Alternate Image-Segmentation Methods

# Active Contour Analysis

## Theory

Active contour analysis, also know as snakes, is a method to segment ROI (regions of interest) from images. It takes a 2-D grayscale image and identifies the foreground and background given a mask. The theory behind active contour is similar to what we learned about in class. There exists a parametric curve that represents the boundary of an object, and an energy function matching the curve. In order to find the boundary of the object, we try to minimize the energy function. The idea behind active contour models is that a user will select an initial curve close to the object to give the snake a starting point. You can run the algorithm without specifying an initial boundary, however if you run it with the default mask the results are very poor. The energy function is a sum of both an internal and external energy along with constraint energy:

The internal energy is the sum of the elastic energy and bending energy, as defined below.

Elastic energy is responsible for shrinking the contour because it discourages stretching by introducing tension. One can think of the elastic energy as if the curve was an elastic rubber band that wanted to shrink.

,

In the above equation, α(s) is a weight used to control elastic energy at different parts of the contour. For most applications this is a constant.

Bending energy is used control the curvature of the contour. It is defined as follows:

In the above equation, β(s) is similar to α(s) in the elastic energy equation, as it is a weight used to control the effect of the bending energy. Bending energy tries to smooth out the curve, thus is minimized for a circle. Now that we have defined both bending and elastic energy, we can express total internal energy as one integral.



The external energy function is derived from the image, called Eimage. There are various functions to choose for this, the important feature is that the function takes on smaller values at the features of interest, in our case boundaries. This is because we are trying to minimize the function, so it should be smallest at the boundaries. One such example of a function is



## MATLAB Implementation

The active contour method was the simplest of all three methods because it was a built in function of MATLAB (added in version 2013a). Thus, the MATLAB implementation was simply calling the function with various parameters. At a most basic level, the activecontour function needs two parameters: 2D gray scale image, and a starting mask. You can also specify number of iterations as well as Chan-Vese vs Edge. For the coins image, I used a very simply mask of all 1’s with a 25 pixel border on all four edges. For the rest of the images, I used the roipoly function so that the user can draw a polygon around the object of interest, which is then turned into a mask and passed to the activecontour function. The mask must include the region of interest, and it is preferable to be relatively close to the object. The Chan-Vese or edge parameter specifies which segmentation method the algorithm will use. Chan-Vese has the advantage that in can adaptively grow bigger on an object if part of it is outside the region of interest, and will make the snake go concave if needed. The edge method unfortunately doesn’t do this.

## Observations

Overall, active contour analysis was incredibly good at identifying boundaries of regions of interest. However, since it only works on gray scale images you must convert color images to gray scale before running the algorithm. One potential solution to this is to convert the image to gray scale, run the algorithm, then apply the binary mask that was outputted by active contour to the image again which would return only the region of interest, in color. Also, I found that using the Edge mode, instead of Chan-Vese resulted in much worse results. There was not a single image that edge worked better than Chan-Vese, and even for the most basic example, coins.png, the results were about the same. In addition, if the number of iterations was not high enough, it resulted in the regions being too large, and not closing in on the object enough. Fortunately, I never found that there was such a thing a too many iterations, although even 300 took about 5 seconds so if you simply said run all images at 500 iterations, you would be wasting your time on most images. Some of the pros of active contour analysis are:

* Works extremely well for identifying subject in portraits
* Works well on distinct shapes
* Is relative fast

Some of the cons however are:

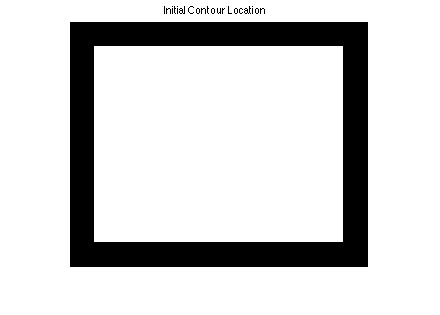
* Must convert color images to gray scale
* Need to specify number of iterations for effective results

