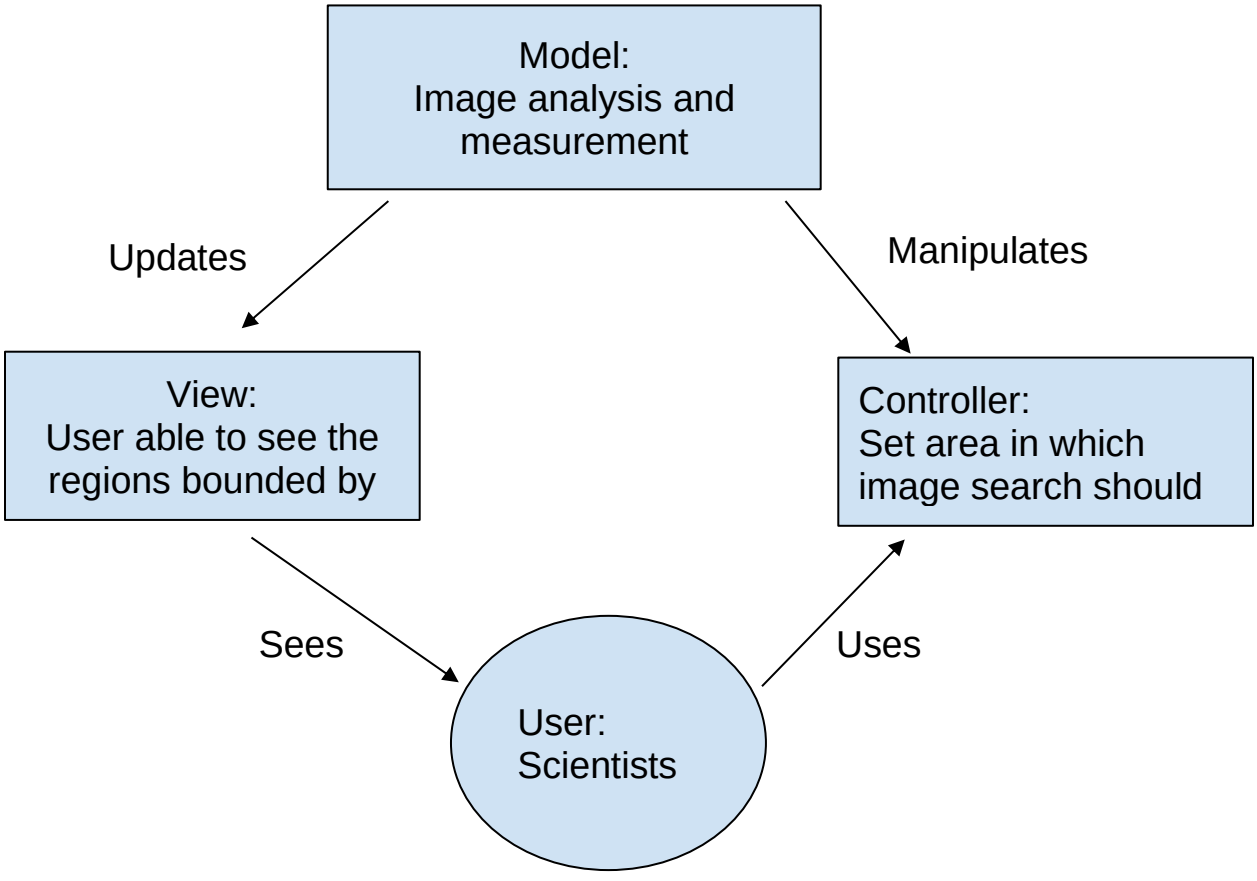
The limitation that was faced while developing this project was not having the technical requirements to be able to develop a much more sophisticated AI trained algorithm. Initially the goal was to make use of tensorflow to be able to perform automated detections, due to the absence of Nvidia graphics card, this was not able to be fulfilled. However the project found alternative directions to explore and through quite a lot of testing and experimentation of different uses of OpenCV library a simple image analysis algorithm was able to be developed.

Scope of design

The overall design of the system is built using Opencv and its libraries. The architecture demonstrates the steps in which this algorithm was developed and later is broken down into a simple explanation of how each part of the architecture works in detail and how the whole system interacts with one component. Incase of a missing component the algorithm would fail and that will be explained in the system architecture.

System architecture

The MVC model helps to better describe the workflow of the whole system and how the software architecture is divided into interconnected elements.

