Table of Contents

Introduction: 1
Program outline: 1
myKmeans (X, k):.. 2
getLabel (corner, centroids):.. 2
myplotting(X, labels, Clusters, k): 2
display_features (image, Title, points): 3
Equations used:.. 3
Flow chart: 4
Implementation results with sample data sets: 4
Different values of k :.. 5
Obtained results:.. 7
References: 9

Introduction:

The following project implements an unsupervised Learning algorithm called K-Means. K-Means is a clustering algorithm which has applications like detecting groups of similar visitors to a site, visualization, anomaly detection etc. In the K-means algorithm the k depicts the number of clusters we wish to divide into. At first based on the k value, a k number of centroids is initialized to any random values either from within or outside the given dataset X. Each point is taken and the Euclidean distance between it and the centroids is found and the point is put into the cluster of the centroid that it is closest to. After the clusters are formed new centroids are calculated by taking the mean of the points in each cluster and the process reiterates again with the new centroids. This process keeps iterating unless all the points are put into the same cluster again or the new centroids calculated remain unchanged or simply iterates a specified number of times. The clusters formed in the last iteration are the result.

Although K-means algorithm is an unsupervised algorithm it requires us to have a pre-knowledge about picking the right k value.

Program outline:

My program makes use of the functions in **openCv**, **NumPy** libraries of python for processing the image in order to implement the K-means algorithm. The colour image is first read by making use of the **cv2.imread().**

In order to find the strongest corners in the image, **cv2.goodFeaturesToTrack()** is used. These corners are displayed on the image as small dots by using the **cv2.circle()** function.

The program, consists of four main functions viz.

- display_features (image, Title, points),
- getLabel (corner, centroids),
- myplotting(X, labels, Clusters, k),
- myKmeans (X, k).

In order to start the K-means process the myKmeans (X, k) method is called which takes in the corners and the k- value as input.

myKmeans (X, k):

This method first initializes random number of centroids based on the given k value. Then there is a while loop in which decides how many times the k means process must iterate. This is decided by a threshold condition which is initially set to 0. If inside the while loop the new centroids and the old centroids are same more than 3 times, then the process must end and the program finishes.

There are three main steps inside this while loop : finding the cluster label of each point , sorting the points into clusters according to their respective labels and calculating the new centroids and comparing them to the old ones.

In order to find the cluster labels of each point the **getLabel (corner, centroids),** method is called and then the resulting label for each point is stored in a Label array. After finding all the Labels the points are sorted into different cluster arrays with respect to their labels and then this each array is appended to a Clusters array. The new centroids are calculated for each of these arrays by taking their mean .

New_centroid= [(x1+x2+x3+…xn)/n, (y1+y2+y3+…yn)/n]

Where (xi, yi) are the coordinates of the points int the cluster whose centroid is being calculated.

The new centroids are compared with the old ones to check if they are the same and they are the same after three iterations then the K-means process is stopped and the Clusters are displayed on the image by calling the myplotting(X, labels, Clusters, k) function, else if the centroids are not the same as the old centroids then the K-means process iterates again.

getLabel (corner, centroids):

This function takes in a point and an array od centroids and calculates the distance between the corner and each centroid using the Euclidean distance formula or the L2 norm.

Euclidean Distance (x, y) = sqrt [(x2-x1)^2 + (y2-y1)^2] where (x1,y1) and (x2,y2) are the points whose distance needs to be found.

The NumPy function **numpy.linalg.norm()** is used to calculate this and the distances are stored in a Distance Array form which the index of the smallest value is found and returned as the label of the given point.

myplotting(X, labels, Clusters, k):

This function takes in the set of points, array of Labels , array of Cluster arrays and the value of k. It contains an array of colours from which to choose to draw a circle on the image. Each point in the given X is drawn on the image with different colours from the colours List based on its respective Cluster label using **cv2.circle()** function

After mapping the points on the image a rectangle of the same colour as the points drawn is used to draw around points of the same cluster. This is first done by finding the area of the cluster points using **cv2.boundingRect()** and then the actual rectangle is drawn using the **cv2.rectangle().**

After which the image is displayed using **cv2.imshow()** and **cv2.waitKey(0).**

display_features (image, Title, points):

