ECE 6560

Analysis of Chan-Vese Image Segmentation

Rahmaan Lodhia

April 25, 2015

Contents

1. Introduction

Image segmentation is a thoroughly explored topic in Image Processing and Computer Vision. The goal of image segmentation is to partition an image into meaningful sections, which is used in a multitude of fields from medical applications such as locating tumors in medical images to recognition applications such as determining fingerprints. Some of the basic implementation of segmentation include thresholding, clustering and edge detection, but there are many robust methods available, some of which involves the concept of active contours.

Active contours uses the idea of energy minimizations to define the movement of a curve to enclose significant objects in an image. The first implementation of these methods, known as the Snakes model proposed Kass et al. [1], was successful in segmentation, but was found to be heavily influenced by the nature of the edges found in an image. Later on, a new set of active contours were developed that were based on region and as result independent of edges in an image. These contours are influenced by the information found in the area enclosed by the active contour and the area outside the active contour allowing it to be unaffected by edges. These region-based contours are the subject of this paper, and the one that will be discussed and analyzed is the Chan-Vese model proposed by Chan and Vese [2].

In this paper, the definition and derivation of the mathematical model used in this algorithm will be shown in Section 2. Section 3 will demonstrate how to discretize and implement this model on a computing application. Section 4 will present and explore several quantitative and qualitative results from the implementation of this method. Finally, Section 5 will provide closing remarks discussing weaknesses found in the implementation and any potential work that could improve upon the model used in this paper.

2. The Chan-Vese Model

2.1. Defining the Energy Functional

Active contour models require the definition of an energy functional. The purpose of an energy functional is to create a model for the energy of the image that the active contour will attempt to minimize. At each time step, the goal of the active contour is to progress to a lower energy. The Chan-Vese model is a region-based active contour, so as a result, the energy functional should attempt to minimize the energy in the regions inside and outside the contour.

The energy functional used in the Chan-Vese model can be defined as follows:

$$E(C,u,v) = \lambda_1 \iint_{\mathrm{R}} (I-u)^2 dxdy + \lambda_2 \iint_{\mathrm{R}^C} (I-v)^2 dxdy + \mu \int_C ds + \nu \iint_{\mathrm{R}} dxdy \qquad (1)$$

In this equation (1), I represents the image, u is the average intensity inside the image, v is the average intensity outside the image, C is the contour, R is the domain inside of the contour, R^C is the domain

outside of the contour, and λ_1, λ_2, μ, and ν are scaling parameters. Each term in this equation is a part of the overall defined energy in the image and are the values we wish to minimize. The first two terms pertain to the intensity found in the two regions the contour seeks to create. Whatever is inside the contour should be an object of interest and whatever is outside of the contour should be the rest of the image. The first two terms seek to minimize the energy on both sides of the contour and keep the regions as uniform as possible. The third term seeks to penalize the length of the contour, which will lead the contour to attempt to enclose with as little length as needed. The final term seeks to penalize the area of the contour, which will attempt to force it only enclose the smallest area possible. The parameters in front of each term vary the influence each energy has on the overall energy.

In its current state, this energy functional (1) does not explicitly show us how the contour C changes at the end of each time step. We need to derive what is known as the gradient descent C_t of the function. The gradient descent is a function that shows how the contour will evolve during each time step of the algorithm. After its derivations, we can redefine the gradient descent in terms of a level-set function, which helps to simplify its numerical computation. The next section details the derivation of this contour evolution model.

2.2. Deriving the Gradient Descent and Level-set Functions

The gradient descent of a function is defined as:

$$C_t = -\nabla E \tag{2}$$

We will use the following relationship to derive the gradient descent (2) of our energy functional defined in (1):

$$\frac{d}{dt}E(C) = \langle \nabla E, C_t \rangle = \int_C (C_t \cdot \nabla E) ds \tag{3}$$

This relationship notes that the time derivative of the energy functional can be rewritten as the inner product of ∇E and C_t. This inner product can also be redefined as the L_2 continuous norm defined over the contour. As a result, the time derivative of the energy functional can be simplified down to this form, which provides the ability to extract ∇E from the integrand once we have isolated C_t.

We will apply this method first to a generic non-linear function f_{in} integrated over the region inside the contour, which resembles several terms in the Chan-Vese energy functional in (1) :

$$\hat{E}(C) = \iint_R f_{in} dx dy \tag{4}$$

In order to use the relationship defined in (3), we need to rewrite the energy functional (4) in terms of an integral around the curve of the contour. If we set $f_{in} = \nabla \cdot \vec{F}$, this relationship can be found by application of the divergence theorem:

$$\hat{E}(C) = \iint_{\mathrm{R}} f_{in} dxdy = \iint_{\mathrm{R}} (\nabla \cdot \vec{F}) dxdy = \int_{C} \vec{F} \cdot N ds \tag{5}$$

Before we can take the time derivative of this integral, we need to replace all instances of the time-dependent variable s with the time-independent parametric variable p. This can be achieved with the following relationship: $\|C_p\| dp = ds$.

$$\hat{E}(C) = \int_{C} \vec{F} \cdot N ds = \int_{0}^{1} \vec{F} \cdot N \|C_p\| \, dp \tag{6}$$

For the rest of the derivation, the following numerical definition of the curve's unit normal and unit tangent components will be applied:

$$N = J \frac{C_p}{\|C_p\|}$$

$$T = \frac{C_p}{\|C_p\|} \tag{7}$$

Now, the time derivative of the energy functional can be applied to our energy functional. Since we removed the time-dependence by switching variables, we can bring the derivative inside the integral and instead take the derivative of the integrand with respect to t:

$$\frac{d}{dt}\hat{E}(C) = \frac{d}{dt}\int_{0}^{1} \left(\vec{F} \cdot N \|C_p\|\right) dp = \int_{0}^{1} \frac{d}{dt}\left(\vec{F} \cdot J \frac{C_p}{\|C_p\|} \|C_p\|\right) dp = \int_{0}^{1} \frac{d}{dt}\left(\vec{F} \cdot J C_p\right) \tag{8}$$

Using the product and chain rule for derivation, we arrive at the next step. In this step, we also switch the order of derivation for C_{pt} and use C_{tp} instead and by substituting in the Jacobian $\left[\frac{\partial \vec{F}}{\partial X}\right] C_t$ for $\frac{d\vec{F}}{dt}$:

$$\frac{d}{dt}\hat{E}(C) = \int_{0}^{1} \left(\frac{d\vec{F}}{dt} \cdot J C_p + \vec{F} \cdot J C_{pt}\right) dp = \int_{0}^{1} \left(\left[\frac{\partial \vec{F}}{\partial X}\right] C_t \cdot J C_p + \vec{F} \cdot J C_{tp}\right) dp \tag{9}$$

