Binary Image Segmentation using Chan-Vese Algorithm

Irfan Al-Hussaini

Contents

List of Figures

List of Tables

Listings

1 Introduction

Image segmentation is the process of partitioning an image into multiple meaningful segments consisting of a set of pixels. The aim is to represent the image in a way that is easier to analyze or understand and can be used to locate particular objects in images. The idea is to assign labels to the image such that those with the same labels share some meaningful characteristic or desirable computed property. The resulting contour This has applications in medical imaging, object detection, surveillance, traffic control systems, retrieving images based on content, etc. Image segmentation on stacks of images has also been used to create 3D reconstructions using the resulting segmentations.
This project attempts to implement an image segmentation algorithm to partition an image into binary segments. The aim is to extract the desired segment while making it resistant to different noise. The following section expands on the particular type of image segmentation used in this project.

2 Active Contour

Active contour is one such image segmentation approach which uses an energy minimizing spline. Using a formulation that pull the edge of evolving contours towards object contours and forces that resist deformation, the contours are evolved [4]. It can be thought of as matching a model that can deform to a sample image using energy minimization.
This project explores a particular image segmentation algorithm by Chan et al [3]. The integral for energy of this approach is shown in equation 1. When minimizing this energy, the first two terms in this region based active contour attempts to shrink and expand the contour. Balancing these two would ideally obtain the boundary of our interested object but this does not take into account the effect of noise. Since there is no length penalty term, there is no necessity to ignore the finer noise particles that the image contains. The last term in this equation takes this into account and attempts to restrict the contour from conforming to these grainy particles.

$$E(C, u, v) = \int_{R_{in}} (I - u)^2 \, dx \, dy + \int_{R_{out}} (I - v)^2 \, dx \, dy + \lambda \int_C dS \tag{1}$$

To actually use this energy to solve an image segmentation problem, a gradient descent time evolving partial differential equation has to be developed. The development of this and the origin of the equation above is shown in the following section.

3 Formulation of Partial Differential Equation

In this particular implementation of image segmentation, Chan-Vese algorithm was used. Before the equation can actually be of any use, the time evolution of the contour must be developed so that it can converge to the desired segment. In order to do this, an update equation for the curve, C, must be developed. The development of this time evolution is shown in this section.
The objective is to bring the energy integral into a form so that the time derivative can be expressed in the following way,

$$\frac{d}{dt} E(C) = \langle \nabla E, C_t \rangle \tag{2}$$

Giving the following simple formulation when the energy is taken at turning points of the energy curve by equating the equation above to zero,

$$C_t = -\nabla E \tag{3}$$

Considering an energy integral as shown below,

$$E(C) = \iint_{R_{in}} f_{in}\, dx\, dy \tag{4}$$

Now, f_{in} can be thought of as the divergence of a vector $\vec{F}$,

$$f_{in} = \nabla.\vec{F} \tag{5}$$

The divergence theorem states,

$$\int_C \vec{F}.\vec{N} = \iint_{R_{in}} \nabla.\,\vec{F}\, dx\, dy \tag{6}$$

Choosing vector field $\vec{F}$ such that,

$$f_{in} = \nabla.\vec{F} \tag{7}$$

So, the energy in terms of this vector $\vec{F}$ and the normal, N,

$$E = \iint_{Rin} \nabla.\vec{F}\, dx\, dy = \int_C \vec{F}.\,\vec{N}\, dS \tag{8}$$

Differentiating with respect to t,

$$\frac{dE}{dt} = \frac{d}{dt}\int_C \vec{F}.\,\vec{N}\, dS \tag{9}$$

But the arc-length dS changes with time. The parameter p is chosen such that it does not depend on time. To switch the order of integration and differentiation, the part inside the integration must be independent of time. So, parameter p is substituted,

$$\begin{aligned} \frac{dE}{dt} &= \frac{d}{dt}\int_0^1 F(C(\vec{p},t)).\,N(\vec{p})\, \|C_p\|\, dp \\ &= \int_0^1 \frac{d}{dt}\vec{F}.\,\vec{N}\, \|C_p\|\, dp \end{aligned} \tag{10}$$