**Results:**

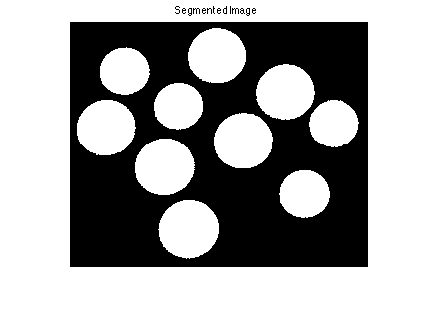
Original (coins.png):



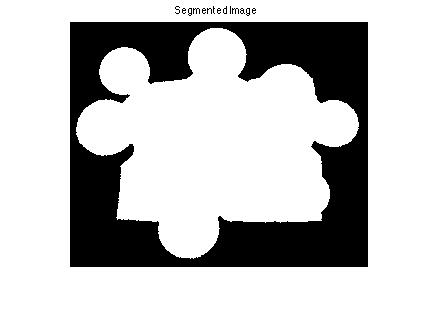
Starting mask:



activecontour(I,mask,300):



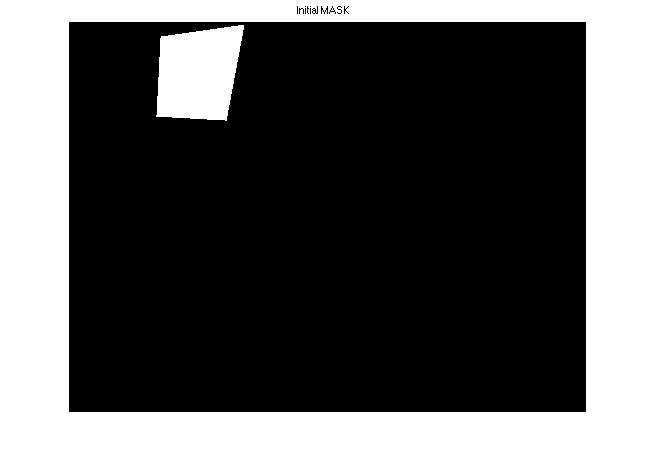
bw = activecontour(I,mask,100);



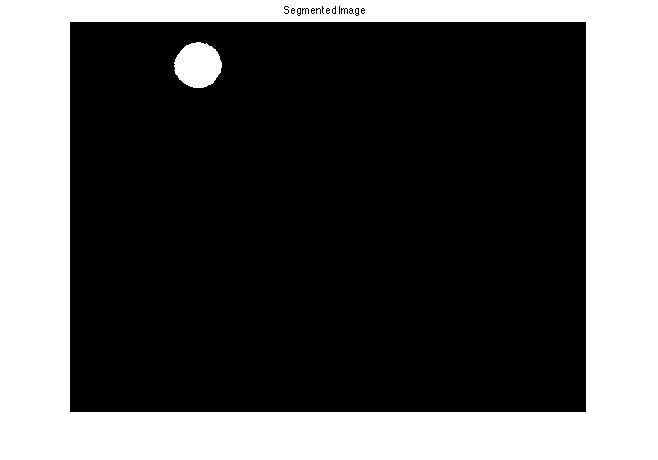
Original (coloredChips.png):



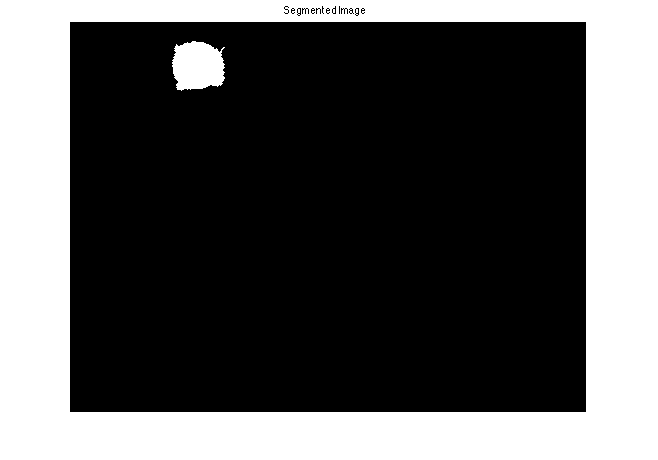
Mask:



activecontour(G, mask, 200, 'Chan-Vese'):