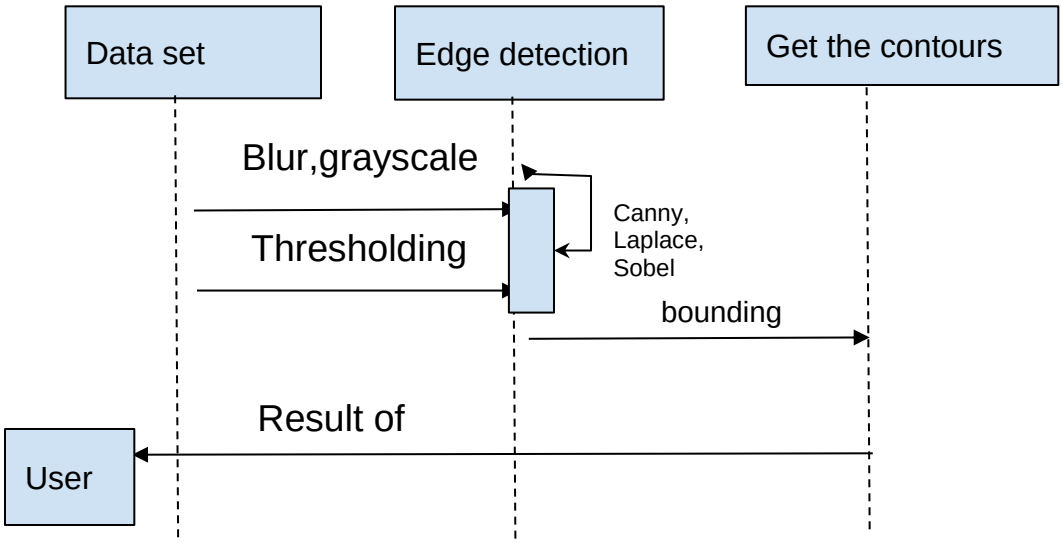


System interaction

The use of sequence diagrams has been put to use here in order to be able to explain the workflow of the functionality of how all the components work together. For this particular software design the, the sequence of the compliments is what has been put to use. Since the overall idea is to be able to use the cv2.findCounturs function in OpenCv. The general working of findCountours makes use of the following methods in the following sequence:

-Blurring the image
-Converting image to greyscale
-Thresholding the image
-Performing edge detection
-Dilation of the image
-Getting contours of image

Sequence diagram



The sequence above shows how there is a codependency between each of the components in order for the final component to work the way it should work, and finally it displays the final result of the image to the user with bounding rectangles that have been labelled.

Development of system

Data augmentation

In general, augmentation refers to the method of increasing the size of the data set by morphologically transforming the image when there is not enough data to train. As mentioned earlier, with the idea kept in mind of making use of machine learning techniques the data augmentation was a required step, later upon realization due to lack of tools the data augmentation was put to a good use in the project as described below.

In order to begin with the project, the requirement of generating a large data set was a good idea, so that there would be more images for analysis without the need to search for over 40 images on the internet.

Step 1:

Making use of OpenCV libraries made it easy to be able to develop a dataset by performing different manipulations of the images such as:

-Resize

The image can be scaled outward or inward. While scaling outward, the final image size will be larger than the original image size. In this application the image has been scaled inward.

```
def random_rescale(image_array: ndarray):
    return sk.transform.resize(image_array,(200,500))
```

-Rotation

One key thing to note about this operation is that image dimensions may not be preserved after rotation. If your image is a square, rotating it at right angles will preserve the image size. If it's a rectangle, rotating it by 180 degrees would preserve the size. Rotating the image by finer angles will also change the final image size. For that reason random rotation has been used

```
def random_rotation(image_array: ndarray):
    # pick a random degree of rotation between 25% on the left and 25% on the right
    random_degree = random.uniform(-25, 25)
    return sk.transform.rotate(image_array, random_degree)
```

-Adding noise

Gaussian noise, which has zero mean, essentially has data points in all frequencies, effectively distorting the high frequency features. This helps to reduce the unwanted contours in the images.

```
def random_noise(image_array: ndarray):
    # add random noise to the image
    return sk.util.random_noise(image_array)
```

-Vertical and horizontal flip

Horizontal shift /Vertical shift or translation is shifting the image left or right based on a ratio that defines how much maximum to shift.

```
def horizontal_flip(image_array: ndarray):
    # horizontal flip doesn't need skimage, it's easy as flipping the image array of pixels !
    return image_array[:, ::-1]

def vertical_flip(image_array: ndarray):
    # horizontal flip doesn't need skimage, it's easy as flipping the image array of pixels !
    return image_array[ ::-1,:]
```

-Gamma corrected

First, our image pixel intensities must be scaled from the range *[0, 255]* to *[0, 1.0]*. Gamma values < *1* will shift the image towards the darker end of the spectrum while gamma values > *1* will make the image appear lighter. Where the random.unifrom() generates 1-gamma and 1+gamma.

```
def brightness(image_array: ndarray , gamma=1,gain=1):
    gamma=random.uniform(1-gamma,1+gamma)
    gamma_1=exposure.adjust_gamma(image_array,gamma,gain)

    return gamma_1
```

-Contrast adjustment

In this kind of image processing transform, each output pixel's value depends on only the corresponding input pixel value (plus, potentially, some globally collected information or parameters).

Examples of such operators include *brightness and contrast adjustments* as well as color correction and transformations.

```
def change_contrast(image_array : ndarray):
    v_min, v_max = np.percentile(image_array, (2.0, 80.0))
    better_contrast = exposure.rescale_intensity(image_array, in_range=(v_min, v_max))
    return better_contrast
```

The added advantage to making use of data augmentation was the fact being, being able to identify if adding noise or rotating, resizing the image would have any different effects to the final output result. The analysis of the results shall be discussed later in the documentation.

As it can be observed, the use of various OpenCV library functions have been used. Making use of the skimage library simplified performance data augmentation, making the code much more efficient with also fewer lines of code, making it easy to understand by other programmers.

```
import random
from scipy import ndarray
from skimage import img_as_ubyte
from skimage import exposure
import numpy as np
import glob

# image processing library
import skimage as sk
from skimage import transform
from skimage import util
from skimage import io
```

We use scipy.ndarray to represent the image to transform. This data structure is convenient for computers, as it’s a two-dimensional array of image’s pixels (RGB colors).

Step 2:

After making the different image transformations the next step was to list all the files in a folder which was named “data set” and read them. For this project it was enough to just generate 20 (line 70) images for the initial testing purposes, although the desired number of images can be increased later on.

```
69  folder_path = '/home/sound/Downloads/dataset'
70  num_files_desired = 20
71
72  # find all files paths from the folder
73  images = [os.path.join(folder_path, f) for f in os.listdir(folder_path) if os.path.isfile(os.path.join(folder_path, f))]
74
75  num_generated_files = 0
76  while num_generated_files <= num_files_desired:
77      # random image from the folder
78      image_path = random.choice(images)
79      # read image as an two dimensional array of pixels
80      image_to_transform = sk.io.imread(image_path)
81      # random num of transformation to apply
82      num_transformations_to_apply = random.randint(1, len(available_transformations))
83
84      num_transformations = 0
85      transformed_image = None
```

After performing the iterations, a random file from the folder is chosen (line 78) and read it with skimage.io.imread, which read images as a scipy.ndarray by default (line 80).

