Mean-shift in Image Segmentation

Nicky Acheila i6202817

1 Introduction

Image segmentation is a method used in Computer Vision, where an image is subdivided into pieces (segments) that will provide a new, less complicated, representation of the image to facilitate its later analysis. In this project, we implement the Mean-shift algorithm to create clusters that correspond to image segments. This algorithm iterates through each pixel of an image and finds a corresponding peak out by calculating the mean of the data in a predefined spherical window of radius r. The window shifts to the computed mean and the process is repeated until the shift becomes smaller than a threshold t, when the calculated peak is assigned to the starting point and then the algorithm moves on to the next point.

2 Implementation

In this implementation, we develop the mean-shift algorithm along with two optimization techniques and we evaluate the results. To run the program, **run *AssignmentCV1.m***. This class, calls imSegment.m which takes an RGB image as input and transforms it to a Lab vector (3D or 5D rows). Then calls the meanshift algorithm to segment the image.

2.1. meanshift.m/ meanshift_opt.m

We initialize the peaks_for_elements vector with -1000 in all entries and this is where a peak for each pixel will be assigned. The algorithm goes through all the data points checking whether the current point is already assigned a label in which case it skips to the next iteration. In every iteration the program calls to the findpeak/ findpeak_opt function to compute the peak of the current point and in findpeak_opt the data path traversed until reaching that peak. Then, we determine whether this peak is associated to one of the already collected peaks in the boolean same_peaks array, meaning the ones that are r/2 close to the current peak. For the ones that are not we create a new label. Depending on the optimization we assign the current peak to the currently considered point and/or to the points that are in the basin of attraction and/or to the points of and r/c close to the data path.

2.2 *finpeak.m/ findpeak_opt.m*

In this function, we initialize a shift of 1000 and run the algorithm until the shift is smaller or equal to the threshold t. In each iteration we determine for each data point whether it is located in the current sphere window (in_shpere) and compute the mean of those that are. Then we shift to the mean, this becomes the current_point, and go on in the same way.

3 Experiments & Discussion

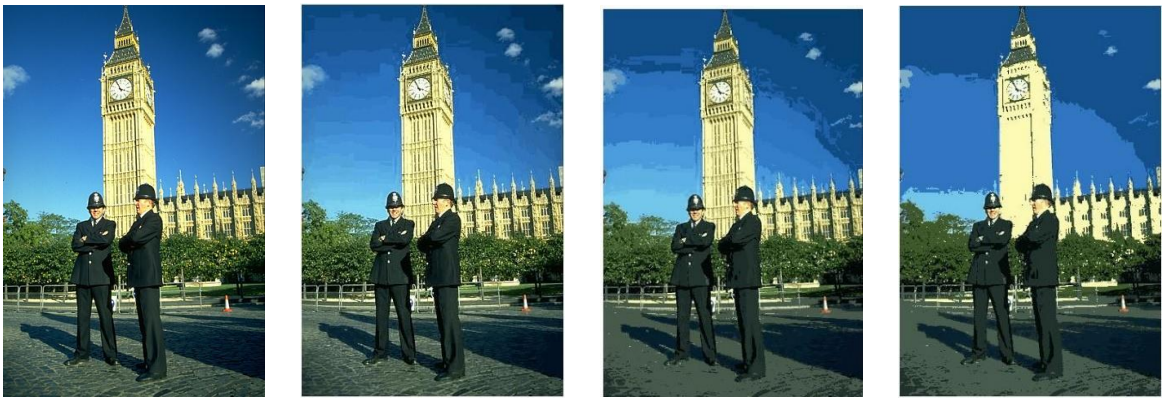
In the first experiment we want to observe the importance of r by running the algorithm with both speed-ups and the same findpeak threshold t=0.01 to all pictures with different radius r.

In Figure 1 we observe that with a small r (r=2) there are very small differences with the original image, for example the girl's pants or the waters in the background. This comes in agreement with the data in Table 1 where we see a high number of clusters in that image. As r gets larger, it is obvious that a lot of the small details are missing, making it maybe a better choice when we want a rough segmentation of an image.

In Figure 2, we see that most of the details remain even in the final frame, such as the clock and details of the building, while in Figure 3 with r=6 the background fence and land have blended into one since the colors were close to begin with.

Original r=2 r=4 r=6

Figure 1



Original r=2 r=4 r=6

Figure 2

Original r=2 r=4 r=6

Figure 3

The following table (Table 1) presents information about the above figures. One interesting thing that one can notice is that the elapsed time in all 3 cases is larger for r=4, when one would expect that the bigger the r, which results as expected in less clusters and iterations, would also decrease time. This unexpected phenomenon could be do to the pictures when we see similar contrast between the colors in all three images or since the findpeak function with a larger r makes smaller shifts until convergence which would result in more iterations on this algorithm. The data correlate with what we see on the images with respect to the number of

clusters, where Figure 2 that has the most details has the highest number of segments in contrast with Figure 3 which has the least amount of details. The radius r has a great impact on the number of clusters as we observe a 98.48% drop in Figure 1, 97.55% in Figure 2 and 99.09% in Figure 3 from r=2 to r=6, so its value should always be assigned with regards to the problem we need to solve.

r	Elapsed time (sec)	Number of clusters	Number of iterations
Figure 1			
2	1586.93	2891	10150
4	1694.92	319	7221
6	839.82	44	4560
Figure 2			
2	1218.36	3927	13404
4	2108.28	413	10457
6	1787.68	96	5957
Figure 3			
2	1165.14	1432	9486
4	1298.05	85	6327
6	723.57	13	3511

Table 1

In the following experiment we temper with the findpeak threshold t, again with both speed-ups and r=6. In Table 2 it is shown that even though the increase of the threshold significantly decreases the time and the iterations, the opposite trend goes on for the clusters, where we see a small increase from one frame to the other, with the final being close to double of the first one. A possible explanation for this is considering that in the findpeak function the algorithm goes through a smaller path to find the peak, thus we have a larger amount of smaller clusters.