We now switch back to our original variable s by the use of the relationship, $\frac{d}{dp} = \frac{d}{ds}\|C_p\|$:

$$\frac{d}{dt}\hat{E}(C) = \int_{0}^{1} \left(\left[\frac{\partial \vec{F}}{\partial X}\right] C_t \cdot J C_s \|C_p\| + \vec{F} \cdot J C_{ts} \|C_p\|\right) dp = \int_{0}^{1} \left(\left[\frac{\partial \vec{F}}{\partial X}\right] C_t \cdot J C_s + \vec{F} \cdot J C_{ts}\right) ds \tag{10}$$

We can now apply integration by parts to the second term in the integrand of (10). This in turn results in the isolation and factoring of the variable C_t in our integrand:

$$\frac{d}{dt}\hat{E}(C) = \int_0^1 \left(\left[\frac{\partial \vec{F}}{\partial X}\right] C_t \cdot JC_s - \left[\frac{\partial \vec{F}}{\partial X}\right] C_s \cdot JC_t \right) ds = \int_0^1 C_t \cdot \left(\left[\frac{\partial \vec{F}}{\partial X}\right]^T JC_s + J\left[\frac{\partial \vec{F}}{\partial X}\right] C_s \right) ds \qquad (11)$$

The functional is now in a form similar to (3). The $\nabla\hat{E}$ can now be extracted from the integrand of (11) as follows:

$$\nabla\hat{E} = \left[\frac{\partial \vec{F}}{\partial X}\right]^T JC_s + J\left[\frac{\partial \vec{F}}{\partial X}\right] C_s = \left[\frac{\partial \vec{F}}{\partial X}\right]^T N + J\left[\frac{\partial \vec{F}}{\partial X}\right] T \qquad (12)$$

Since the energy functional defined in (1) is a geometric function, the tangential component of (12) can be assumed to be cancelled out after expanding the equation. The following simplification is implemented:

$$\begin{aligned}
\nabla\hat{E} &= \left(T^T \left[\frac{\partial \vec{F}}{\partial X}\right]^T N\right) T + \left(N^T \left[\frac{\partial \vec{F}}{\partial X}\right]^T N\right) N + \left(T^T J\left[\frac{\partial \vec{F}}{\partial X}\right] T\right) T + \left(N^T J\left[\frac{\partial \vec{F}}{\partial X}\right] T\right) N \\
&= \left(T^T \left[\frac{\partial \vec{F}}{\partial X}\right]^T N\right) T + \left(N^T \left[\frac{\partial \vec{F}}{\partial X}\right]^T N\right) N - \left(N^T \left[\frac{\partial \vec{F}}{\partial X}\right] T\right) T + \left(T^T \left[\frac{\partial \vec{F}}{\partial X}\right] T\right) N \\
&= \left(N^T \left[\frac{\partial \vec{F}}{\partial X}\right]^T N\right) N + \left(T^T \left[\frac{\partial \vec{F}}{\partial X}\right] T\right) N \\
&= \left(N^T \left[\frac{\partial \vec{F}}{\partial X}\right] N\right) N + \left(T^T \left[\frac{\partial \vec{F}}{\partial X}\right] T\right) N
\end{aligned} \qquad (13)$$

We can now see that only the normal component of the gradient influences the contour, which matches the behavior of a geometric functional. Note that the result in (13) can be viewed as a matrix equation, $a^T Aa + b^T Ab$, where a and b are part of an orthonormal basis and $A = \left[\frac{\partial \vec{F}}{\partial X}\right]$. Both N and T form an orthonormal basis, and A is diagonalizable. As a result, $a^T Aa + b^T Ab = trace(A)$. Applying this knowledge to (13) and using the original definition of the function $f_{in} = \nabla \cdot \vec{F}$, we now arrive at:

$$\nabla\hat{E} = trace\left(\left[\frac{\partial \vec{F}}{\partial X}\right]\right) N = trace\left(\begin{bmatrix} \frac{\partial \overrightarrow{F_1}}{\partial x} & \frac{\partial \vec{F}_1}{\partial y} \\ \frac{\partial \overrightarrow{F_2}}{\partial x} & \frac{\partial \overrightarrow{F_2}}{\partial y} \end{bmatrix}\right) N = \left(\nabla \cdot \vec{F}\right) N = f_{in} N \qquad (14)$$

With the result in (14), we can now define our gradient descent function:

$$C_t = -\nabla\hat{E} = -f_{in}N \tag{15}$$

Since the level-set implementation of the gradient set is being discussed in this paper, one final simplification brings us to our level-set function:

$$\Psi_t = f_{in}\|\nabla\Psi\| \tag{16}$$

The result found in (16) can now be applied to any part of the Chan-Vese energy functional in (1) that involves integrating over the area inside the contour. However, there is still a term that involves the integration over the area outside the contour. With a slight modification, the result in (16) can still be applied.

We first note that the integration outside of the curve is equivalent to the difference between the integration over the entire image and the integration over the region inside the curve for a given function:

$$\iint_{R^C} f_{out}dxdy = \iint_{\Omega} f_{out}dxdy - \iint_{\mathrm{R}} f_{out}dxdy \tag{17}$$

Note that the integration over the region of the entire image has no time dependence. When we take the time derivative of the functional as we did in (8), the first term of (17) equates to zero. This means we only need to consider the second term of (17), which using the same method to find (16), results in:

$$\Psi_t = -f_{out}\|\nabla\Psi\| \tag{18}$$

With the results noted in (16) and (18), three of the four terms in the Chan-Vese energy functional (1) from now have a level-set implementation defined. The final term left is the part of the function that defines the length of the contour, which acts as the diffusion term. We apply the same method of derivation to this function as we did for the other terms and begin by taking the time derivative of the length after switching out the time dependent variables:

$$\frac{d}{dt}\hat{E}(C) = \frac{d}{dt}\int_C ds = \frac{d}{dt}\int_0^1 \|C_p\|\,dp = \int_0^1 \|C_p\|_t \tag{19}$$

Now, we can use substitute $\|C_p\|_t = \left(\sqrt{C_p \cdot C_p}\right)_t = \frac{C_{pt}\cdot C_p}{\|C_p\|}$, simplify, and swap derivative parameters:

$$\frac{d}{dt}\hat{E}(C) = \int_0^1 \left(\frac{C_{pt}\cdot C_p}{\|C_p\|}\right)dp = \int_0^1 (C_{pt}\cdot T)dp = \int_0^1 (C_{tp}\cdot T)dp \tag{20}$$