Step 3:

This is where all the different calls to the functions are made by creating a dictionary of transformation images. The number of transformations are chosen for a single image and the kind of transformations to apply (line 89). Then we just call the function defined in our transformations dictionary (line 89).

```
52  # dictionary of the transformations we defined earlier
53  available_transformations = {
54      'rotate': random_rotation,
55      'noise': random_noise,
56      'resize': random_rescale,
57      'vertical flip': vertical_flip,
58      'horizontal_flip': horizontal_flip,
59      'gamma corrected':brightness,
60
61      'contrast adjust':change_contrast
```

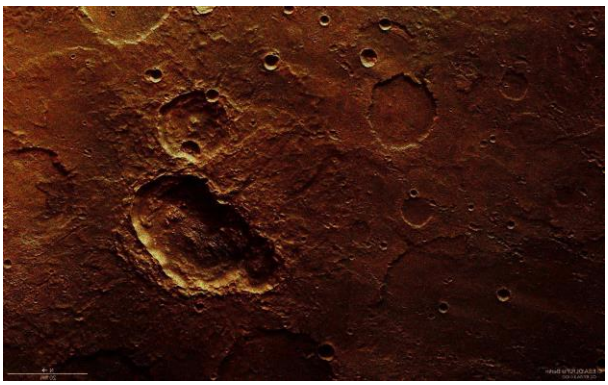
```
86      while num_transformations <= num_transformations_to_apply:
87          # random transformation to apply for a single image
88          key = random.choice(list(available_transformations))
89          transformed_image = available_transformations[key](image_to_transform)
90          num_transformations += 1
91
92          new_file_path = '%s/augmented_image_%s.jpg' % (folder_path, num_generated_files)
93
94          # write image to the disk
95          io.imsave(new_file_path, img_as_ubyte(transformed_image))
96
97      num_generated_files += 1
```

Step 4:

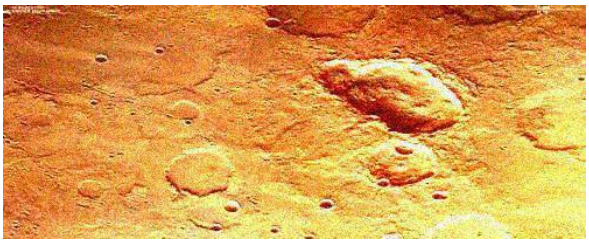
Saving the new images using io.save in the file path defined previously.

```
92          new_file_path = '%s/augmented_image_%s.jpg' % (folder_path, num_generated_files)
93
94          # write image to the disk
95          io.imsave(new_file_path, img_as_ubyte(transformed_image))
96
97      num_generated_files += 1
```

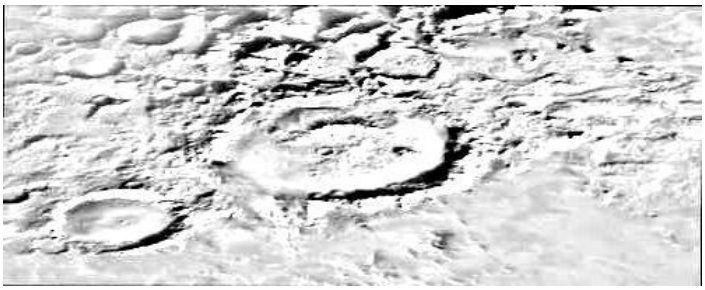
Results of the some of the augmented images from generated data set:



[Figure 1: Contrast]



[Figure 3: Flipped]



[Figure 2: Gaussian contrast]

[Figure 4: Resize]

Edge detection

This is yet another image processing technique used to find the boundaries or edges of the image, by determining where the brightness of the image changes dramatically. As mentioned previously, the reason edge detection was adopted for this project was to be able to extract the structure of objects in an image in order to be able to indicate to the user viewing the results of the detected image to make further observations based of the the extracted areas of interest.

Since we were interested to find the number,size and relative location of objects/shapes in the image, edge detection allows us to focus on the parts of the image which are most useful, ignoring parts of the image that will not help us.

Basic math behind edge detection

In grayscale images the edges can be detected based of the information of the regions of changing intensities by measuring rate of change also known as the gradient magnitude.This magnitude is what determines an edge, where g> pixels = 0 dropping all the other pixels from being identified as edges.

$$Edge\_Gradient\ (G) = \sqrt{G_x^2 + G_y^2}$$

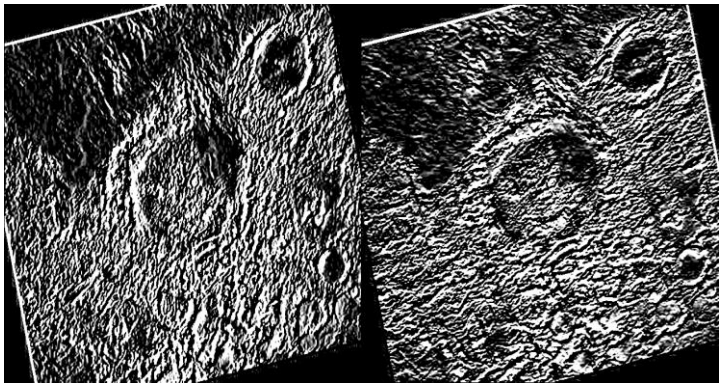
Sobel edge detector

Sobel detector is a gradient based method that uses the first order derivate on the image pixels in the kernel separately in the x axes and in the y axes. The operator uses two 3x3 kernels which are convolved with the original image to calculate approximations of the derivatives.

