"Cartoonize" an Image With K-Means Clustering and Edge Detection

Nabeel Naiyer, *Student, EE 371R*

***Abstract*— Producing a "cartoonized" version of an image is not necessarily a difficult task, but there are several different methods to produce one. In this paper, we will show how to achieve this task by making use of K-Means Clustering as well as using a Sobel version of the edge detection algorithm [1]. The K-Means Clustering allows us to segment an area of interest from the background. It will classify an image into different groups, and it will let us classify classes of pixels in order to achieve the desired result. Although the clustering can be done with any number, trial and error is required in order to discover what number of clusters will give the best possible image result.**

***Index Terms*— K-Means, cluster, Sobel, cartoon**

I. Introduction

THE goal of taking an image and processing it in a way such that it appears to be a cartoon is so that the image can be used in art mediums such as cartoons or video games. Certain mediums are emphasized, such as video games due to the fact that the matrix of pixel values can be embedded into game software. Converting the art style of an image into a cartoon is important for realism in the context of video games since a hyper-realistic image will look out of place and be jarring compared to the rest of the video game world.

Some methods used to convert images to video games include a pyramid shift segmentation algorithm, a normalized means segmentation algorithm, or extensive bilateral filtering used to smooth an image and preserve boundaries such that an image can be made to look less realistic. These are usually coupled with a Canny edge detection algorithm to find and preserve edges within the image. While these methods are certainly useful, the goal of K-Means Clustering is to achieve the same result with less intensive code and less complexity.

There are three distinct sets of processes that the cartoonizing process undergoes. The first process is clustering the image. The second process is finding the Sobel edge map of the image and then inverting it. The third process is outlining the clustered image using the inverted edge map as a basis.

II. K-Means Clustering

K-Means Clustering is a clustering algorithm which is initialized by randomly guessing a solution. The 'K' input is where the user decides how many subsets or clusters they would like to have. Every time the program runs, a point is assigned to a cluster and then whenever new points are added to the cluster, they become averaged.

The algorithm has a few steps: the initialization, partitioning, and cluster averaging. The initialization is the starting condition, or the number of desired clusters. The partitioning is the part where points are assigned to a cluster. The averaging is where clusters recalculate values based on whether a point is added or removed from that cluster.

Different initializations can give different results, which is the main flaw of this algorithm, since usually only one K value will give the best result. Since an initialization requires a guessed value for K, it doesn't make the overall process an automatic one.

III. Image Samples

The following images are the small test images used for testing the implementation of the cartoonizing program:

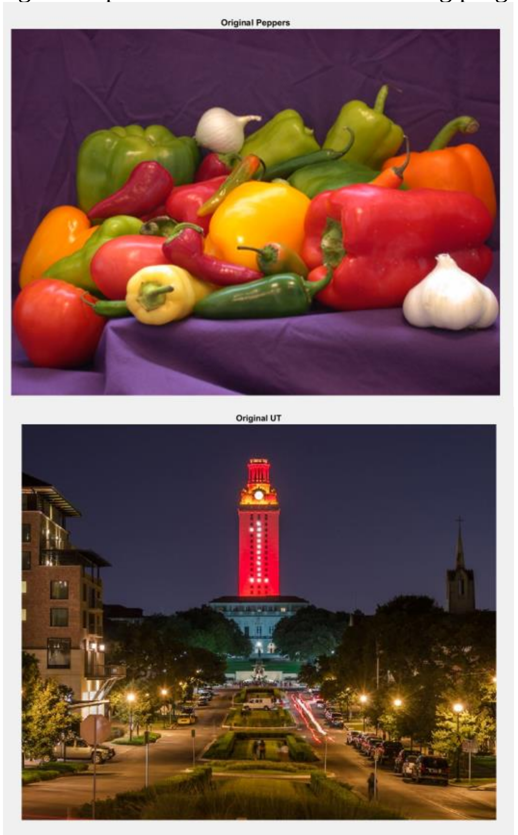


Fig. 1. MATLAB standard image of 'peppers.png' (top) and an image of the U.T. Tower at night (bottom).