From this point onwards, the vectors headers will be excluded and assumed where applicable. Expanding the above equation in terms of the new parameterization, p,

$$\frac{dE}{dt} = \int_0^1 [DF]\, C_t.\, N\, \|C_p\| \;+\; F.\, N_t\, \|C_p\| \;+\; F.\, N\, \|C_p\|_t\, dp \tag{11}$$

where D is the Jacobian of F. Here the values of $\|C_p\|_t$ and N_t are as follows,

$$\|C_p\|_t = (\sqrt{C_p.C_p})_t = \frac{C_{pt}.C_p}{\sqrt{C_p.C_p}} = C_{pt}.T \tag{12}$$

$$
\begin{aligned}
N_t = (JT)_t &= (J\frac{C_p}{\|C_p\|})_t \\
&= J\frac{C_{pt}}{\|C_p\|} - \frac{C_p}{\|C_p\|^2}\|C_p\|_t \\
&= \frac{J}{C_p}\left(C_{pt} - T\left(C_{pt}.T\right)\right)
\end{aligned} \tag{13}
$$

$T\left(C_{pt}.T\right)$ is the tangential component of C_{pt}. Since the tangential component is subtracted, the result is the normal component $(C_{pt}.N)N$

Since, $JN = -T$,

$$
\therefore N_t = \frac{J}{C_p}(C_{pt}.N)N = -\frac{(C_{pt}.N)T}{\|C_p\|} \tag{14}
$$

$$
\frac{dE}{dt} = \int_0^1 C_t\,.\,\left([DF]^T\,N\right)\,\|C_p\| \;+\; F.\,N\,(C_{pt}.\,T) \;-\; (F.\,T)(C_{pt}.N)\,dp \tag{15}
$$

Since $C_{tp} = C_{ts}\|C_p\|$ and $dS = \|C_p\|dp$,

$$
\therefore \frac{dE}{dt} = \int_0^1 C_t\,.\,\left([DF]^T\,N\right)\,+\,F.\,N\,(C_{ts}.\,T)\,-\,(F.\,T)(C_{ts}.N)\,dS \tag{16}
$$

Expanding the following terms above,

$$
\begin{aligned}
(F.N)(C_{ts}.T) &= -(F.N)(C_t.T_s) - (F.N_s)(C_t.T) - (F_s.N)(C_t.T) \\
-(F.T)(C_{ts}.N) &= (F.T)(C_t.N_s) + (F.T_s)(C_t.N) + (F_s.T)(C_t.N)
\end{aligned} \tag{17}
$$

and Frenet-Serret equations: $T_s = \kappa N$ and $N_s = -\kappa T$. Adding the two above equations and applying the Frenet-Serret equations,

$$
\begin{aligned}
(F.N)(C_{ts}.T) - (F.T)(C_{ts}.N) = \\
&- (F.N)(C_t.N) + \kappa(F.T)(C_t.T) - (F_s.N)(C_t.T) \\
&- \kappa(F.T)(C_t.T) + \kappa(F.N)(C_t.N) + (F_s.T)(C_t.N)
\end{aligned} \tag{18}
$$

$$
\frac{dE}{dt} = \int_0^1 C_t.\left([DF]^T N\right) - (C_t.T)([DF]^T.N) + (C_t.N)([DF]^T.T)\,dS \tag{19}
$$

Since we want to go in the direction of steepest slope, the above equation is evaluated to zero and the following is obtained for the part to the right of the dot products with C_t terms,

$$
\nabla E = [DF]^T N - (C_t.T)([DF]^T.N) + (C_t.N)([DF]^T.T)\,dS \tag{20}
$$

Since, $E(C) = \iint_{R_{in}} f\,dx$,

$$
\therefore \frac{dE}{dt} = \frac{d}{dt}\int_C F.N\,dS = \int (C_t.N)\left[T^T\,[DF]\,T + N^T\,[DF]\,N\right]\,dS \tag{21}
$$

In the above equation, $\left[T^T\,[DF]\,T + N^T\,[DF]\,N\right]$ is actually,

$$
trace([DF]) = \nabla.F = f \tag{22}
$$

giving,

$$\nabla_C E = f_{in} N \tag{23}$$

For area outside the curve,

$$E = \iint_{R_{out}} f_{out}\, dx = \int_{\Omega} f_{out}\, dx - \int_{R_{in}} f_{out}\, dx \tag{24}$$

where Ω depends on the whole image domain and not on the curve.

$$\therefore \nabla_C E = -f_{out} N \tag{25}$$

Using these, the development of the Chan Vese algorithm is shown in the following subsection.