Applying integration by parts and using the relationships defined in (7), the following equation is derived:

$$\frac{d}{dt}\hat{E}(C) = -\int_0^1 (C_t \cdot T_p)dp = -\int_0^1 (C_t \cdot T_s \|C_p\|)\frac{ds}{\|C_p\|} = -\int_0^1 (C_t \cdot T_s)ds \tag{21}$$

Factoring C_t, extracting $\nabla\hat{E}$ from our integrand as defined in (3), and noting that $T_s = kN$, where k is the curvature of the contour, the gradient descent function can be defined as:

$$C_t = -\nabla\hat{E} = kN \tag{22}$$

Using the value of curvature $k = -\nabla \cdot \left(\frac{\nabla\Psi}{\|\nabla\Psi\|}\right)$, the level-set function is found to be:

$$\Psi_t = -k\|\nabla\Psi\| = \nabla \cdot \left(\frac{\nabla\Psi}{\|\nabla\Psi\|}\right)\|\nabla\Psi\| \tag{23}$$

Finally, using the level-set implementations found in (16), (18), and (23) on the original energy functional in *(1)*, the level-set gradient descent equation for the Chan-Vese algorithm is computed as:

$$\Psi_t = \lambda_1(I-u)^2\|\nabla\Psi\| - \lambda_2(I-v)^2\|\nabla\Psi\| + \mu\nabla \cdot \left(\frac{\nabla\Psi}{\|\nabla\Psi\|}\right)\|\nabla\Psi\| + \nu\|\nabla\Psi\| \tag{24}$$

3. Discretization and Implementation

3.1. Defining the Discretization Scheme

In order to apply the level-set equation derived in the previous section to an image, a discretization scheme must be defined for the derivatives needed in this equation. Looking at (24), we can note that the desired function has three non-linear terms and one diffusion term. Therefore, the level-set functions requires the usage of two discretization schemes.

The first discretization scheme can be found by expanding out the diffusion term of (24):

$$\nabla \cdot \left(\frac{\nabla\Psi}{\|\nabla\Psi\|}\right)\|\nabla\Psi\| = \frac{\Psi_x^2\Psi_{yy} - 2\Psi_x\Psi_y\Psi_{xy} + \Psi_y^2\Psi_{xx}}{\Psi_x^2+\Psi_y^2} \tag{25}$$

This formula involves first- and second-order partial derivatives. Each of these derivatives are best computed by the use of a central difference scheme, as this scheme captures information from bordering pixels on both side of a given point and provides a more accurate approximation. The central difference equations used to approximate these derivatives are as follows:

$$\Psi_x(x,y,t) = \frac{\Psi(\mathrm{x}+\Delta x,y,t) - \Psi(\mathrm{x}-\Delta x,y,t)}{2\Delta x}$$
$$\Psi_{yy}(x,y,t) = \frac{\Psi(\mathrm{x},y+\Delta y,t) - 2\Psi(\mathrm{x},y,t) + \Psi(\mathrm{x},y-\Delta y,t)}{\Delta y^2} \tag{26}$$

$$\Psi_{xy}(x,y,t) = \frac{\Psi_x(\mathrm{x}, y+\Delta y, t) - 2\Psi_x(\mathrm{x}, y, t) + \Psi_x(\mathrm{x}, y-\Delta y, t)}{\Delta y^2}$$

By replacing the values in (25) with the respective equations found in (26), we can approximate the diffusion component of the level-set. It is important to note that the choices of Δx, Δy, and Δt influence the stability of this discretization scheme. Van Neumann analysis was used to determine the CFL condition for these terms, which demonstrate the relationship needed between the values of Δx, Δy, and Δt in order for this approximation to maintain stability. The CFL conditions for the discretization scheme of (25) is:

$$CFL\ Conditions\ for\ Cental\ Differences: \Delta t \leq \frac{\Delta x^2}{2}, \Delta t \leq \frac{\Delta y^2}{2} \tag{27}$$

For the non-linear terms, a different discretization scheme is required to calculate the norm of the level-set gradient. We cannot use central differences for this approximation, and we cannot simply use a forward or backward difference with no modifications. The partial derivatives of the level-set can come out to be positive or negative within the absolute value operator, which means the same scheme cannot be used for all values of the level-set. The nature of the non-linear terms' approximation depends on the sign of its coefficient and the signs of the approximation schemes used in it. To properly account for differences in sign, an upwind entropy scheme is needed. The upwind entropy scheme is as follows:

$$\alpha\|\nabla\Psi\| = \begin{cases} \alpha\sqrt{\max^2(0,\Psi_x^+) + \min^2(0,\Psi_x^-) + \max^2\left(0,\Psi_y^+\right) + \min^2(0,\Psi_y^-)}, & \alpha > 0 \\ \alpha\sqrt{\min^2(0,\Psi_x^+) + \max^2(0,\Psi_x^-) + \min^2\left(0,\Psi_y^+\right) + \max^2(0,\Psi_y^-)}, & \alpha < 0 \end{cases} \tag{28}$$

Within the upwind entropy scheme, both the backward and forward difference are used in the calculation of the norm of the level-set gradient. As a result, the discretization will now correctly be calculated depending on the sign of the coefficient and the signs of both differencing schemes. The backward and forward differences used in (28) can be approximated with following equations:

$$\begin{aligned} \Psi_x^+(x,y,t) &= \frac{\Psi(\mathrm{x}+\Delta x, y, t) - \Psi(\mathrm{x}, y, t)}{\Delta x} \\ \Psi_y^-(x,y,t) &= \frac{\Psi(\mathrm{x}, y, t) - \Psi(\mathrm{x}, y-\Delta y, t)}{\Delta y} \end{aligned} \tag{29}$$

Once again, to ensure the stability for this scheme for all approximations, the CFL condition must be fulfilled when choosing the parameters. The CFL condition of this discretization scheme is:

$$CFL\ Condition\ for\ Upwind\ Entropy\ Scheme: \Delta t \leq \frac{\Delta x^2}{\sqrt{2}}, \Delta t \leq \frac{\Delta y^2}{\sqrt{2}} \tag{30}$$

Since we have two different discretization schemes being used, we have two different CFL conditions that need to be fulfilled in order for the overall scheme to be stable. Therefore, for the overall discretization scheme, we must fulfill the most restrictive of these CFL conditions, which will correspond to the diffusion term's CFL found in (27).