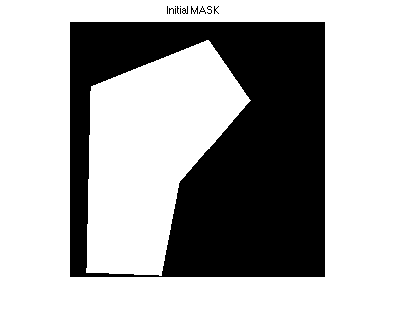
activecontour(G, mask, 20, 'Chan-Vese'):

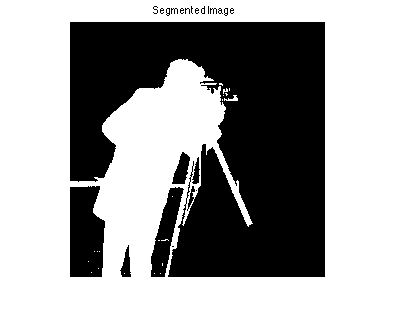


Original (cameraman.tif):

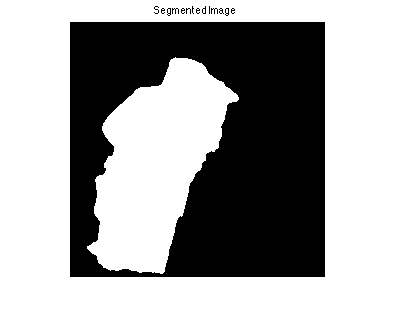


Mask:



activecontour(I, mask, 250, 'Chan-Vese'):  


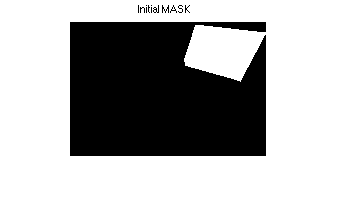
activecontour(I, mask, 250, 'Edge'):



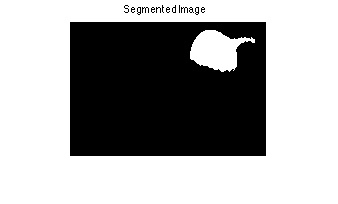
Original (onion.png):



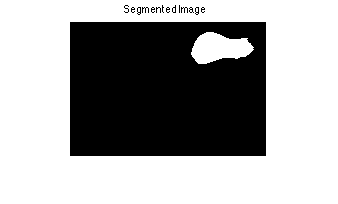
Mask:



activecontour(I, mask, 250, 'Chan-Vese'):