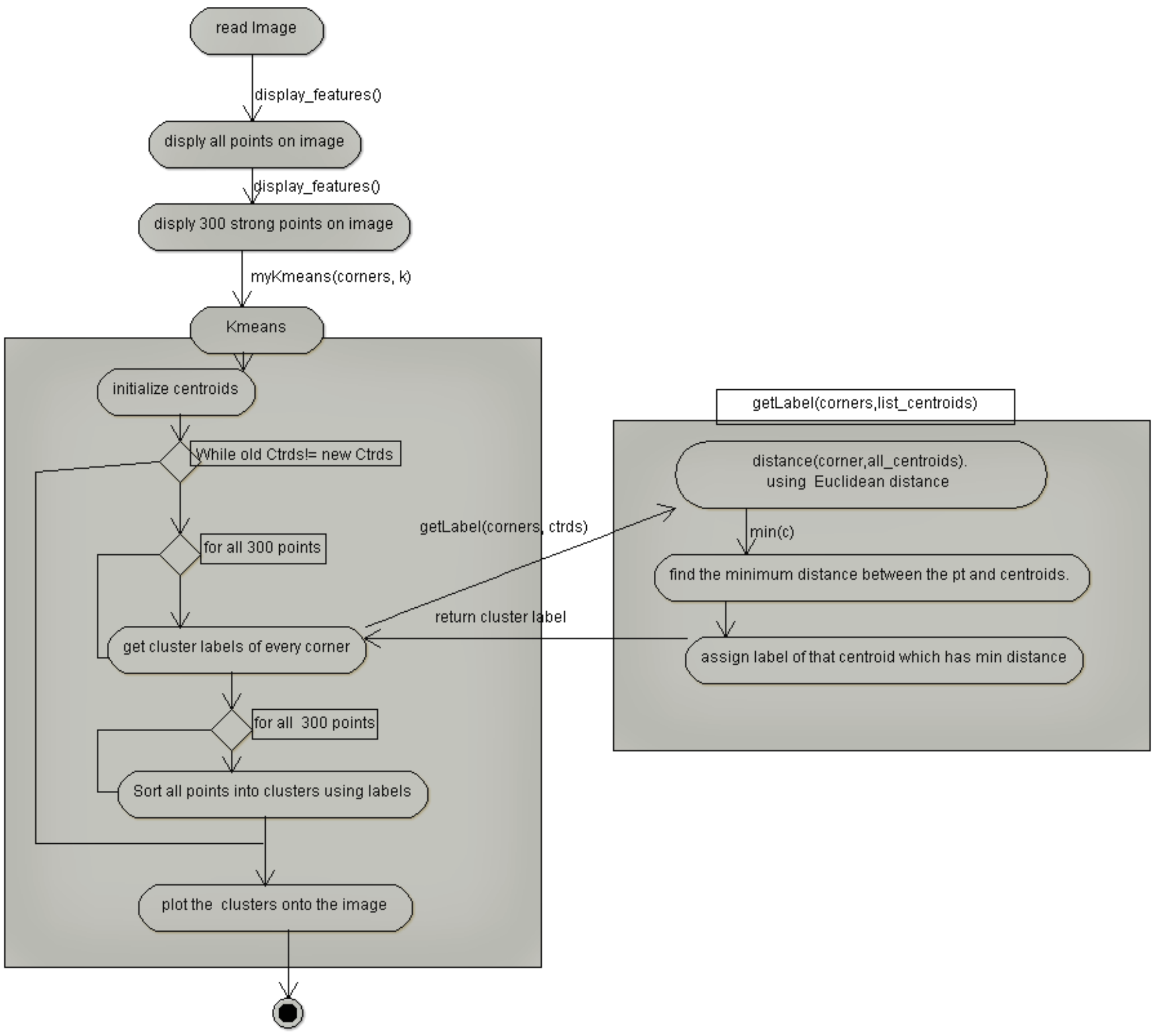
This function is called to display the strongest points on the given image by ,making use of the **cv2.goodFeaturesToTrack()** function**.** These corners are displayed on the image as small dots by using the **cv2.circle()** function. After which the image is displayed using **cv2.imshow()** and **cv2.waitKey(0).**

Equations used:

- Euclidean Distance (x, y) = sqrt [(x2-x1)^2 + (y2-y1)^2] where (x1,y1) and (x2,y2) are the points whose distance needs to be found.
- **New_centroid= [(x1+x2+x3+…xn)/n, (y1+y2+y3+…yn)/n]**

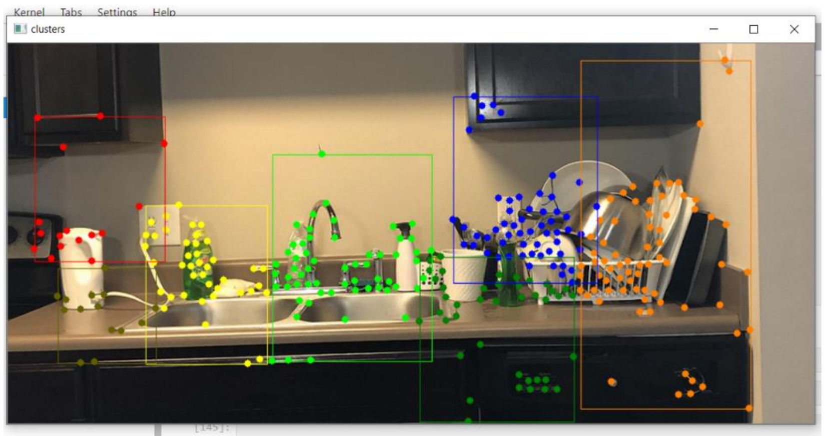
  Where (xi, yi) are the coordinates of the points int the cluster whose centroid is being calculated.
- **numpy.array_equals(New_centroids, Old_centroids)** where the New_Centroids are a list of the newly calculated centroids and the old _centroids are the previous centroids. This is used to decide when to stop the process of K-Means.

Flow chart:

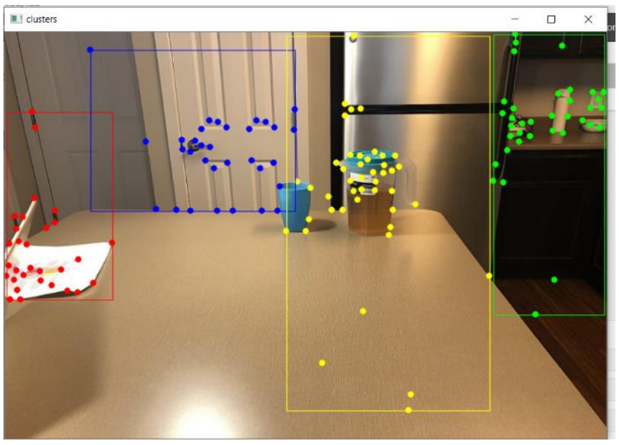

read Image
display_features()
disply all points on image
display_features()
disply 300 strong points on image
myKmeans(corners, k)
Kmeans
initialize centroids
While old Ctrds!= new Ctrds
for all 300 points
get cluster labels of every corner
for all 300 points
Sort all points into clusters using labels
plot the clusters onto the image
getLabel(corners, ctrds)
return cluster label
getLabel(corners,list_centroids)
distance(corner,all_centroids). using Euclidean distance
min(c)
find the minimum distance between the pt and centroids.
assign label of that centroid which has min distance

Implementation results with sample data sets:

The myKmeans() algorithm was run on two other sample images and the results are as follows. We notice that when there are too many objects in the image the algorithm doesn’t yield a perfect result. Also we must know before hand which k is the must suitable for clustering despite this being an unsupervised learning algorithm.