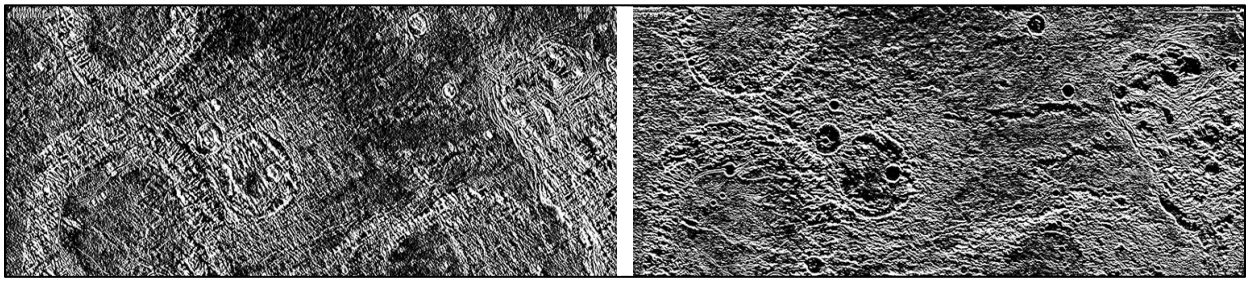
```
def sobel_x_y(image):
    sobelx = cv2.Sobel(image, cv2.CV_64F, 1, 0, ksize=5)  # x
    sobely = cv2.Sobel(image, cv2.CV_64F, 0, 1, ksize=5)  # y
    res=np.hstack([sobelx, sobely])

    return res
```

Results:

From the results we can see that the edges have been extracted and we can notice a difference, although not a very big difference between the images, is that



[Figure 1: Right(sobel x), Left (sobel y)]



[Figure 2: Right(sobel x), Left (sobel y)]

Laplacian edge detector

This is the 2nd order derivative detector, upon comparison with the sobel edge detector the laplacian seems to yield better results in edge extraction. Although, the laplacian detector is very sensitive to noise.

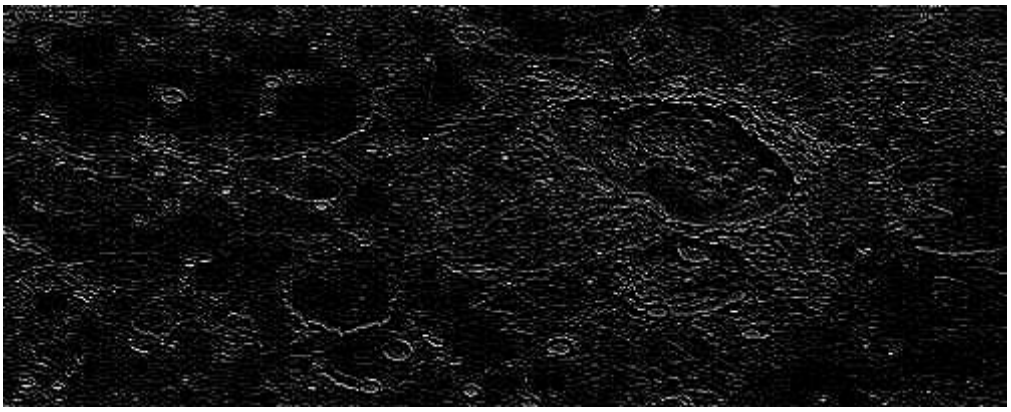
$$L(x,y) = \frac{\partial^2 I}{\partial x^2} + \frac{\partial^2 I}{\partial y^2}$$

Since the laplacian detector is very sensitive to noise, gaussian smoothing is performed to reduce high frequency noise components before performing the differentiation step.
Unlike the sobel detectors laplacian uses only one kernel to calculate the second derivative in a single pass:

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

```
def laplace(image):
    img = Gausblur(image)

    return cv2.Laplacian(img,cv2.CV_64F)
```

Results:



[Figure 1: Edges visible]

[Figure 2: Edges visible, low clarity due to high noise in image]

Canny edge detector

The canny edge detector is an algorithm that detects edges for any input image.

This method is said to be the best edge detector as compared to laplacian and sobel and this can further be analysed in the results.
This algorithm uses hysteresis thresholding which takes the pixels higher than the gradient magnitude and neglects the ones that are below the threshold value.

There were 3 main conclusions that were made about the algorithm:

- Good detection: Eliminates the possibility of getting false positives and false negatives.
- Good localisation: Detected edges are close to the true edges.
- Constraint: THe detector must return one point only for each edge point.

OpenCV puts all of the following functions into one cv.Canny()

Noise removal:

Before performing the canny edge detection it is important to perform image smoothing. The pixel may not be close to its similar neighbouring pixels. This could result in improper and inappropriate edge detection. As observed previously with the Laplacian edge detector. It has been observed that the edges were not detected very well, this is due to the fact that the images had high frequency noise that even the gaussian filter could not reduce to the required amount.

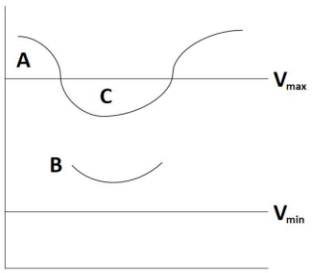
Derivative:

Calculates the derivative of filter in the X and Y dimensions and convolve it with I to give the gradient magnitude along the dimensions.

Non-max suppression

THe canny edge performs non-maximum suppression which eliminates thin edges by only considering pixels with g values that are local maximums.