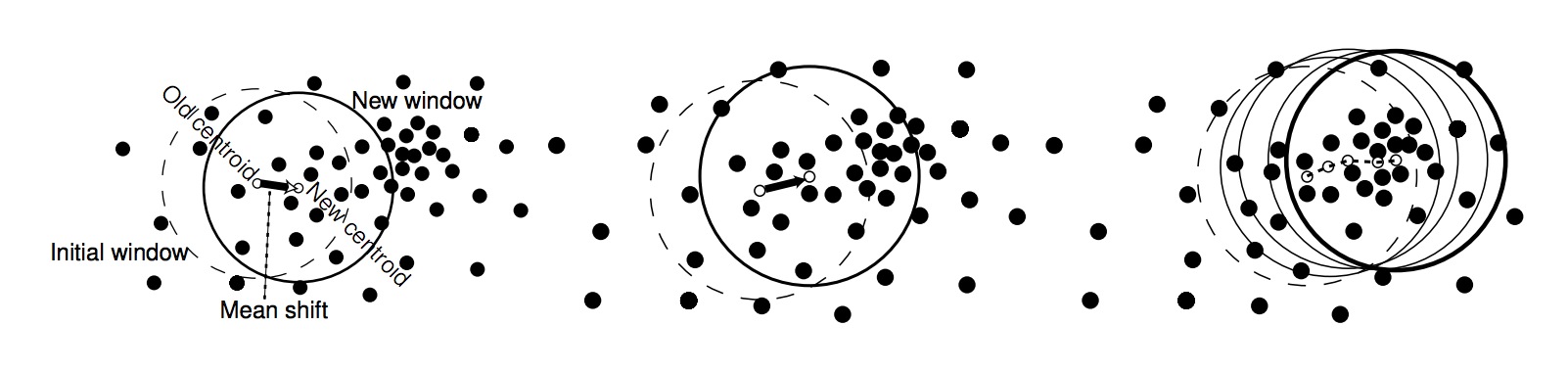
activecontour(I, mask, 250, 'Edge)**:**

****

**Mean Shift**

**Theory**

At a high level, mean shift is a very elegant and simple algorithm. The basic idea is to find the densest region. The way we go about doing that is to compute the center of mass for a given region of interest. Assuming the center of mass is not equal to the center of the region, shift the center of the region (usually a circle). Now, compute the center of mass again and shift the region to the center of mass. Continue to do this until the center of mass is equal to the center of the region, i.e. there is no shift. The following graphic is a good demonstration of how mean shift works



The window shifts to the right in order to be centered on the densest population of pixels.

The math is relatively straightforward, however it really depends on which kernel you pick. For some kernel function K(xi-x) given, the mean shift, denoted m(x) as follows

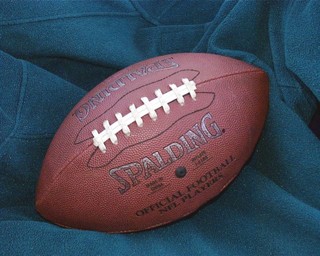
Some examples of kernel functions are:

## MATLAB Implementation

After reading about MATLAB and mean shift, it seemed that everyone recommended using the EDISON C++ system from Rutgers University. The RIUL Lab at Rutgers developed EDISON, which stands for Edge Detection and Image SegmentatiON system, in 2003. Even today, it seems that this is the best implementation of mean shift, outside of open cv. The reason why I chose to use this over a native MATLAB implementation is speed. Most people on the internet said that MATLAB implementations of mean shift often took many hours just to process a single image, and EDISON could do it in just a handful of seconds. In order to use the EDISON code in MATLAB, you must first run the compile\_edison\_wrapper script, which will build the Edison library for your platform. Then, simply executing the mean\_shift\_demo script will run all of the demos. The hs and hr are the spatial and range bandwidth parameters needed by mean shift, and M is the minimum region size.

**Results:**

Original (football.jpg):



mean\_shift(I, 20, 20, 20):

**

mean\_shift(I, 5, 5, 10):



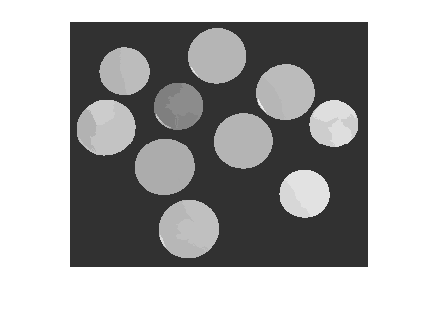
mean\_shift(I, 50, 50, 50):



Original (coins.png):



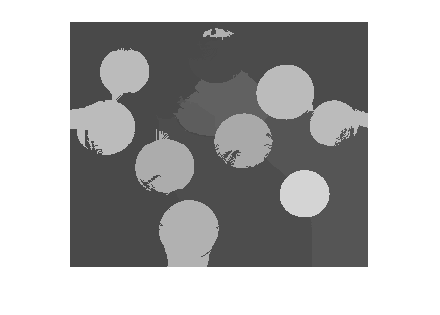
mean\_shift(I, 11, 11, 11):



mean\_shift(I, 4, 4, 4):



mean\_shift(I, 40, 40, 25):