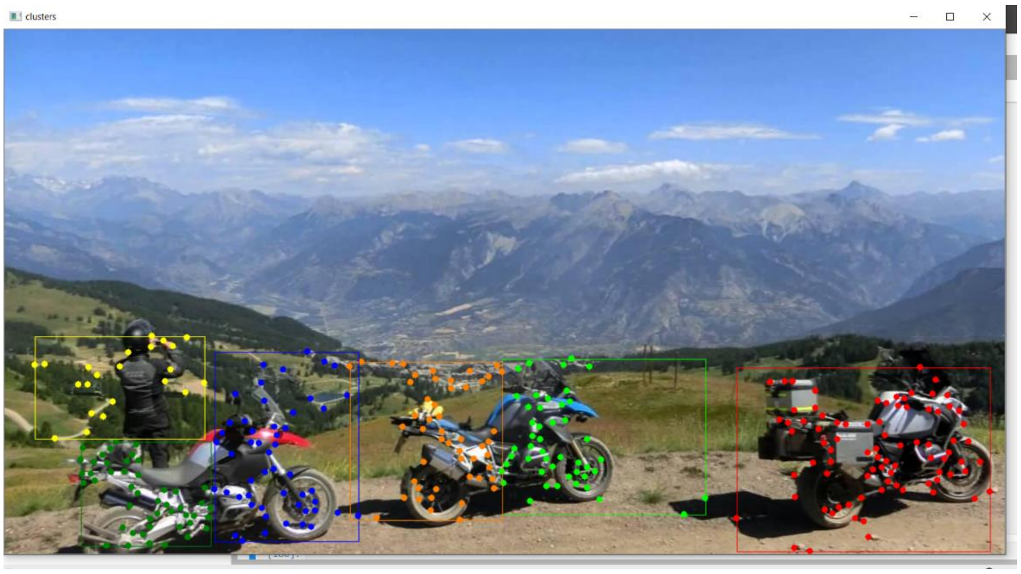
sample test image where k=7



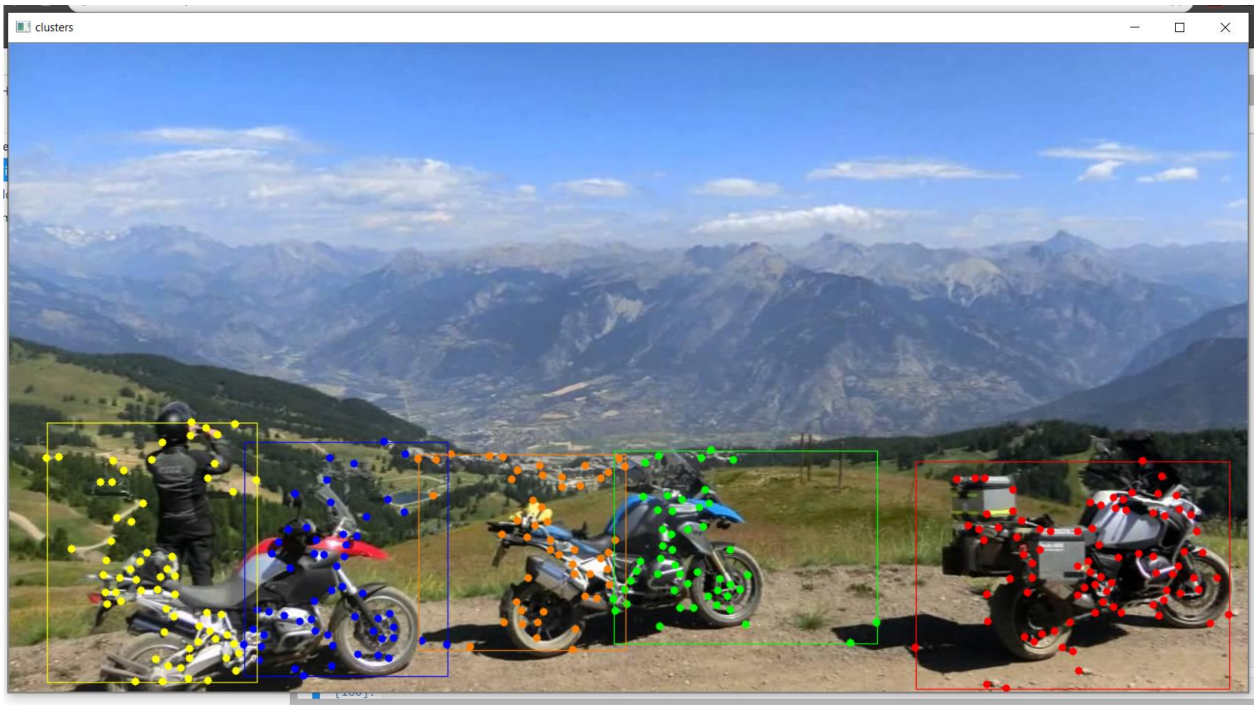
sample test image where k=4

Different values of k :