Hysteresis thresholding:



If the gradient is g > Cmax = edge pixels
If g< Cmin = non edge pixel
Cmin<g<Cmax = low and high

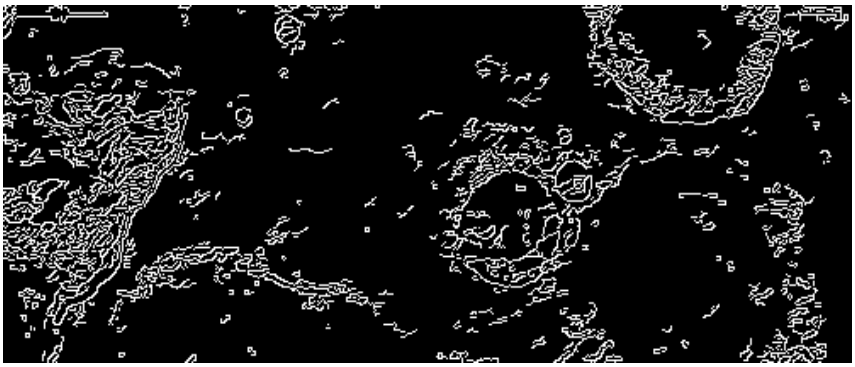
With the perspective of the development of the project, canny requires less image alterations to be carried out before performing edge detection, and it is a well tested algorithm which is used in many various applications for image analysis.

There is ne problem with using the canny edge detector, it is difficult to assume the upper and lower threshold values of the pixels.

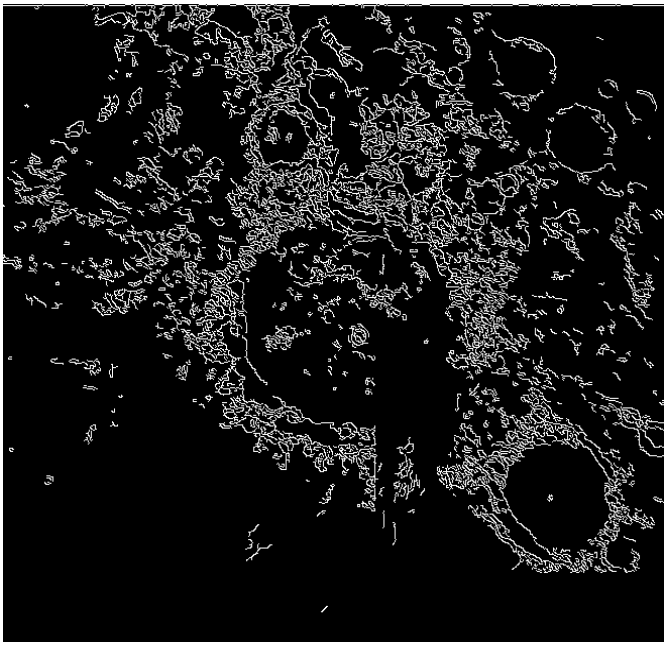
A function can be defined to automate the edge detection.
Sigma is can be used to vary the percentage thresholds that are determined based on simple statistics.
Np.median computes the median of the pixel intensities in the image.
The lower values of sigma indicates tighter threshold, whereas the larger value of sigma gives a wider threshold. The optimal value of sigma for obtaining good results is 0.33.

```
def auto_canny(image, sigma=0.33):
    # Compute the median of the single channel pixel intensities
    v = np.median(image)

    # Apply automatic Canny edge detection using the computed median
    lower = int(max(0, (1.0 - sigma) * v))
    upper = int(min(255, (1.0 + sigma) * v))
    return cv2.Canny(image, lower, upper)
```

Results:



[Figure 1: Clear edges]



[Figure 2]

Observation of edge detection

After experimenting with the various methods of edge detection, observation was made that the canny edge detector works the best in comparison to the sobel and Laplacian. This is because the canny detector is an algorithm with combines the Laplacian and sobel to determine the best and worst edges through the method of hysteresis thresholding and non- maximum suppression. For the next steps Canny detector was the chosen detector to be used in the image analysis algorithm in this project.

Contour detection workflow

The main goal of this part of the system if to combine all the methods used above and apply it to find contours in the image to be able to perform shape detection.

The basic working principle of finding contours in OpenCV is that it detects changes in the image color and marks it as a contour.

cv.findContours(`image,cv2.RETR_EXTERNAL,cv2.CHAIN_APPROX_NONE`)

Find contours function retrieves all the contours in the image that it can find. To make it easy to find contours which we are interested in the use of “Retrieval modes” are important.

The output of RetrievalModes is array Hierarchy which shows how various contours are linked to each other, their relation with other contours, Parent Child relation.

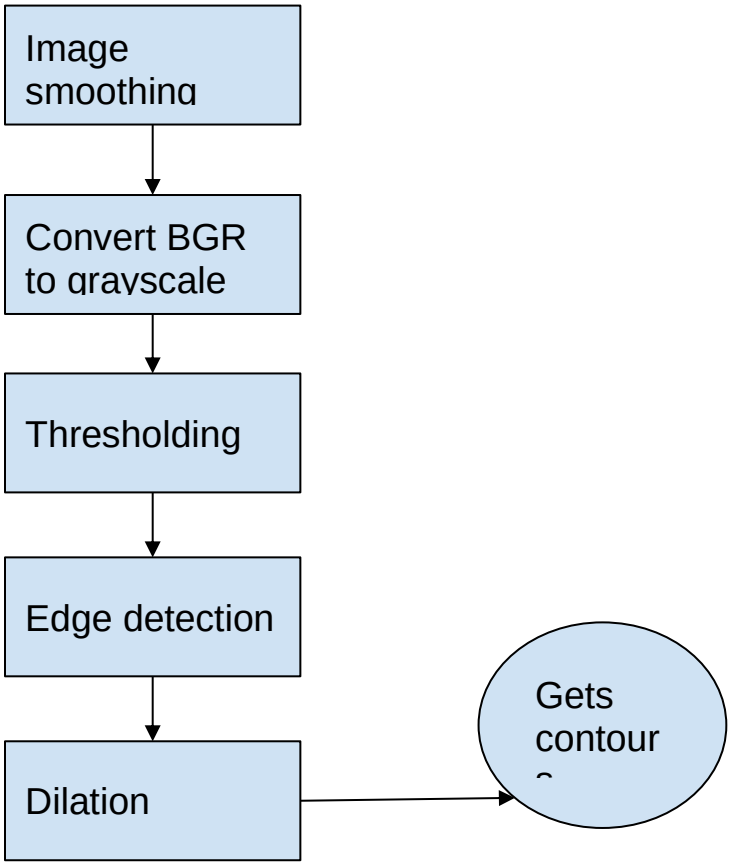
In this project the **RETR_EXTERNAL** has been used, which retrieves only the extreme outer contours. With this application no parent, child relationship is given. All the external contours are at the same hierarchy level.