Original (peppers.png):



mean\_shift(I, 25, 25, 25):



mean\_shift(I, 8, 8, 8):



mean\_shift(I, 50, 50, 50):



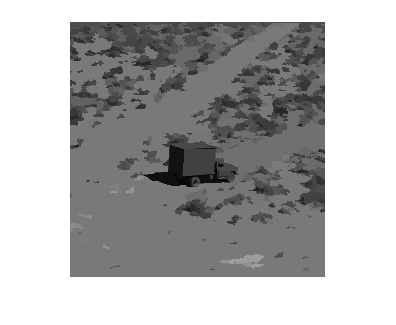
Original (truck.gif):



mean\_shift(I, 11, 11, 10):



mean\_shift(I, 4, 4, 5):



mean\_shift(I, 23, 23, 25):



## Observations

The mean shift algorithm worked really well on both color and gray scale images. However, as a segmentation algorithm it doesn’t work quite as well as active contour analysis, as the output image is simply regions of the same pixel value compared to that of active contour, which identifies boundaries of regions. I also found that the hr and hs parameters were extremely important in producing a good output image. As you can see in the results, the second image in each column is what I thought was a “good” output then the next two were bad. On the peppers image for instance, I had to use a much larger hr and hs value (more than double) in comparison to the truck image. This is because the feature is so much smaller in the truck image. I found that this algorithm worked much better for images with less going on, like the football image for instance. When there were many regions of interest, like in the peppers image, the algorithm sometimes merged the boundaries of two regions. Some of the pros of this method include

* Works well for colorful images
* Does not assume spherical clusters
* Just a single parameter needed (window size)
* Finds variable number of nodes
* Robust to outliers

On the other hand there are some problems with it such as:

* Spatial and range bandwidth parameters or else the regions tend to merge
* It averages out the color of an object, unlike active contour, which forms a boundary.

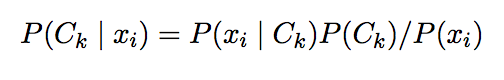
# Expectation Maximization

## Theory

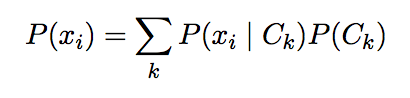
Fundamentally, the Expectation Maximization algorithm segments an image by assigning each pixel to the segment (referred to as cluster) for which it most likely belongs. Each pixel consists of a property vector of length *n* that contains the data (e.g., red level, green level, blue level, etc.) that “builds” the aspects that characterize each cluster: a mixture of *n* Gaussian (normal) distributions. The EM algorithm consists of three major steps: parameter initialization, expectation, and maximization.  
  
**Parameter Initialization**  
  
Each cluster *Ck* consists of three parameters, a mean vector *uk* of length *n*, a standard deviation vector ***σ****k* of length *n*, and a weight *wk* = *P(Ck).* The mean vector values are initialized to random values within the range of what is expected in the property vector. Lastly, each weight *wk* is initialized to *1/K*, giving each cluster an equal weight.

**Expectation**

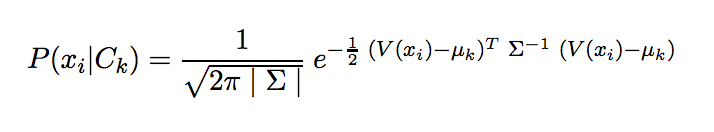
During this step, the probability *P(Ck | xi)* that each pixel *xi*belongs to each cluster *Ck* is calculated. The probability is calculated with the following equation:



where *P(xi)* is calculated with



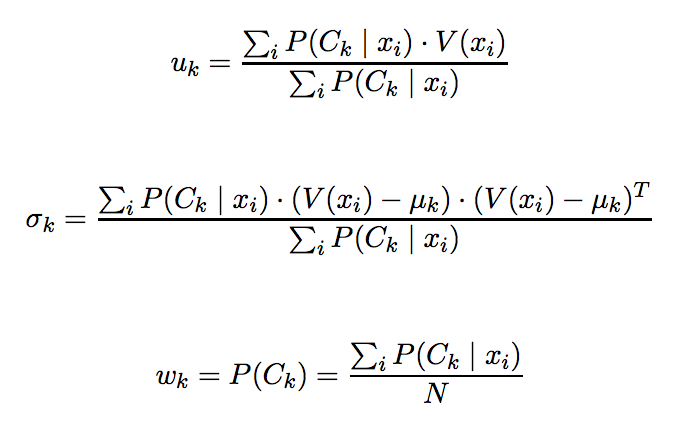
and *P(Ck)* is equal to the weight wk. Since this implementation regards property vector values as random variables belonging to Gaussian distributions, *P(xi | Ck)* is calculated with:



where *V(xi)* is the property vector of pixel *xi*.

**Maximization**

Now that each pixel has been “assigned” to a cluster, we recognize that the random values we initialized the parameters of the Gaussian distributions to are nonsense and are not likely to accurately describe the “true” parameters. In the maximization step, we estimate the values that are most likely to be the parameters of the *n* Gaussian distributions of each cluster *Ck*. Each cluster’s parameters are updated with the following equations:



The Expectation and Maximization steps are repeated until convergence occurs. There are several ways to detect convergence. In our implementation, we calculate the sum of the squared differences of the cluster parameters between the current and previous iterations. Once this sum (or energy) has fallen below the convergence tolerance parameter given by the user, the algorithm has finished.

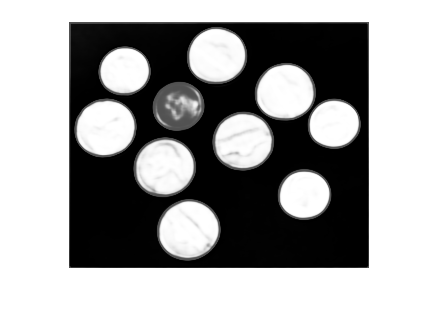
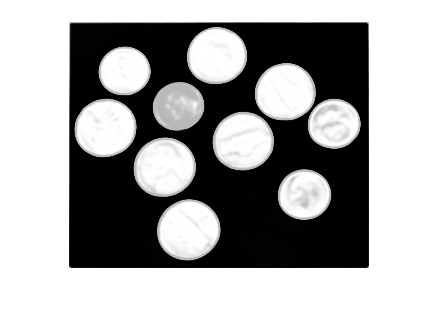
## MATLAB Implementation

Our implementation of the expectation maximization algorithm (“em.m”) segments a greyscale or color image by color space. The implementation can be tuned with several parameters. K controls how many clusters or segments the image is partitioned into. maxIter limits the number of EM steps that can happen before convergence occurs. convTol is the value that the energy must fall below in order for convergence to occur. useG determines whether to apply a Gaussian filter to the image before segmentation. gSize controls the height and width of the Gaussian filter mask. Lastly, gSigma represents the sigma value of the Gaussian filter. Our implementation is inspired by code by Rongwen Lu.

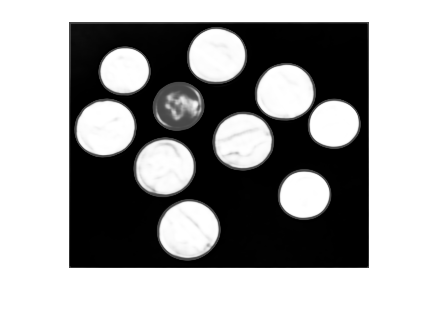
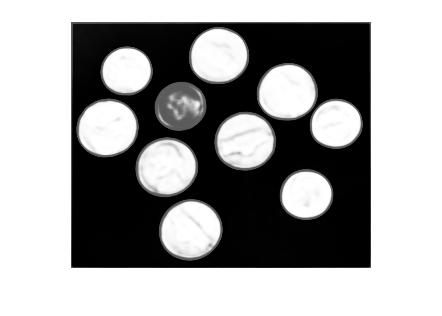
## Results