The next two images are larger test images used for testing the implementation of the cartoonizing program:

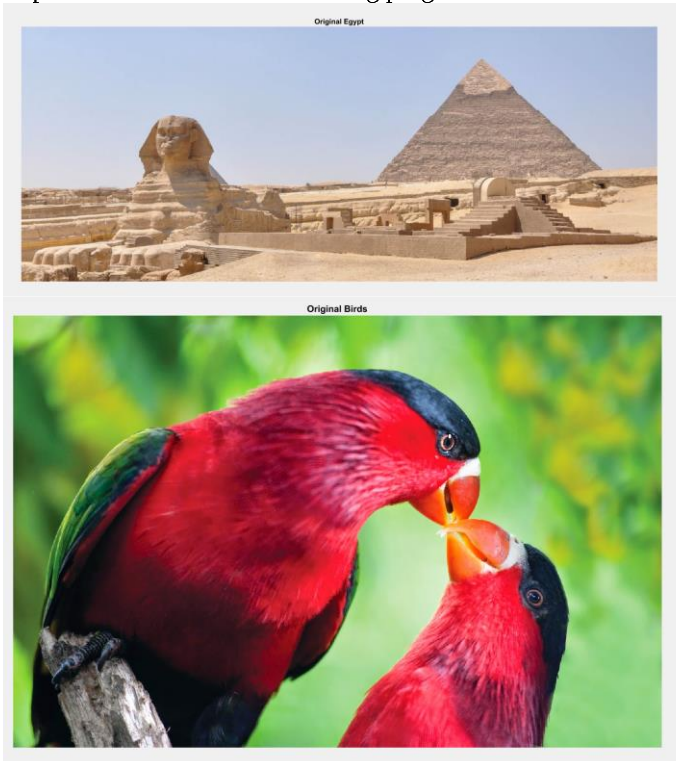


Fig. 2. Image of the sphinx and a pyramid from Egypt (top) and an image of two red birds (bottom)

IV. K-MEANS CLUSTERED IMAGE

After running the K-Means Clustering on the above images, clustered images were produced. The clustered images are the starting point for the cartoonizing process. It was discovered through testing the algorithm that large image sizes took noticeably longer to produce an output compared to smaller image sizes. Although this is to be expected, the amount of time was disproportionate, with smaller images taking anywhere from ten to thirty seconds, and larger images taking anywhere from three to four minutes. This means that some optimization of the K-Means algorithm is necessary.

The following images are the results of the clustered versions of the small images:

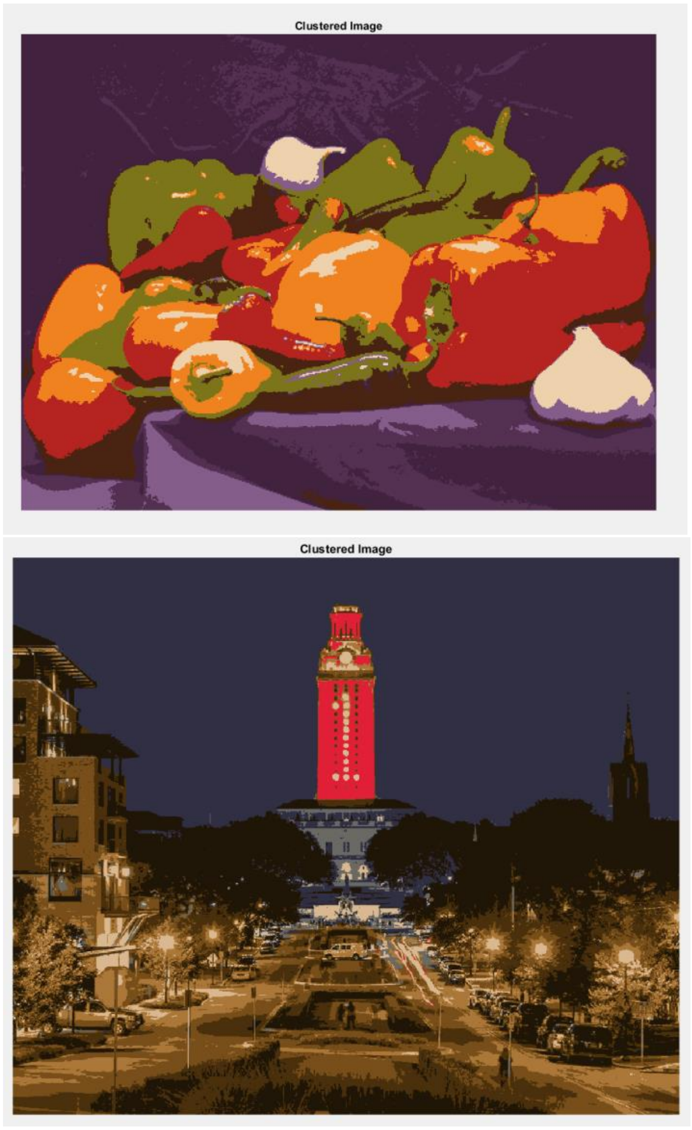
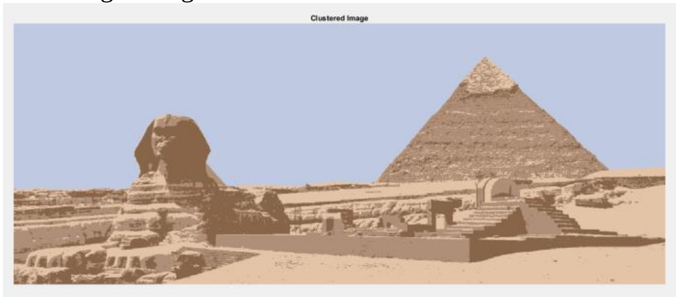