Parameter- Contour approximation

There are namely two main approximation methods to store the values of the found contours.
CHAIN_APPROX_NONE stores all the contour points.
CHAIN_APPROX_SIMPLE compress horizontal, vertical and diagonal segments and leaves only their end-points.

For the scope of this project it was better to use CHAIN_APPROX_NONE, since the image that is being analyzed is of a much larger scope and the main goal is to be able to find all possible “interesting” contours.

Work flow:



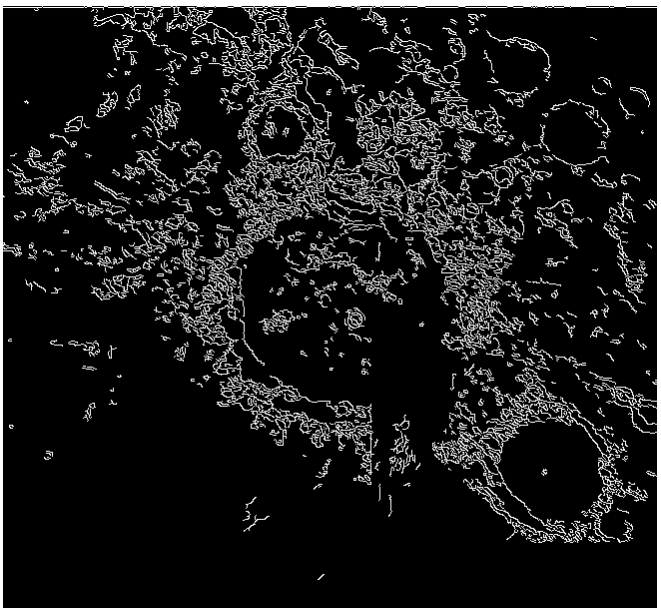
Why use dilation?

Dilation adds pixels to the boundaries of objects in an image. This is applied to binary images, due to which thresholding of the images must be first performed.he basic effect of the operator on a binary image is to gradually enlarge the boundaries of regions of foreground pixels (white pixels). Areas of foreground pixels grow in size while holes within those regions become smaller.

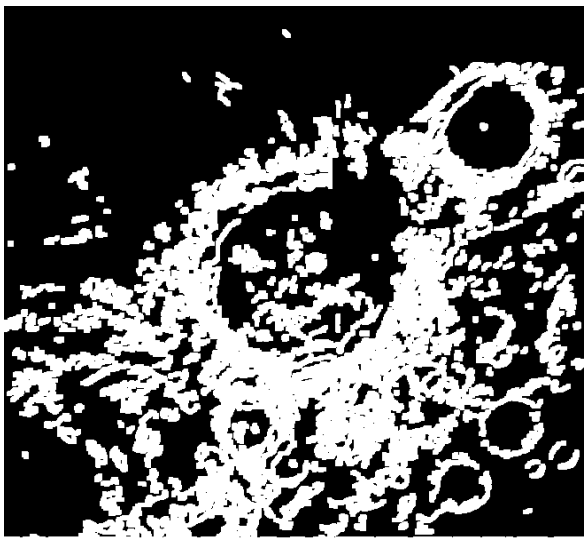
After performing edge detection it is a good idea to use this dilation method, just so that the boundaries look much more defined and after performing the find contours algorithm, the observation is much better.

```
def dilate(image):


    kernel = np.ones((5, 5), np.uint8)
    dilate1 =cv2.dilate(image, kernel, iterations=1)

    return dilate1
```

Results:



[Figure 1: Canny image]



[Figure 2: Dilated image]

As observed it is clearly seen that the boundaries are much more clearly visible, because the foreground pixels have been enlarged.

Steps for shape detection in images:

Step 1: Reading a selected image from data set

Step 2: Adding gaussian blur to image

Step 3: Converting gray to BGR

Step 4: Performing edge detection using the auto_canny function

Step 5: Dilation function

```
248 while True:
249     #sample images taken
250     image = cv2.imread('augmented_image_10.jpg')
251
252     imgcontours = image.copy()
253
254     imgblur= Gausblur(image)
255     imggray=cv2.cvtColor(imgblur,cv2.COLOR_BGR2GRAY)
256     threshold=binary_thresh(imggray)
257     imgcanny=auto_canny(image,sigma=0.1)
258     imdil=dilate(imgcanny)
259     getcontours(imdil,imgcontours)
260     cv2.imshow("output", np.hstack([image, imgcontours]))
261     cv2.waitKey(0)
```

Step 6: Getting contours

In this step a function of contours is written. Line 231 uses OpenCV contours of the image. To display the contours the draw() function is used line 237, the contours are drawn on the image.