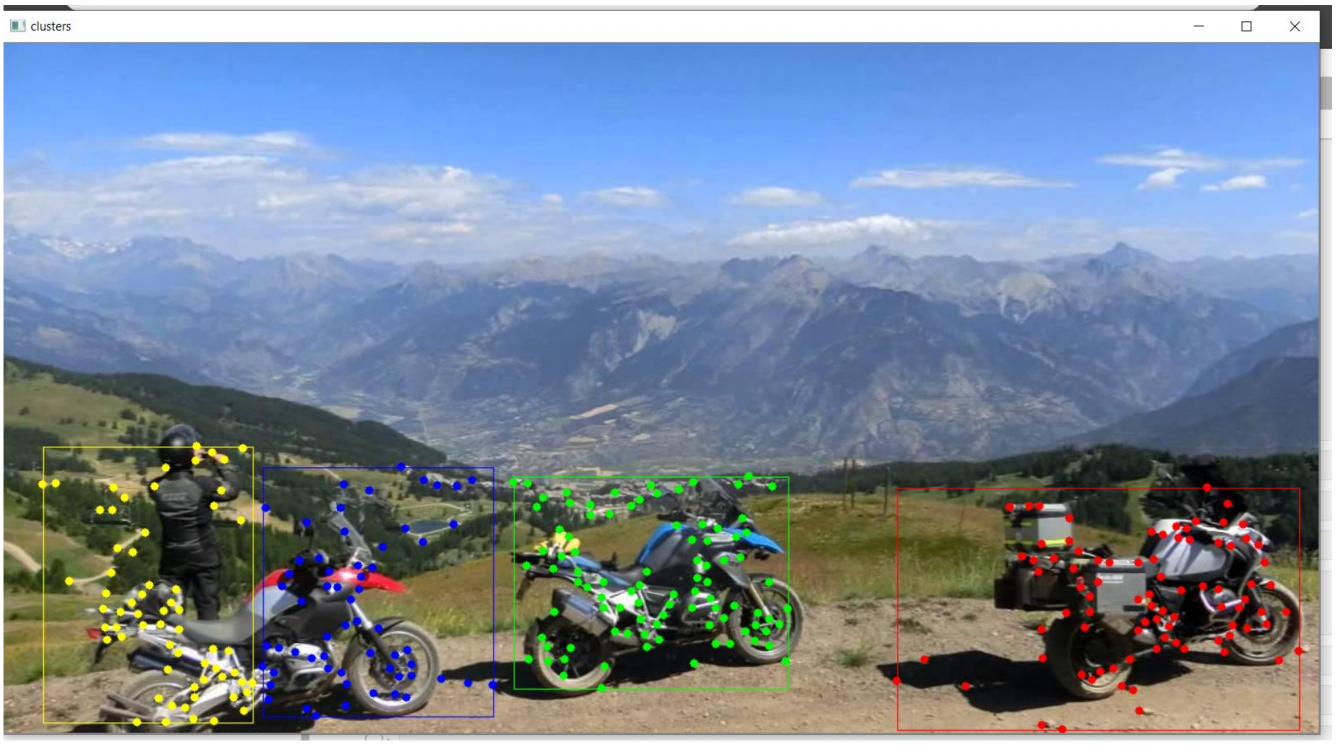
The given image was used to find the clusters using the K-means algorithm with different values of K and it was found that k=3,k=4 give the closest result and cluster the points . the following are the results for different values of k.