Fig. 3. Clustered version of peppers from Fig. 1. (top) and clustered version of the U.T. Tower from Fig. 1. (bottom).

The following images are the results of the clustered versions of the larger images:



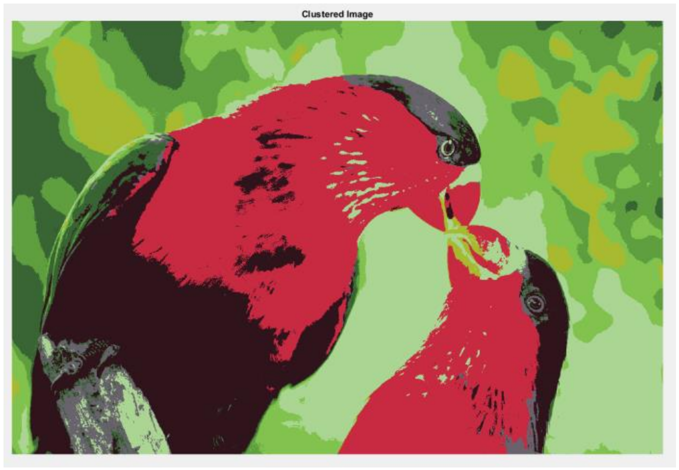


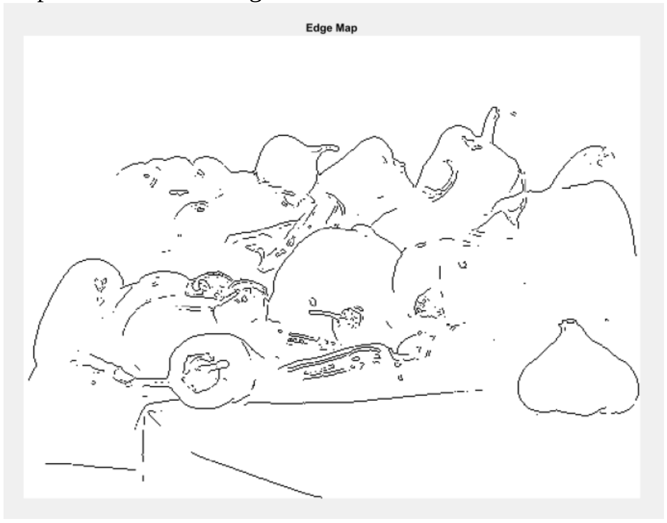
Fig. 4. Clustered version of the sphinx from Fig. 2. (right) and clustered version of the two red birds from Fig. 2. (bottom).

V. Sobel edge map

The Sobel edge detection algorithm makes use of two 3x3 kernels which are convolved with the original image. These are used to approximate derivatives for both horizontal and vertical components.

There exists a built in Sobel edge detection function in MATLAB, so there was no need to write an original one. After computing the Sobel edge map, the next step was to take the complement of the image. This was necessary for the purpose of outlining the images in question; since the outline needed to be in black, it was a simple matter of writing a loop that would iterate over the inverted edge map and locate all the black pixels in that image. Following that black pixel detection, it was a simple to write another loop that would take the locations of the black pixels, and map the pixel location in the original image to black as well.