We can detect the area of our contours and remove any unwanted contour areas by creating a for-loop.
To look into the area of the function in line 234.
The contours are only drawn if the area is above a certain defined threshold (line 236), then the contours are drawn. Making use of this eliminates the noise by using the area.

The corner points can be found. In order to do this the length of the contours are obtained. This can be done by using the arclength function line 238, where “TRUE” means that the contour is closed. We will use the length of this perimeter to approximate the type of shape detected.

The approximation helps to achieve the above, with the use of cv.approxPolyDP() function with a given resolution of 0.06*peri in a closed contour, line 239. This approximation array will have certain amount of points, and based one these points we can determine if it is a square, rectangle, circle. The length of these contours are printed, line 240.

Although in this project the goal is not to find perfect squares and perfect rectangles, so the next best step is to draw a bounding box that highlight the area of where the object is, line 241 and it is displayed using a rectangle on the image contour, line 242.

X,y are the initial points, and x+w, y+w is the final corners of the bounding box.

To display the values, making use of the cv.putText to show the points and area of the shape detected on the image, lines 244, 245.

```
227 def getcontours(image,imgcontours):
228
229
230
231     contours,hierachy = cv2.findContours(image,cv2.RETR_EXTERNAL,cv2.CHAIN_APPROX_NONE)
232 #to remove the unwanted contours
233     for cnt in contours:
234         area =cv2 .contourArea(cnt)
235         #areaMin=cv2.getTrackbarPos("Area","Parameters")
236         if area>300:
237          cv2.drawContours(imgcontours,contours,-1,(255,255,255),7)
238          peri = cv2.arcLength(cnt,True) #getting length of contour
239          approx = cv2.approxPolyDP(cnt,0.06*peri,True) # approximating the type of shape
240          print(len(approx)) # gets the number of points in the array
241          x , y , w, h = cv2.boundingRect(approx)
242          cv2.rectangle(imgcontours,(x ,y ),(x + w , y + h ),(0,255,0),5)
243
244          cv2.putText(imgcontours, "Points:"+str(len(approx)),(x+w+20,y+20),cv2.FONT_HERSHEY_COMPLEX,.5,(0,255,0),2)
245          cv2.putText(imgcontours,"Area:" + str(int(area)),(x+w+20,y+45),cv2.FONT_HERSHEY_COMPLEX,0.5,(0,255,0),2)
246
```

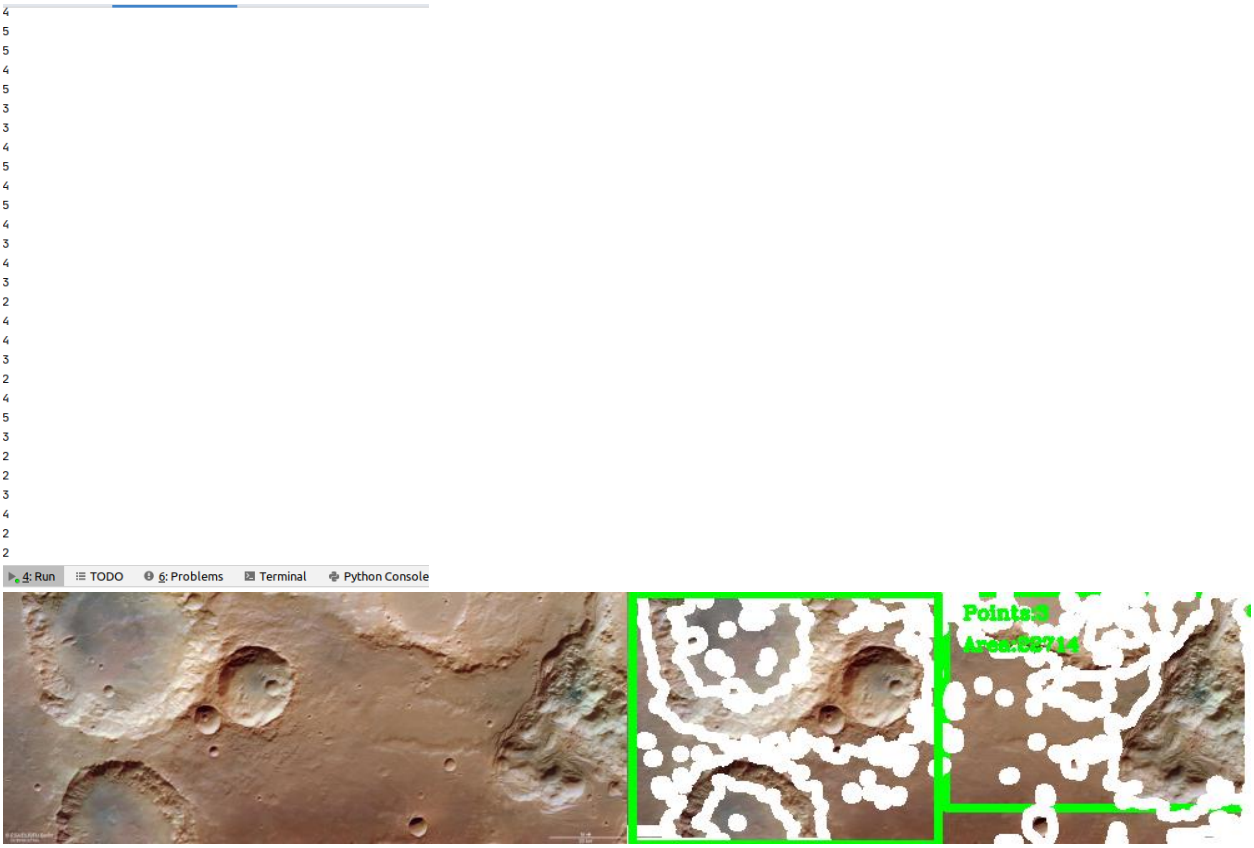
Finally to print the images for the user to see it, the np.stack() is used to show the original image and the image with the found contours side by side.

```
getcontours(imdil,imgcontours)
cv2.imshow("output", np.hstack([image, imgcontours]))
cv2.waitKey(0)
```