3.1 Chan Vese

Our particular implementation involves the Chan-Vese algorithm [3]. u is the mean of the image value inside the contour and v is the mean of the image value outside the contour. In the equation below, $(I-u)^2$ refer to f_{in} and $(I-v)^2$ refer to f_{out}, referencing the mean square match,

$$E(C,u,v) = \int_{R_{in}} (I-u)^2\, dx\, dy + \int_{R_{out}} (I-v)^2\, dx\, dy \tag{26}$$

The derivative with respect to time is thus,

$$\begin{aligned} C_t &= -\nabla_C E \\ &= \left[-(I-u)^2 + (I-v)^2\right] N \\ &= \left[-I^2 + 2Iu - u^2 + I^2 - 2vI + v^2\right] N \\ &= \left[2I(u-v) - (u+v)(u-v)\right] N \\ &= 2I(u-v)\left[I - \frac{u+v}{2}\right] N \end{aligned} \tag{27}$$

However, this equation does not involve a length penalty term. So the evolving contour would not encapsulate noisy grains in the image rather wrap around those creating extra undesired segmentations. To get rid of this problem, an extra penalty term is introduced as shown in the equation below,

$$E(C,u,v) = \int_{R_{in}} (I-u)^2\, dx\, dy + \int_{R_{out}} (I-v)^2\, dx\, dy + \lambda \int_C dS \tag{28}$$

Combining the above with equation 27,

$$C_t = 2(u-v)\left[I - \frac{u+v}{2}\right] N - \lambda \kappa N \tag{29}$$

The following section details the time evolving level set formulation which can be actually applied to a digital image.

3.2 Level Set

Using the following transformations from curve evolution to level set evolution,

$$C_t = \kappa N \quad \Rightarrow \psi_t = \nabla . \left(\frac{\nabla \psi}{\|\nabla \psi\|} \right) \|\nabla \psi\| \tag{30}$$

$$C_t = N \quad \Rightarrow \psi_t = \|\nabla \psi\| \tag{31}$$

Expanding equation 30,

$$\begin{aligned} \psi_t &= \nabla . \left(\frac{\nabla \psi}{\|\nabla \psi\|} \right) \|\nabla \psi\| \\ &= \frac{\psi_x^2 \psi_{yy} - 2\psi_x \psi_y \psi_{xy} + \psi_y^2 \psi_{xx}}{\psi_x^2 + \psi_y^2} \end{aligned} \tag{32}$$

Tying it all together and since the direction of normal considered was outward normal, there is a further change of sign the following gradient descent equation of the algorithm is obtained for the level set,

$$\begin{aligned} \psi_t &= -2(u - v) \left[I - \frac{u + v}{2} \right] \|\nabla \psi\| + \lambda \nabla . \left(\frac{\nabla \psi}{\|\nabla \psi\|} \right) \|\nabla \psi\| \\ &= \left((I - u)^2 - (I - v)^2 \right) \|\nabla \psi\| + \lambda \nabla . \left(\frac{\nabla \psi}{\|\nabla \psi\|} \right) \|\nabla \psi\| \end{aligned} \tag{33}$$

For ease of fining an appropriate value of λ a weight of $(1 - \lambda)$ was applied to the first two terms. The value of λ is restricted to $0 \leq \lambda \leq 1$ resulting in the following final equation,

$$\psi_t = (1 - \lambda) \left((I - u)^2 - (I - v)^2 \right) \|\nabla \psi\| + \lambda \nabla . \left(\frac{\nabla \psi}{\|\nabla \psi\|} \right) \|\nabla \psi\| \tag{34}$$

The following section expands on the discretization with code snippets for each discretization scheme used.

4 Discretization

4.1 Initial Level Set

The initial ψ is generated as circular level sets. The code snippet in Listing 1 generated the level sets. It starts by finding the center of the contour or contours depending on how many is desired. The centers are equally spaced and the circumference of zero level is determined using the desired radius. The level sets are monotonically decreasing towards the center of each circle from the circumference and monotonically increasing away from the circumference.