****

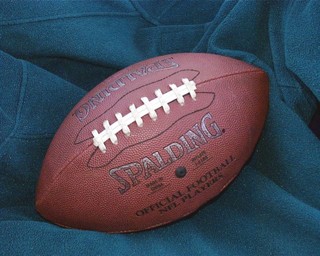
coins.png

****

em(I, 3, 20, 10e-6, 1, 5, 4) em(I, 3, 75, 10e-6, 1, 5, 4)



em(I, 3, 150, 10e-6, 1, 5, 4) em(I, 3, 300, 10e-6, 1, 5, 4)

football.jpg em(I, 10, 1000, 10e-3, 1, 5, 4)



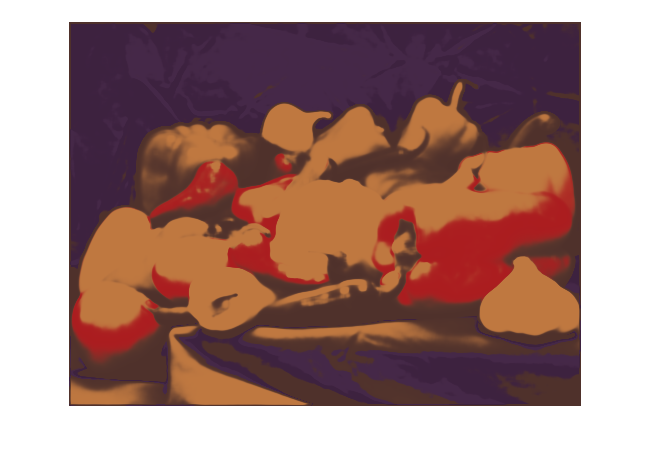
em(I, 10, 1000, 10e-4, 1, 5, 4) em(I, 10, 1000, 10e-5, 1, 5, 4)



em(I, 10, 1000, 10e-6, 1, 5, 4) em(I, 10, 1000, 10e-7, 1, 5, 4)



peppers.png em(I, 2, 1000, 10e-6, 1, 5, 4)



em(I, 5, 1000, 10e-6, 1, 5, 4) em(I, 10, 1000, 10e-6, 1, 5, 4)



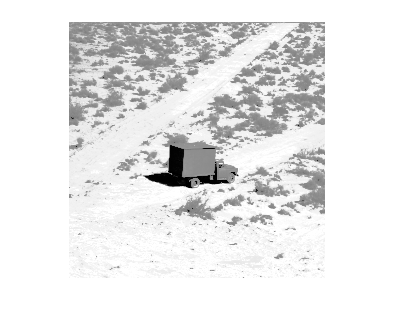
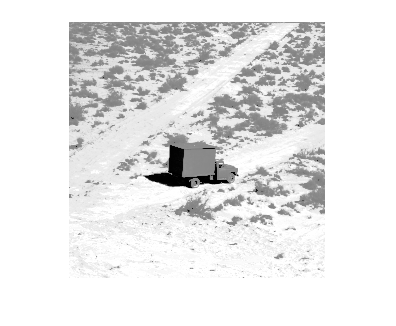
em(I, 15, 1000, 10e-6, 1, 5, 4) em(I, 20, 1000, 10e-6, 1, 5, 4)

## Macintosh HD:Users:goose:Dropbox:PSU:Junior:454:CV_honors:Images:truck.gif

truck.gif

## Macintosh HD:Users:goose:Dropbox:PSU:Junior:454:CV_honors:expec_max_out_images:truck2 10e-6.pngMacintosh HD:Users:goose:Dropbox:PSU:Junior:454:CV_honors:expec_max_out_images:truck5 10e-4.png

em(I, 2, 1000, 10e-6, 0, 5, 4) em(I, 5, 1000, 10e-4, 0, 5, 4)



em(I, 10, 1000, 10e-4, 0, 5, 4) em(I, 15, 1000, 10e-3, 0, 5, 4)

## Observations

Due to the standard practice of initializing the means of the Gaussian distributions to random values found in the input property vectors, the expectation maximization algorithm is not exactly consistent. Running the algorithm multiple times with the same parameters will generally produce the same output with the exception of a few “outlier” outputs that could be drastically different.