3.2. *Implementing the Chan-Vese Algorithm in MATLAB*

The basic algorithm used to code the Chan-Vese algorithm is as follows:

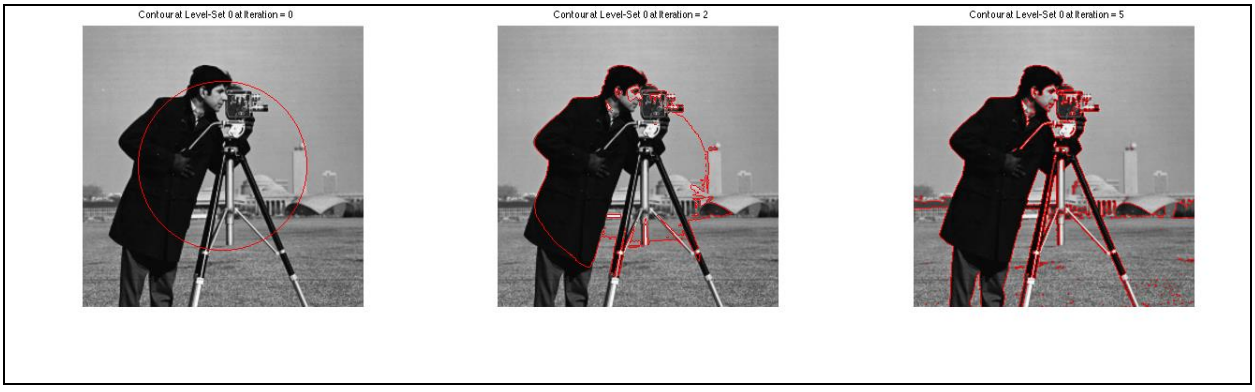
1. Initialize the parameters λ_1, λ_2, μ, ν and Δt
2. Initialize a level-set function with a desired shape and location for the contour
3. Using the regions defined by the current contour, calculate values for u and v
4. Calculate the resulting Ψ_t using the discretization schemes in (25) and (28) and the values of u and v from step 3
5. Calculate $\Psi^{n+1} = \Psi^n + \Delta t * \Psi_t$ in order to evolve the contour
6. Repeat steps 3-5 until desired segmentation

In order to implement the above algorithm in MATLAB, the parameters Δx, Δy, and Δt used to define the discretization scheme must be determined. Setting $\Delta x = \Delta y = 1$ aided in simplifying the code used to calculate the partial derivatives in our gradient descent function, as this selection allowed the calculations to be computed using MATLAB's robust matrix indexing and operations. Plugging in these chosen values for Δx and Δy into our CFL condition from (30), we determined that $\Delta t \leq 0.5$. For our implementation, we set $\Delta t = 0.2$, which falls well within our CFL condition. In addition, unless otherwise stated, we used the default scaling values of $\lambda_1 = \lambda_2 = 1$, $\mu = 0.5$ and $\nu = 0$ for our experiments. The code used for this implementation of the Chan-Vese algorithm is documented in Appendix A: MATLAB Source Code.

4. Experiments and Results

4.1. *Performance in Regular Images*

For the following two images, we applied the Chan-Vese algorithm using the default scaling values discussed and a simple circular level-set to see if the algorithm could achieve proper edge segmentation. Figure 1 below shows the result of the algorithm when it is applied to the "Cameraman" image.



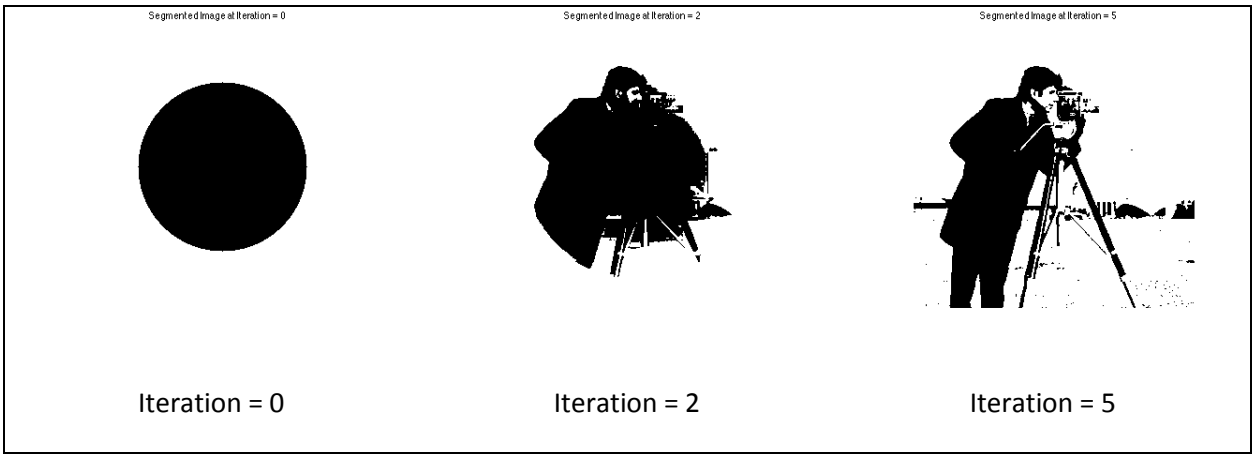


Figure 1. Application of the Chan-Vese Algorithm on Benchmark Image "Cameraman".

From the evolution of the contour detailed in the figure, it is apparent how quickly the algorithm evolved our chosen contour to the correct segmentation. The cameraman in the image and his camera are enclosed almost perfectly by the evolving contour.

We can further examine the application of the algorithm by viewing how the energy, which we are trying to minimize, changes at each iteration step. Figure 2 below shows the corresponding energy values found while applying the Chan-Vese algorithm to the image in Figure 1.

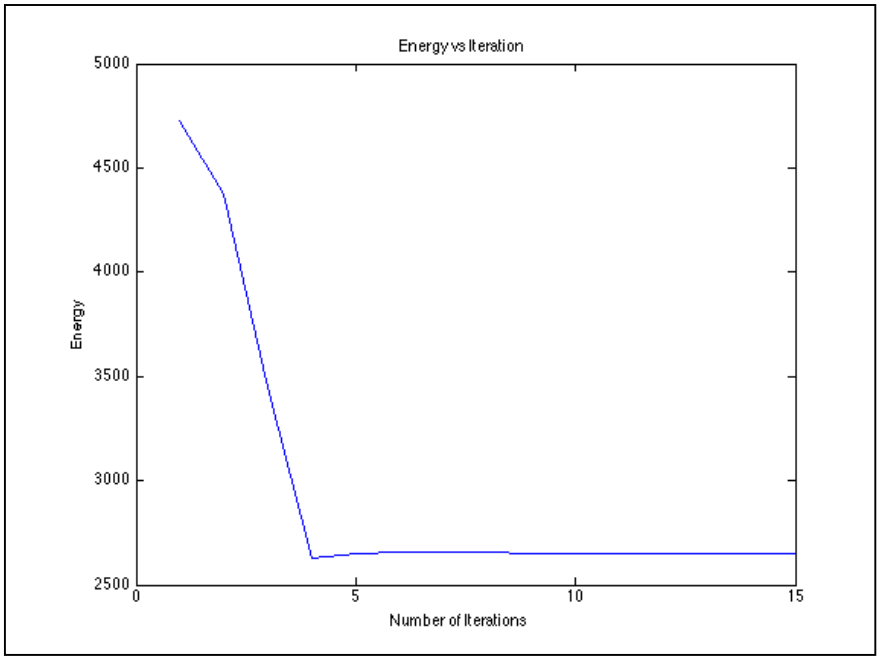


Figure 2. Value of the Energy Functional for "Cameraman" as a Function of Iteration.