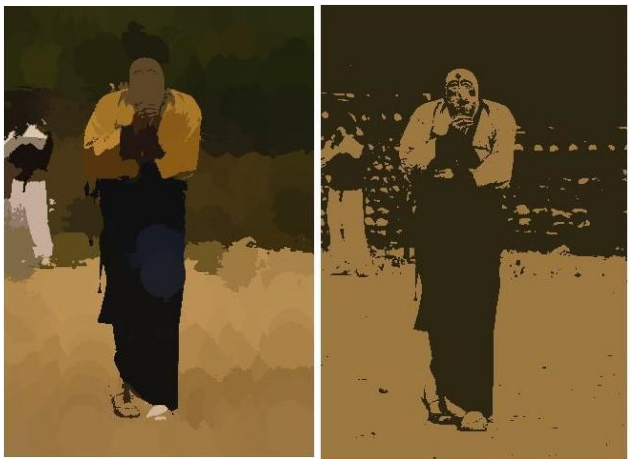
t=0.05 t=0.1 t=0.2 t=0.3

Figure 4

t	Elapsed time (sec)	Number of clusters	Number of iterations
0.05	821.44	48	4406
0.1	653.87	52	4282
0.2	527.34	76	4036
0.3	406.48	93	3769

Table 2

Now, we show in Figure 5 a comparison between 3D and 5D feature input, using both speedups, with r=25 and t=0.4. When running with 5 dimensional data (including spatial information), the ellapsed time was 206 sec, with 2450 iterations and computed 161 clusters, whereas in the second case the program conlcuded after 1.5 sec with 23 iterations and found 2 clusters. It is clear than when adding the coordinate information at the calculation, the segmentation stays able to capture a rather good representation of the image, even with larger r and t values, when the 3D considerably fails.



5D 3D

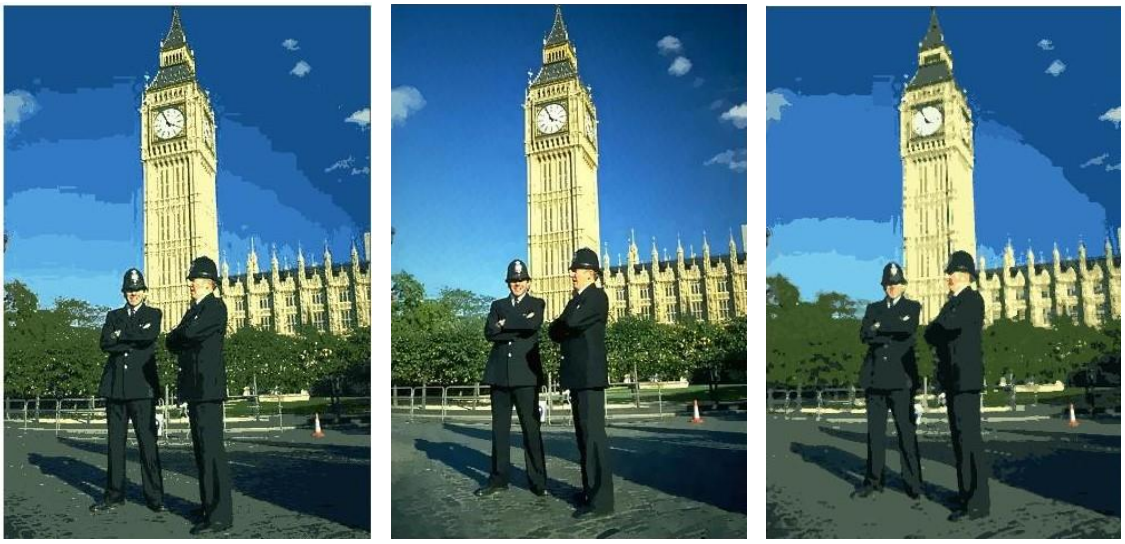
Figure 5

In Figure 6, we observe results with less speed-ups, with r=10, t=0.4, path of r/2 distances. The results indicate that the better strategy is using both optimizations, and also that the data path with a distnace of r/2 does not succeed to capture details of the image such as the face mask, possibly because the data path point are not that close to the peak. The basin of attraction provides accuracy, but is very time costly.

Basin of Attraction [1400 sec, 11 clusters] Data Path [81 sec, 4 clusters] Both[17 sec, 7 clusters]

Figure 6

Lastly in Figure 7 and Table 3, we see that 5D features again provide a very detailed result, while with the use of Median Filter preprocessing we get a picture of significantly less clusters in very short time that is still an accurate representation of the objects in the picture.



3D 5D 3D + Median Filter

Figure 7

	Elapsed time (sec)	Number of Clusters
3D	334.06	109
5D	1711.97	8930
Median filter	261.9	73

Table 3

4 Summary

In this assignment we implemented a means-shift algorithm for image segmentation enhanced with two speed-ups. From the experiments, it seems that it accomplishes a successful segmentation, although at times very time consuming. The data brings out the importance of r dependent on the lever of granularity we want to achieve, while the 5D feature input reinforces the accuracy and ensures better results than the 3D one, but is relatively very time costly. Further enhancements on the algorithm could include preprocessing techniques such as mean filter, histogram equalization and normalization, as well as the refinement of the feature input, although this could potentially slow down the program.