Listing 1: MATLAB code to generate initial ψ

```
% Create two matrices where each point has the value
% of the coordinate split into corresponding positions
% in X and Y matrices
X = zeros(n,n);
Y = zeros(n,n);
```

```
for i = 1 : n
    for j = 1 : n
        X(i,j) = j;
        Y(i,j) = i;
    end
end

% Generate contours with equally spaced centers
k=1;
psiInit(1,:,:)=zeros(n,n);
for i = 1:nContours
    for j = 1:nContours
        f1=1+(i-1)*2;
        f2=1+(j-1)*2;
        centerX = n/nContours/2*f1;
        centerY = n/nContours/2*f2;
        psiInit(k,:,:) = sqrt((X-centerX).^2 + (Y-centerY).^2)-radius;
        k=k+1;
    end
end
% Find the minimum of each point to get overall initial contour
psi = squeeze(min(psiInit,[],1));
```

4.2 Central Difference: $\nabla.\left(\frac{\nabla\psi}{\|\nabla\psi\|}\right)\|\nabla\psi\|$

A central difference scheme is used for the discretization of the level set evolution of equation 30 and 32 as this is equivalent to the geometric heat equation which can be thought of as the linear heat equation extended to a perpendicular set axes, i.e. the normal and tangential component with one of them removed. Thus, the intuition behind selecting a central difference scheme is the same as that for a linear heat equation.

$$\begin{aligned}
\psi_x(x,y,t) &= \frac{\psi(x+\Delta x,y,t)-\psi(x-\Delta x,y,t)}{2\Delta x} \\
\psi_y(x,y,t) &= \frac{\psi(x,y+\Delta y,t)-\psi(x,y-\Delta y,t)}{2\Delta y} \\
\psi_{xx}(x,y,t) &= \frac{\psi(x+\Delta x,y,t)-2\psi(x,y,t)+\psi(x-\Delta x,y,t)}{\Delta x^2} \\
\psi_{yy}(x,y,t) &= \frac{\psi(x,y+\Delta y,t)-2\psi(x,y,t)+\psi(x,y-\Delta y,t)}{\Delta y^2} \\
\psi_{xy}(x,y,t) &= \frac{\psi_x(x,y+\Delta y,t)-\psi_x(x,y-\Delta y,t)}{2\Delta y}
\end{aligned} \tag{35}$$

The derivation for a CFL condition of linear heat equation expanded to 2D but keeping the y constant to derive the update of ψ using ψ_{xx} is shown below. So, obtaining the CFL condition for

this central difference scheme,

$$\begin{aligned}\psi(x,y,t+\Delta t) &= \psi(x,y,t) + \Delta t \frac{\psi(x+\Delta x,y,t) - \psi(x,y,t)}{x} \\ \psi(x,y,t+\Delta t) &= \psi(x,y,t) + \Delta t \frac{\psi(x,y,t) - \psi(x-\Delta x,y,t)}{\Delta x}\end{aligned} \tag{36}$$

$$\begin{aligned}\psi(x,y,t+\Delta t) &= \psi(x,y,t) + b\Delta t \frac{\psi(x+\Delta x,y,t) - 2\psi(x,y,t) + \psi(x-\Delta x,y,t)}{\Delta x^2} \\ \psi(\omega,y,t+\Delta t) &= \psi(\omega,y,t) + b\Delta t \frac{e^{j\omega\Delta x}\psi(\omega,y,t) - 2\psi(\omega,y,t) + e^{-j\omega\Delta x}\psi(\omega,y,t)}{\Delta x^2} \\ &= \psi(\omega,y,t)\left(1 + b\Delta t \frac{e^{j\omega\Delta x} - 2 + e^{-j\omega\Delta x}}{\Delta x^2}\right)\end{aligned} \tag{37}$$

Taking the part inside brackets which must be less than 1 for stability,

$$\begin{aligned}1 &\geq \left\| 1 + b\Delta t \frac{e^{j\omega\Delta x} - 2 + e^{-j\omega\Delta x}}{\Delta x^2} \right\| \\ &= \left\| 1 + b\Delta t \frac{2(cos(\omega\Delta x) - 1)}{\Delta x^2} \right\| \\ -1 &\leq 1 + b\Delta t \frac{2(cos(\omega\Delta x) - 1)}{\Delta x^2} \leq 1\end{aligned} \tag{38}$$