Of note in this graph is how quickly the energy becomes minimized around the fourth iteration, which corresponds to when our contour fully evolves into the proper segmentation. Once that segmentation

is achieved, the gradient descent of our functional reaches steady-state in the subsequent iterations. This behavior is visibly validated by the energy plot above.

Next, we apply the Chan-Vese algorithm to a zoomed image of a corral. This image provides a good target for the algorithm as it is predominantly composed of significant segments. Figure 3 below shows the result of our implementation on this image.

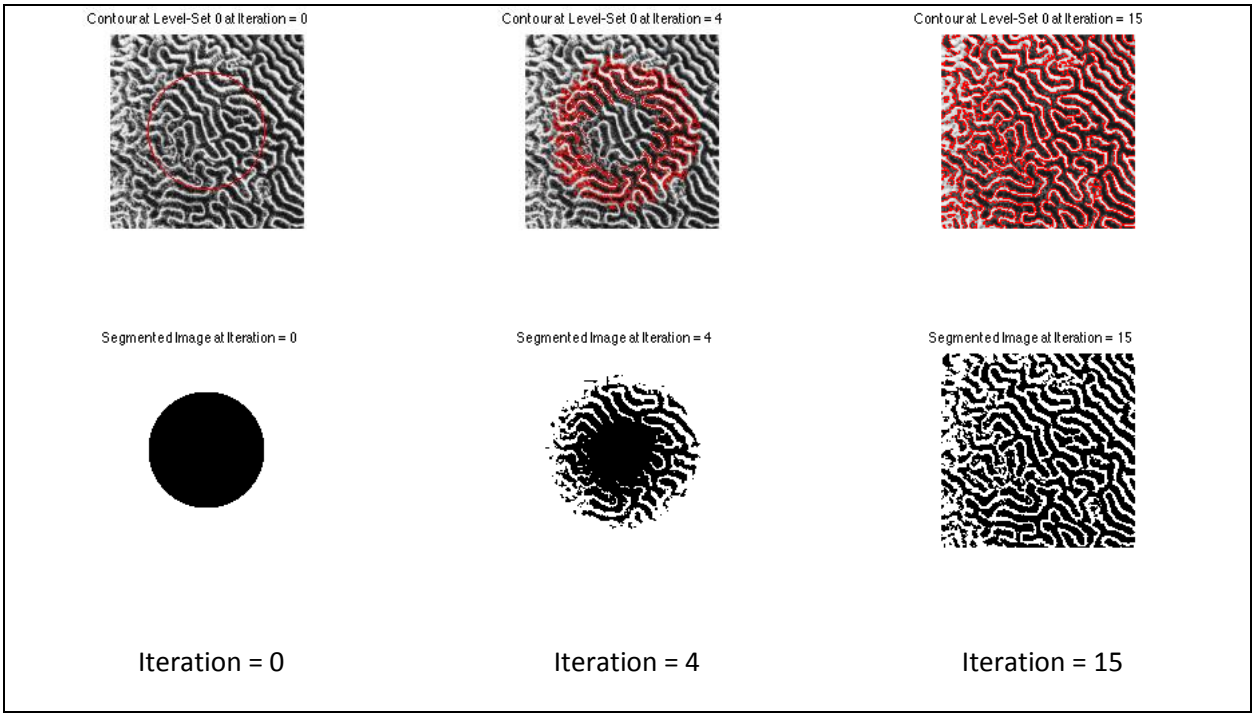


Figure 3. Application of the Chan-Vese Algorithm on Image "Corral".

Once again, the Chan-Vese algorithm is successful at extracting the segmentation in the image in a quick manner. By iteration 15, we have extracted a vast majority of the segments present in the image. We can verify the performance of the algorithm by examining its energy function over time in Figure 4.

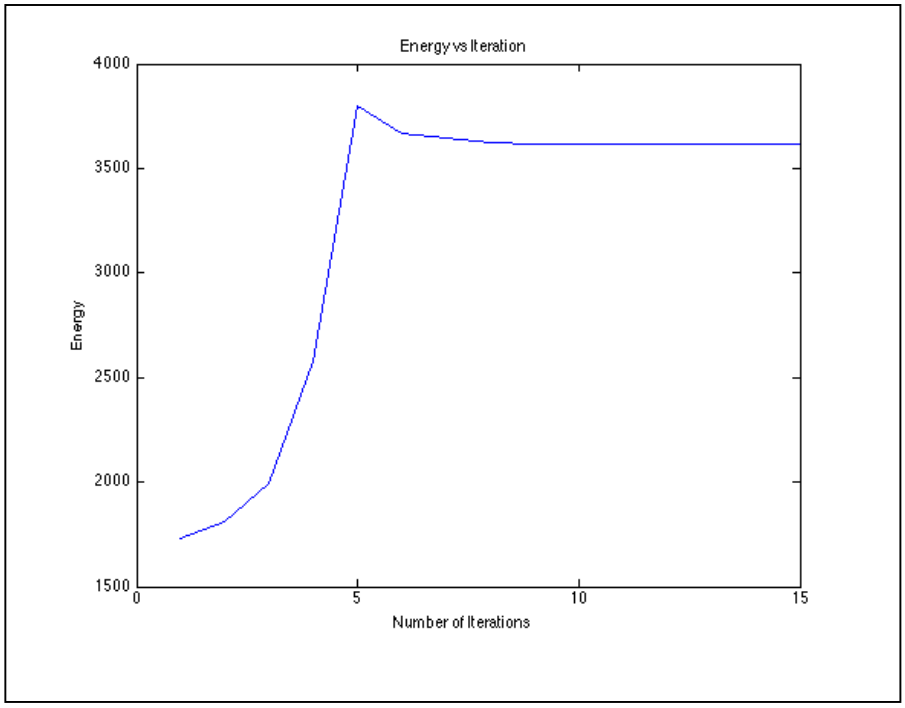
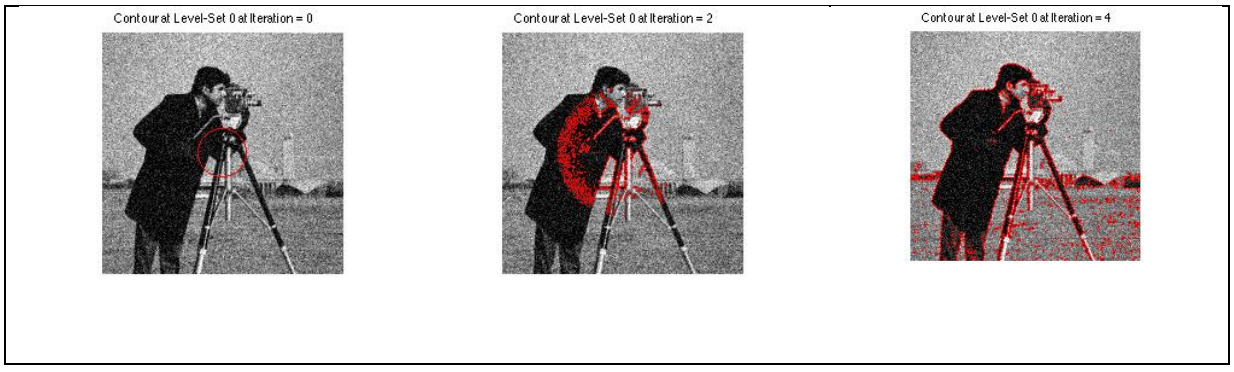