The following images are the results of the inverted edge maps for the small images:



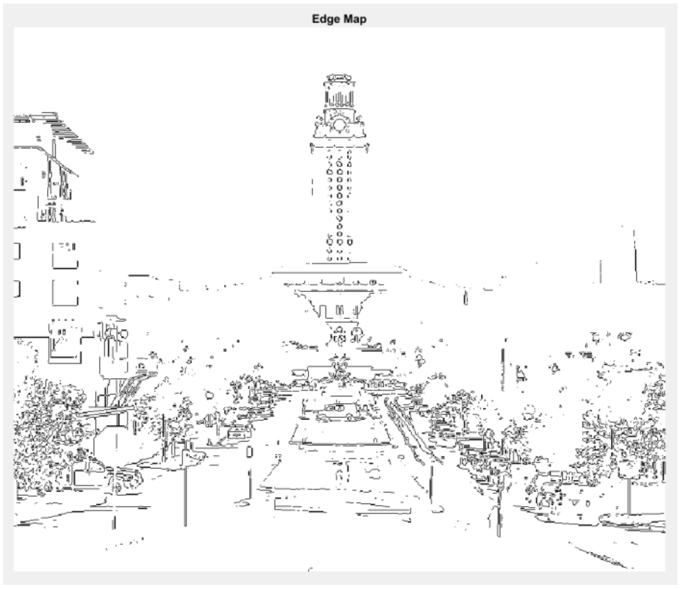


Fig 5. Edge map for the peppers from Fig. 1. (left column, bottom of page) and the edge map for the U.T. Tower from Fig. 1 (above).

The following images are the results of the inverted edge maps for the larger images:

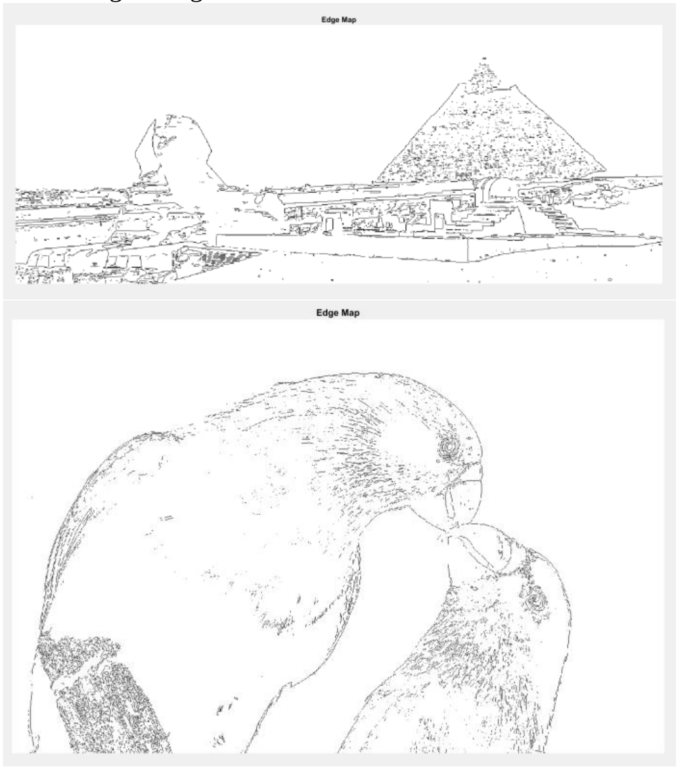


Fig. 6. Edge map for the sphinx and pyramid from Fig. 2. (top) and the edge map for birds from Fig. 2. (bottom).

VI. Outlining

The outlining process is a simple one. It is performed by first running a loop to find all of the black pixel locations in the inverted Sobel edge map. Then, the next step is to store those pixel locations, and run another loop which will locate those pixel locations in the clustered image. Finally, another loop will go to the pixel locations in the clustered image and convert it to 0, which is being used as the value for black.

This process will essentially outline the image in a black line, which adds to the cartoonish realism that we are trying to achieve.

The following images are the results of the smaller images being outlined:

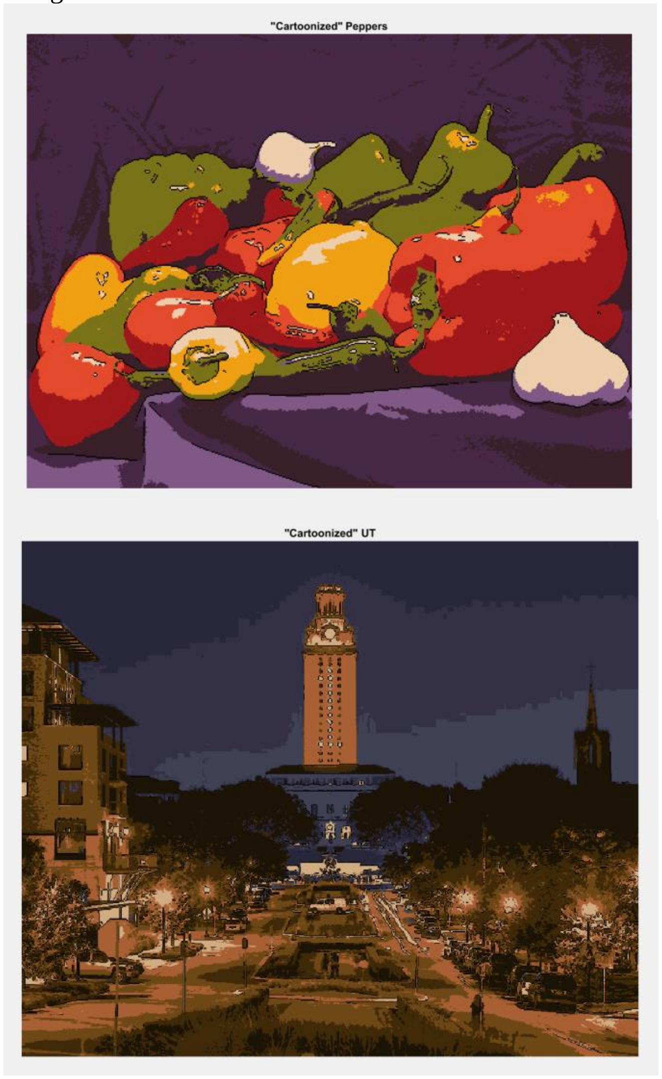


Fig. 7. Outlined version of peppers from Fig. 3. (top) and outlined version of U.T. Tower from Fig. 3. (bottom).

The following images are the results of the larger images being outlined:

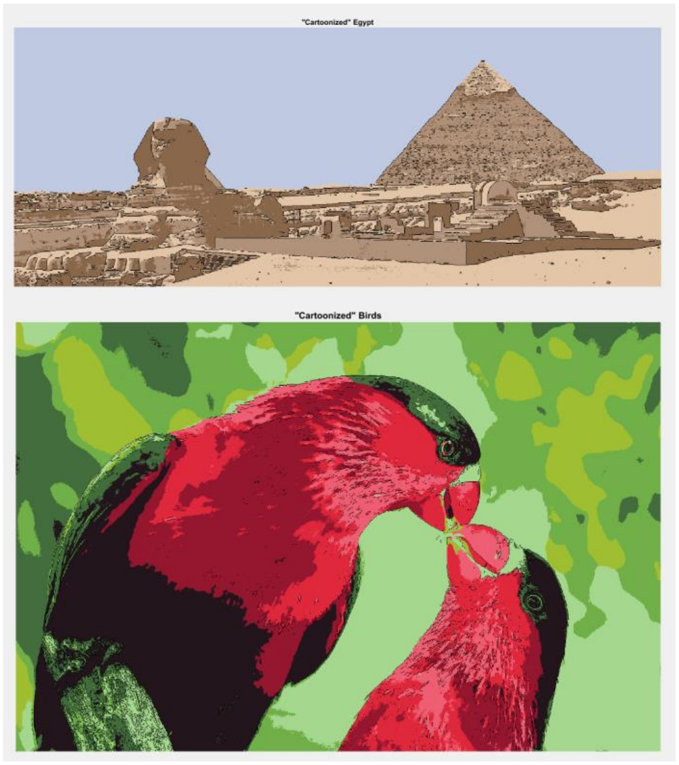


Fig. 8. Outlined version of sphinx and pyramid from Fig. 4. (top) and outlined version of birds from Fig. 4. (bottom).

VII. Results

As shown in Figures 7 and 8, the resulting images from the cartoonizing program are more vibrant and have an outline which make them seem less realistic and more animated.

Comparing Figures 7 and 8 to Figures 1 and 2 shows how this program may be useful in an artistic manner.

VIII. Conclusion

As a single person working on this project, the author of this paper wrote all of the code and decided on the techniques to be used, the images to be used, and the format of this paper.

This entire process showed how difficult it can be to get an image algorithm to respond to the methods one wishes to apply to it, as well as how to manipulate an image over pixels instead of colors or shapes.

References

[1] MATLAB. (1994-207). Detecting a Cell Using Image Segmentation. Mathworks. United States of America. [Online]. Available: https://www.mathworks.com/help/images/examples/detecting-a-cell-using-image-segmentation.html

[2] K. Muthukannan, M. Merlin Moses. 29 Dec. 2010. Detecting a Cell Using Image Segmentation. Kongu Engineering College. India [Online]. Available: http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=5738735&tag=1