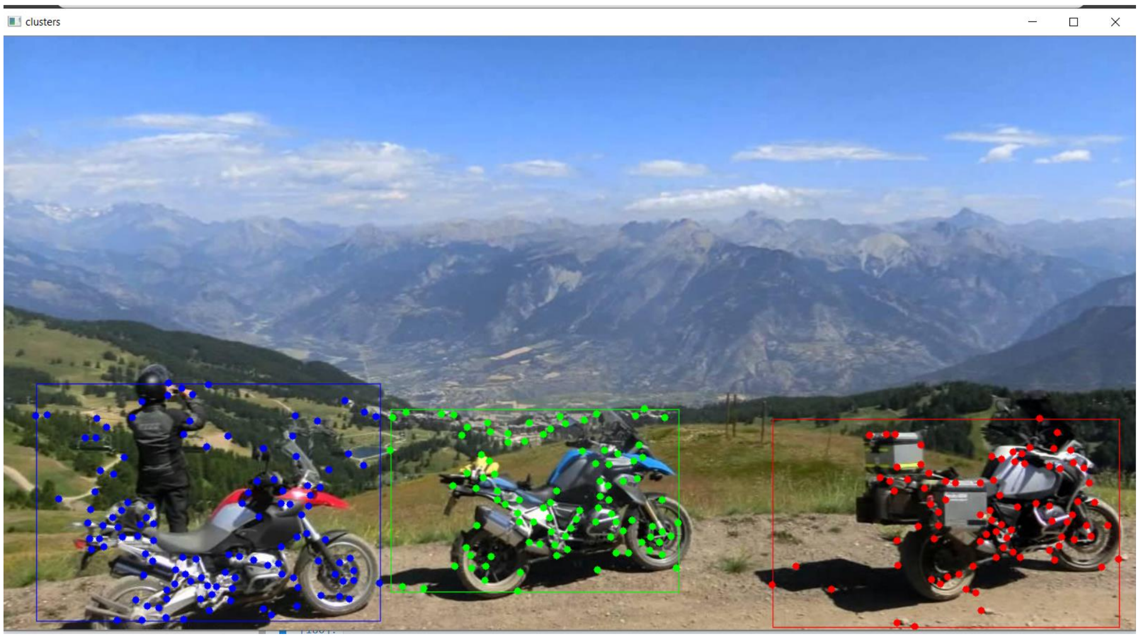
when K=6



when k=5



when k=4



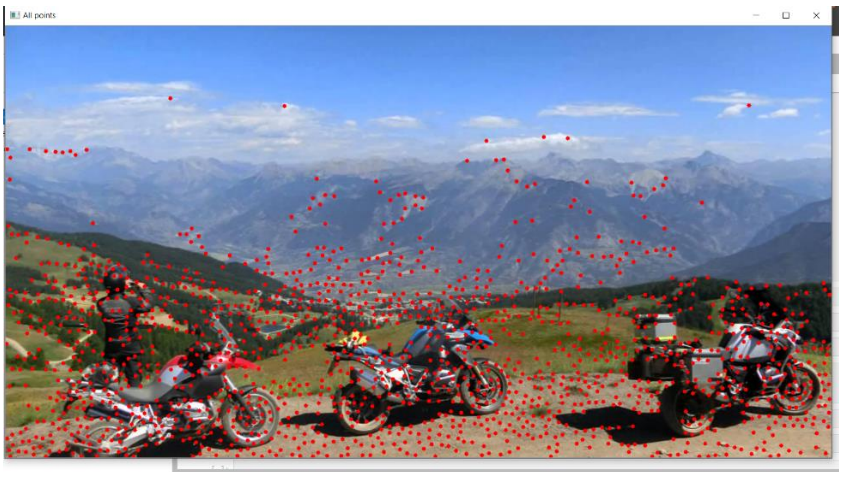
when k=3

As can be seen from the above results we notice that k=3, 4 have the closest to best results while k=4 is the most suitable clustering as the 2 bikes on the right are treated as one cluster and the leftmost bike's front portion as 1 cluster and the man and the back as the other cluster. Whereas in k=3 the both the bike and the man are treated as one cluster and hence k=4 is better since there are 4 objects in the images.