Figure 4. Value of the Energy Functional for "Corral" as a Function of Iteration.

In this plot, we note that the gradient descent achieves steady state around the fifth iteration, where it achieves its minimum value. However, this plot differs from the plot of the previous image in Figure 1 in that the energy rises greatly at first before reach steading state. This phenomenon is explained by the difference in our level-set contour and the amount of segments present in the image. As the contour evolves, it is natural for its energy to increase greatly because the contour must spread to segment a majority of the image. This causes the energy to rise before it can fall and settle on a minimum value. In both cases discussed, the Chan-Vese algorithm provides excellent results on clean images with notable segments.

4.2. Performance in Noisy Images

Since there is no guarantee that an input image to the function will be clean, it is vital to examine the results of the algorithm on noisy images. Figure 5 below shows the results after it is applied to an image with Gaussian noise.



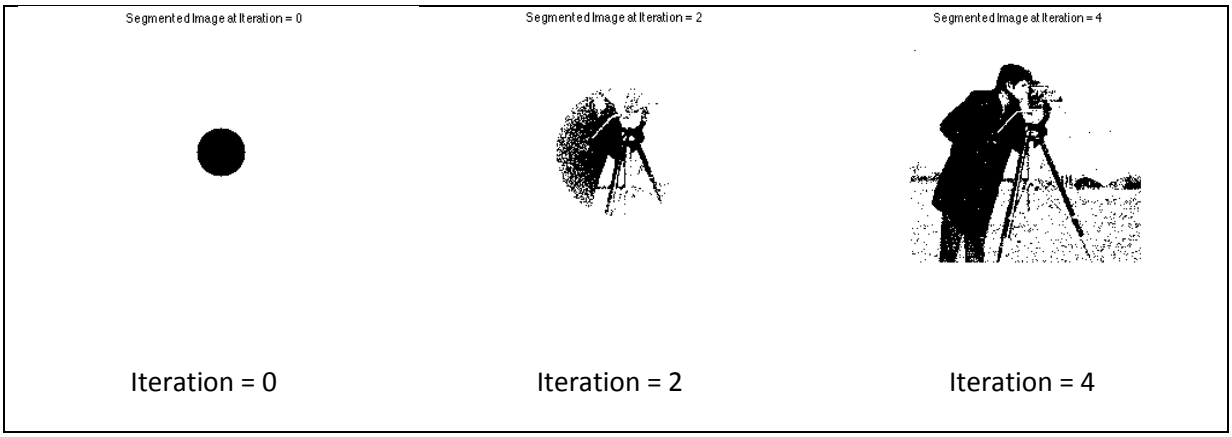


Figure 5. Application of Chan-Vese Algorithm on an Image with Gaussian Noise.

Overall, the algorithm extracts the major segments by its fourth iteration. Despite its success with the major segment, the algorithm appears to have picked several tiny segments of pixels appearing in the lower background of the image. From this sample, we can note that noise will cause other fragments of segments to be enclosed, but the significant objects will still be captured by the algorithm.

To further explore the effect of noise on the algorithm, we apply it to an image with "salt & pepper" noise. The results of this experiment are shown below in Figure 6.