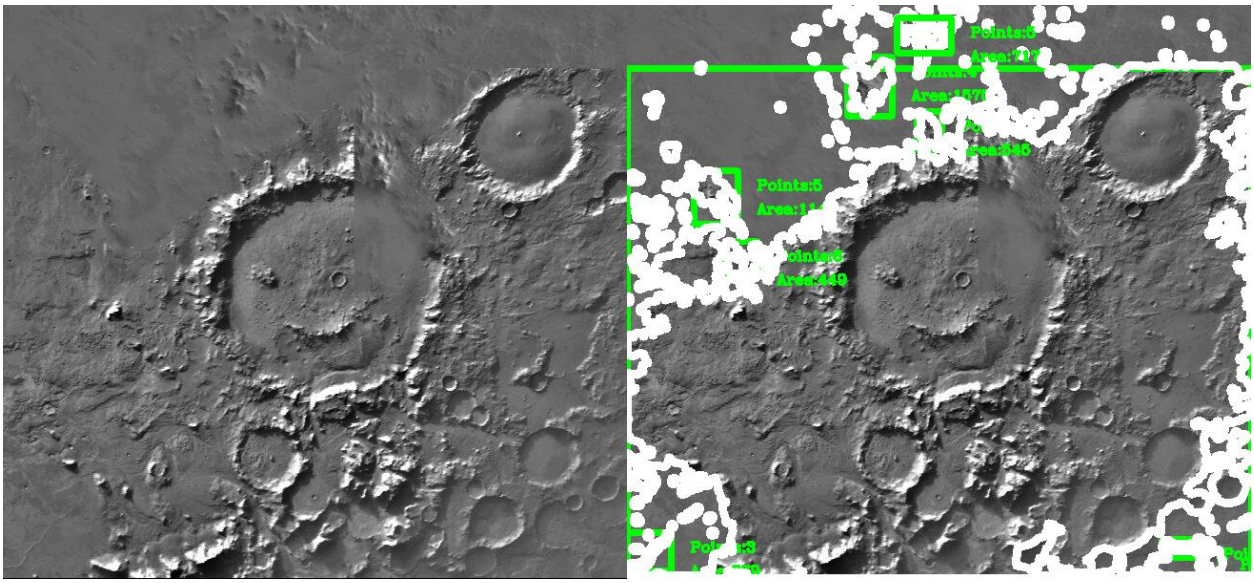
Experimental results:

From the observations below, the algorithm does not yet work perfectly, as there are some unnecessary contours that are still being detected. In the scope of this project different regions of interest in the image are extracted, with specification the area and the corner points, with this the shape of the object detected can be approximated.

Printed approx length of contours



[Figure 1: Simple images, displaying , craters of different sizes]



[Figure 2: Large are full of craters detected, smaller regions, which look like sand grains detected]

Summary

Evaluation of project

As mentioned in the introduction, the approach of this project was in the aim of creating an algorithm that has applicability in the real world with the intent to improvise image analysis techniques in the space industry.

Upon completion of the project, here is an analysis.

After using various methods of edge detection and deciding to choose canny edge detection with experimental proof, it happened to have a good impact over the project, as the desired results were able to be obtained.

However, there are still some issues that this algorithm faces, the smaller areas of interest are not able to be seen very clearly so the thickness of the contours could be decreased for future applications, also some unnecessary contours were detected, and this could be eliminated by the use of trackbars where the user can manually alter the area in which the image analysis is to be performed.

Some limitations were encountered in this project, due to the lack of Nvidia graphics card. As mentioned earlier the main goal of generating a data set was to be able to use it to train a model to be able to perform automated shape detection with the use of tensorflow, or YOLO, and a much more sophisticated algorithm could have been generated. For example: detecting the regions of interest and also performing analysis on them by providing the actual nature of the object, if it is a carter, sand dune, sand grain or even a Martian microbe!

Nevertheless, this project shows various ways in which shape/ object detection can be performed from start just by using OpenCV libraries. It is also possible to build your own CNN from scratch just by using numpy arrays. This project did not make use of that method due to time constraints and would lead to time overkill in the perspective of the project.

The future holds new opportunities, with the skills learned in this project it will be possible to be able to develop a much more sophisticated algorithm. It would be interesting to develop such a system where the Mars rover is able to perform this shape detection in real time and send the information to the space centers on earth.

Link to whole code: https://github.com/Sound245939/AICV.git

References

https://towardsdatascience.com/farm-segmentation-from-satellite-images-using-holistically-nested-edge-detection-63454a24b164

https://docs.opencv.org/master/d4/d73/tutorial_py_contours_begin.html

https://towardsdatascience.com/complete-image-augmentation-in-opencv-31a6b02694f5

https://projet.liris.cnrs.fr/imagine/pub/proceedings/ICIP-2014/Papers/1569913441.pdf

Kozakiewicz, J. Image analysis algorithm for detection and measurement of Martian sand grains. *Earth Sci Inform* 11, 257–272 (2018). https://doi.org/10.1007/s12145-018-0333-y

https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_canny/py_canny.html

https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_morphological_ops/py_morphological_ops.html

https://homepages.inf.ed.ac.uk/rbf/HIPR2/log.htm

https://medium.com/analytics-vidhya/opencv-findcontours-detailed-guide-692ee19eeb18

https://docs.opencv.org/master/d4/d73/tutorial_py_contours_begin.html