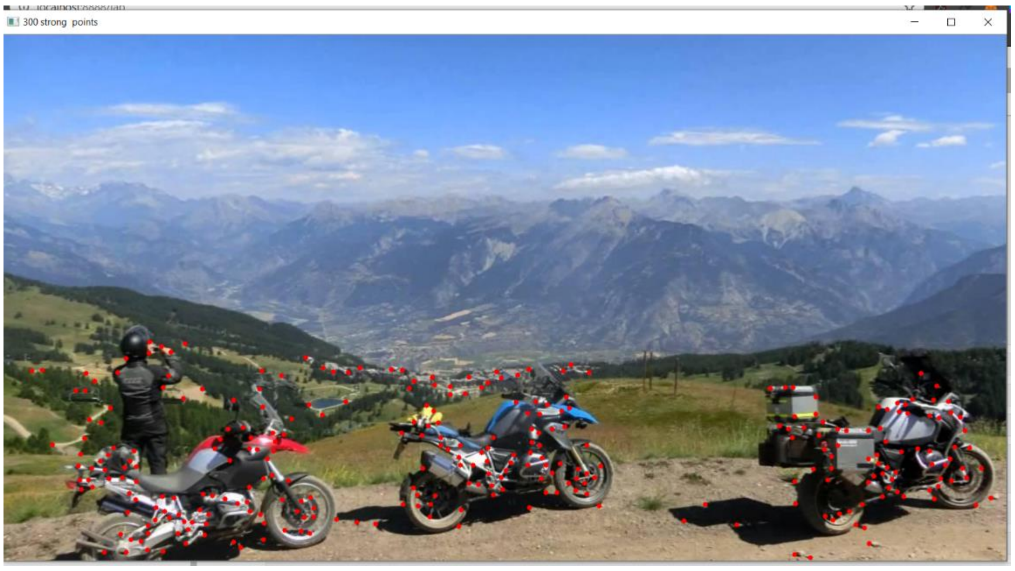
Obtained results:

After comparing the results with different values of k, the points on the image are best clustered when k=4 . The given image shows 3 bikes and 1 man standing . When k=4 as chosen the points of the bike on the right are classified as one cluster , the ones on the middle bike as one cluster, the front half of the left bike as the third cluster and the man and the latter end of the bike as the fourth cluster.

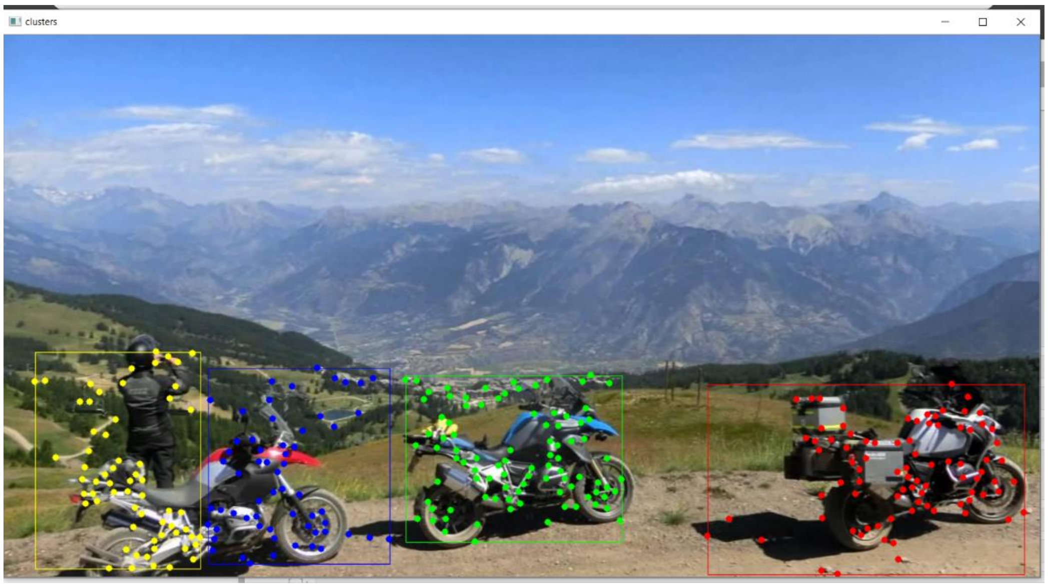
The following image shows all the strong points on the image marked as red dots.



The image below depicts the strongest 300 points on the image.



The picture below shows the 300 strongest points divided into four cluster using K-means algorithm.



References:

https://docs.scipy.org/doc/numpy/reference/

https://pythonprogramming.net/

https://stackoverflow.com/questions/1401712/how-can-the-euclidean-distance-be-calculated-with-numpy

https://docs.opencv.org/2.4/modules/core/doc/drawing_functions.html

https://towardsdatascience.com/k-means-clustering-from-a-to-z-f6242a314e9a

https://www.geeksforgeeks.org/python-opencv-cv2-rectangle-method/

https://www.youtube.com/watch?v=ZS-IM9C3eFg&t=519s