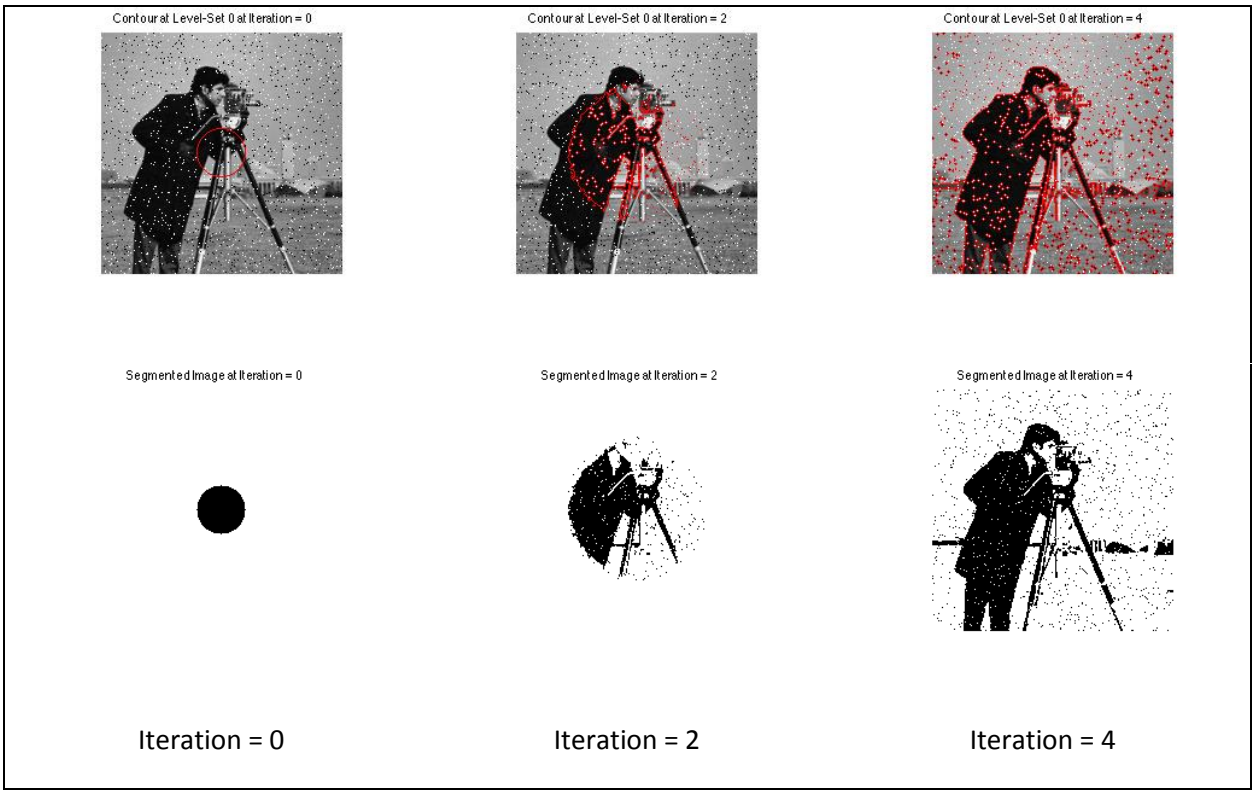


Figure 6. Application of the Chan-Vese Algorithm on an Image with "Salt & Pepper" Noise.

The results from this experiment demonstrate an interesting contrast with the other noisy image. Like the other experiment, this one also resulted in the major segments to be enclosed and extracted. However, due to the nature of this noise, the Chan-Vese algorithm was greatly affected by the "salt & pepper" values appearing throughout the entirety of the background, causing it to enclose extra pixel segments throughout the image. These extra segments are less desirable than the extra segments found while processing a Gaussian noisy image.

4.3. Effect of Area Penalty in Energy Functional

In the past two experiments, we have defaulted to setting the value of the final scaling factor to $\nu = 0$. This scaling factor corresponds to the area penalty in our energy functional defined in (1). Compared to the other penalties in the equation, the area penalty has very unique effects on the algorithm. Figure 7 below demonstrates at a fixed iteration and fixed values for the other scaling factors how the value of ν affects the contour evolution on a visibly segmented image.

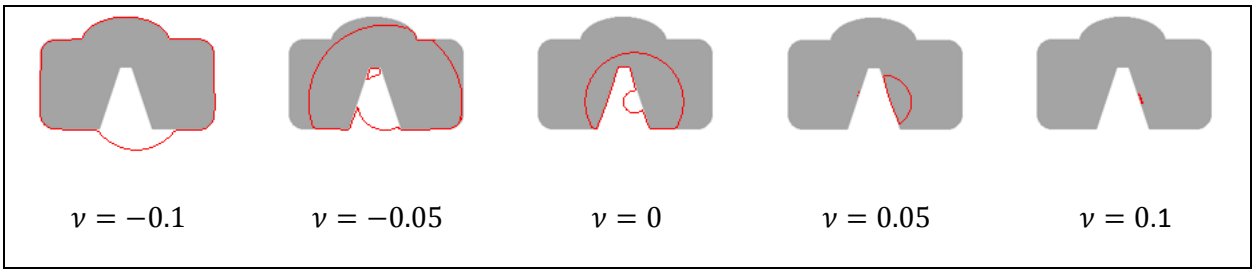


Figure 7. Result of Chan-Vese Algorithm Using Various Scaling Values for the Area Penalty.

From the results above, there are several observations we can make about the value of ν. When $\nu = 0$, we implement the same contour evolution as shown in the past few examples. If we keep iterating this evolution, we will eventually obtain a proper segmentation for the image. However, if ν is non-zero, the area penalty will being taking effect. At a positive value for ν, the area penalty will be aiming to minimize the area contained by the contour. As can be seen in the figure, increasing the value of ν in the positive direction eventually forces the contour to become invalid. The opposite occurs when ν goes in the negative direction. As shown above, when ν becomes more negative, the function will be growing towards contain as much area as possible. This in turn leads the contour to expand past areas where the edges exist and eventually expand to contain the entire image. Neither of these extreme behaviors are desirable for locating segmentation in images. From these observations, we can see that the value of the area penalty in our energy functional is sensitive and needs to be applied with great care. The nature of how the area penalty is mathematically defined might be the reason it can influence the energy more than the other terms, as it is defined as a scaled normed of the level-set gradient. This value has the potential to be quite strong depending on the current value of the level-set.

5. Conclusions

5.1. Strengths and Weaknesses of the Chan-Vese Algorithm

Overall, the Chan-Vese algorithm seems to be a value tool in the area of image segmentation. As shown in most of our results, the biggest strength of the Chan-Vese algorithm is how quickly it can reach steady state while still capturing the information desired in an image. However, through the experiments

conducted, we also found that Chan-Vese does have some susceptibility to certain types of noise in images, one of which is “salt & pepper” noise. Since the Chan-Vese model attempts to create uniform regions of intensity, having noise spread throughout the image can trick the algorithm into creating regions that do not have actual segments. In addition, we discovered how sensitive the function becomes when we take into consideration the area enclosed by the contour into our energy functional. This area penalty that we attempt to minimize has the potential to create extreme results that invalidate the contour evolution as the value of the area penalty can quickly exert more influence than the other factors in the energy functional.

5.2. Potential Improvements on the Implementation

During the implementation of the method, we noted that as the gradient descent equation is applied to the image, the range of the values found in the level-set equation can exponentially increase. This behavior is a potential from of instability and a possible detriment if the algorithm did not converge to steady state so quickly. Therefore, an exploration of how and why the level-set tends towards positive and negative infinity and finding a method to restrict this growth would prove to be a worthwhile endeavor. In addition, many implementations of this algorithm include the concept of reinitializing the level-set to level zero at certain intervals during the algorithms calculation. This can be a necessity as the level-set has a tendency to become flat in its values, and a redefinition of its zero level contour may also aid in the overall stability of the algorithm. If there was more time given for this project, this level-set re-initialization would have been implemented in our program.

6. References

[1] M. Kass, A. Witkin, and D. Terzopoulos. Snakes: Active contour models. International journal of computer vision, 1(4):321–331, 1988.

[2] T. Chan and L. Vese. Active contours without edges. IEEE Transactions on image processing, 10(2):266– 277, 2001.