Altering the maximum number of iterations (maxIter) is demonstrated with “coins.png.” All parameters were kept constant except maxIter. In the coin examples above, none of the images converged before the maximum number of iterations was hit. It is easy to see that the EM algorithm is quite good at partitioning the image into acceptable regions in a small number of iterations. Even with 15 times the number of EM iterations, the output result with 300 iterations remains quite comparable to the result with only 20.

Altering the convergence tolerance (convTol) is demonstrated with “football.jpg.” All parameters were kept constant except convTol. Note how larger convergence tolerances lead to coarser segments, and smaller tolerances result in finer segments.

Altering the number of segments (K) is demonstrated with “peppers.png” and “truck.gif.” A 5x5 Gaussian filter with sigma=4 was applied to “peppers.png” and not applied to “truck.gif.” Since this implementation segments on color space alone, fine color noise can be an issue. Applying a Gaussian filter before segmentation can alleviate this issue. We also observed that greyscale images sometimes struggle to converge with the convergence tolerance set to 10e-6. Increasing this parameter by several orders of magnitude may be necessary to achieve convergence depending on the value of K.

# Works Cited

Richard Szeliski. Computer Vision: Algorithms and Applications. Springer, New York, 2010.

M. Kass, A. Witkin, and D. Terzopoulos, "Snakes: Active contour models.“, International Journal of Computer Vision. v. 1, n. 4, pp. 321-331, 1987.

Chenyang Xu and Jerry L. Prince , "Snakes, Shape, and Gradient Vector Flow“, IEEE Transactions on Image Processing, 1998.

C. Xu and J.L. Prince, “Gradient Vector Flow: A New External Force for Snakes”, Proc. IEEE Conf. on Comp. Vis. Patt. Recog. (CVPR), Los Alamitos: Comp. Soc. Press, pp. 66-71, June 1997

Comaniciu, D.; Meer, P., "Mean shift: a robust approach toward feature space analysis," *Pattern Analysis and Machine Intelligence, IEEE Transactions on* , vol.24, no.5, pp.603,619, May 2002

Carson, Chad, Serge Belongie, Hayit Greenspan, and Jitendra Malik. "Blobworld: Image Segmentation Using Expectation-Maximization and Its Application to Image Querying." *IEEE Transactions on Pattern Analysis and Machine Intelligence*(2002): Web. <http://www.cs.berkeley.edu/~malik/papers/CBGM-blobworld.pdf>.

Ng, Andrew. "Mixtures of Gaussians and the EM Algorithm." Web. <http://cs229.stanford.edu/notes/cs229-notes7b.pdf>.

Tomasi, Carlo. "Estimating Gaussian Mixture Densities with EM – A Tutorial." Web. <http://www.cs.duke.edu/courses/cps196.1/spring04/handouts/EM/tomasiEM.pdf>.

Rennie, Jason. "A Short Tutorial on Using Expectation-Maximization with Mixture Models." Web. <http://www.cs.duke.edu/courses/cps196.1/spring04/handouts/EM/rennieEM.pdf>.

Liu, Dingding, Bilge Soran, Gregg Petrie, and Linda Shapiro. "A Review of Computer Vision Segmentation Algorithms." Web. <http://courses.cs.washington.edu/courses/cse576/12sp/notes/remote.pdf>.

Lu, Rongwen. "Image Segmentation with EM Algorithm." N.p., n.d. Web. <http://www.mathworks.com/matlabcentral/fileexchange/34164-image-segmentation-with-em-algorithm>.