7. Appendix A: MATLAB Source Code

```
% Rahmaan Lodhia
% ECE 6560
% Final Project Chan-Vese Algorithm
% Main Run File

% Set parameters to use for Chan-Vese calculation
% Desired nxn size of sample
n = 300;

%Set to 1 if noise is desired in image
addNoise = 0;

% Load and scale image and add noise if applicable
filename = 'Images/cameraman.png';
I = imread(filename);
scaleI = imresize(I, [n, n]);
if addNoise == 1
    scaleI = imnoise(scaleI, 'gaussian');
    f0 = double(scaleI)./max(max(double(scaleI)));
else
    f0 = double(scaleI)./max(max(double(scaleI)));
end


% Creates a grid of circle contours depending on k
% For this project, only 1 circle was created as the level-set
[Y,X] = meshgrid(1:n,1:n);
k = 1; %number of circles
r = .1*n/k;
phi0 = zeros(n,n)+Inf;
for i=1:n
    for j=1:n
        c = ([i j]-1)*(n/k)+(n/k)*.5;
        phi0 = min( phi0, sqrt( (X-c(1)).^2 + (Y-c(2)).^2 ) - r );
    end
end

% Set time parameters
deltaT = 0.2;
maxT = 0.8;

% Set scaling parameters for energy terms
lambda1 = 1.0;
lambda2 = 1.0;
mu = 0.5;
nu = 0.0;

% Compute segmentation with Chan_Vese
[phi, niter, u, v, E] = chan_vese(f0, phi0, deltaT, maxT, lambda1, lambda2, mu, nu);

% Plot final curvature result
figure
imshow(scaleI); colormap(gray); hold on
title(sprintf('Contour at Level-Set 0 at Iteration = %d', niter));
contour(phi, [0 0], 'r');
hold off

% Create final segment plot
figure
segment = zeros(n,n);
segment(phi > 0) = 1;
segmentI = mat2gray(segment);
imshow(segmentI)
title(sprintf('Segmented Image at Iteration = %d', niter));

% Plot Energy vs. Iteration Curve
figure
plot(1:niter,E);
xlabel('Number of Iterations')
ylabel('Energy')
title('Energy vs Iteration')
```

```
% Rahmaan Lodhia
% ECE 6560
% Final Project Chan-Vese Algorithm
% chan_vese.m

function [phi, niter, u, v, E] = chan_vese(I, phi0, tau, maxT, lambda1, lambda2, mu, nu)
% Applies the Chan-Vese algorithm to the Image I with initial level-set phi

% Initialize variables
% Number of iterations
niter = round(maxT/tau);

% Initialize Energy vector
E = zeros(niter,1);

% Set initial levelset
phi = phi0;

for i = 1:niter
    % Find current values for u and v using fixed curve defined by current
    % phi

    % Average intensity inside contour
    u = sum(sum(I(phi < 0)))/length(I(phi < 0));

    % Average intensity outside contour
    v = sum(sum(I(phi > 0)))/length(I(phi > 0));

    % Calculate curvature value K
    K = curvature(phi);

    % Determine the phi at the next time step using the defined PDE
    phi = phi + tau.*lambda1.*((I-u).^2).*upwindEntropyNorm(phi,1) - tau.*lambda2.*((I-
v).^2).*upwindEntropyNorm(phi,-1) - tau.*mu.*K + tau.*nu.*upwindEntropyNorm(phi,1);

    % Calculate energy in current level-set
    intesnityInside = (I-u).^2;
    intensityOutside = (I-v).^2;
    c = contourc(phi, [0 0]);
    E(i) = lambda1.*sum(sum(intesnityInside(phi < 0))) + lambda2.*sum(sum(intensityOutside(phi > 0))) +
mu.*contourLength(c) + nu.*length(intesnityInside(phi < 0));
end

end
```

```
% Rahmaan Lodhia
% ECE 6560
% Final Project Chan-Vese Algorithm
% curvature.m

function K = curvature(phi)
% Approximate curvature using central difference scheme
% deltaX and deltaY are both equal to 1

Phi_x = ( phi([2:end 1],:,:)-phi([end 1:end-1],:,:) )./2;
Phi_y = ( phi(:,[2:end 1],:)-phi(:,[end 1:end-1],:) )./2;
Phi_xy = ( Phi_x(:,[2:end 1],:)-Phi_x(:,[end 1:end-1],:) )./2;
Phi_xx = ( phi([2:end 1],:,:)- 2.*phi + phi([end 1:end-1],:,:) )./ 1;
Phi_yy = ( phi(:,[2:end 1],:)- 2.*phi + phi(:,[end 1:end-1],:) )./ 1;

% Calculate K with partial derivaties
% For the denominator, eps is used to avoid NaN results
K = -(Phi_xx.*(Phi_y.^2) - 2.*Phi_x.*Phi_y.*Phi_xy + Phi_yy.*(Phi_x.^2))./(max(eps,
((Phi_x.^2)+(Phi_y.^2))));

end
```

```
% Rahmaan Lodhia
% ECE 6560
% Final Project Chan-Vese Algorithm
% upwindEntropyNorm.m

function norm = upwindEntropyNorm(phi, sign)
% Calculate the norm of the gradient of the level set using the upwind
% entropy scheme
% deltaX and deltaY are both equal to 1

% Calculate forward and backward derivatives
DxF = phi([2:end 1],:,:)-phi;
DxB = phi - phi([end 1:end-1],:,:);
DyF = phi(:,[2:end 1],:)-phi;
DyB = phi - phi(:,[end 1:end-1],:);

% Depending on input sign, use the specified entropy scheme
if sign == 1
    norm = (sum(sum((max(DxF,eps)).^2 + (min(DxB,eps)).^2 + (max(DyF,eps)).^2 +
(min(DyB,eps)).^2))).^(1/2);
elseif sign == -1
    norm = (sum(sum((min(DxF,eps)).^2 + (max(DxB,eps)).^2 + (min(DyF,eps)).^2 +
(max(DyB,eps)).^2))).^(1/2);
end

end
```

```
% Rahmaan Lodhia
% ECE 6560
% Final Project Chan-Vese Algorithm
% contourLength.m

function len = contourLength(c)
% Take a contour matrix defined at level 0 and compute its total length

currentIndex = 1;
len = 0;

while currentIndex < length(c)
    if c(1,currentIndex) == 0
        for i = (currentIndex+1):(currentIndex+c(2,currentIndex)-1)
            len = len + sqrt((c(1,i)-c(1,i+1)).^2 + (c(2,i)-c(2,i+1)).^2);
        end
        currentIndex = i+1;
    end

    currentIndex = currentIndex + 1;
end


end
```