Second inequality is always true given that $b \geq 0$. Since $min(\,2(cos(\omega\Delta x)-1)\,) = -2$, equating the first inequality,

$$\begin{aligned}-1 &\leq 1 + b\Delta t \frac{2(cos(\omega\Delta x) - 1)}{\Delta x^2} \\ \therefore b\Delta t &\leq \frac{\Delta x^2}{2}\end{aligned} \tag{39}$$

The equation ,

$$\psi_t = \frac{\psi_x^2\psi_{yy} - 2\psi_x\psi_y\psi_{xy} + \psi_y^2\psi_{xx}}{\psi_x^2 + \psi_y^2}$$

resembles the geometric heat equation which has the same CFL condition as the linear heat equation. So, the CFL condition for this part of the gradient descent algorithm,

$$b\Delta t \leq \frac{\Delta x^2}{2} \tag{40}$$

The discretization of this part of the equation with edge extension is shown in the Code snippet Listing 2

Listing 2: MATLAB Code Central Difference Discretization

```
% Compute the level set formulation of kappa * Normal using a central
% difference scheme and edge preservation
function kn = kappaNormal(psi)
% Preserve the edges so identical values at extension of image edge
    psix = ( psi([2:end end],:) - psi([1 1:end-1],:) )./2;
    psiy = ( psi(:,[2:end end]) - psi(:,[1 ,1:end-1]) )./2;
```

```
    psixy = ( psix(:,[2:end end])-psix(:,[1   ,1:end-1]) )./2;
    psixx = ( psi([2:end end],:)- 2.*psi + psi([1  1:end-1],:) );
    psiyy = ( psi(:,[2:end end])- 2.*psi + psi(:,[1   ,1:end-1]) );

% Level set formulation of kappa*normal
    kn = (psixx.*(psiy.^2) - 2.*psix.*psiy.*psixy + psiyy.*(psix.^2))...
        ./(max(eps, ((psix.^2)+(psiy.^2))));
end
```

4.3 Upwind Entropy: $\|\nabla\psi\|$

The discretization of the level set evolution of equation 31 uses an upwind entropy scheme as shown below depending on the sign of the coefficient, β of the term $\|\nabla\psi\|$,

$$\begin{aligned}\beta > 0 \Rightarrow \|\nabla\psi\| &= \sqrt{max^2(D_x^+,0) + min^2(D_x^-,0) + max^2(D_y^+,0) + min^2(D_y^-,0)} \\ \beta < 0 \Rightarrow \|\nabla\psi\| &= \sqrt{min^2(D_x^+,0) + max^2(D_x^-,0) + min^2(D_y^+,0) + max^2(D_y^-,0)}\end{aligned} \tag{41}$$

The value of β would be positive for the part inside the contour R_{in} and negative for the part outside the contour R_{out}. D_x^+ and D_y^+ refers to forward difference while D_x^- and D_y^- refers to backward difference as follows,

$$\begin{aligned}D_x^+ &= \frac{\psi(x+\Delta x,y,t) - \psi(x,y,t)}{\Delta x} \\ D_x^- &= \frac{\psi(x,y,t) - \psi(x-\Delta x,y,t)}{\Delta x} \\ D_y^+ &= \frac{\psi(x,y+\Delta y,t) - \psi(x,y,t)}{\Delta y} \\ D_y^- &= \frac{\psi(x,y,t) - \psi(x,y-\Delta y,t)}{\Delta y}\end{aligned} \tag{42}$$

For 1D transport equation with a forward differencing scheme,

$$\begin{aligned}\psi(x,y,t+\Delta t) &= \psi(x,y,t) + b\Delta t\frac{\psi(x+\Delta x,y,t) - \psi(x,y,t)}{\Delta x} \\ \psi(\omega,y,t+\Delta t) &= \psi(\omega,y,t) + b\Delta t\frac{e^{j\omega\Delta x}\psi(\omega,y,t) - \psi(\omega,y,t)}{\Delta x} \\ &= \psi(\omega,y,t)\left(1 + b\Delta t\frac{e^{j\omega\Delta x}-1}{\Delta x}\right)\end{aligned} \tag{43}$$

Taking the magnitude of the part inside brackets and expanding, which must be less than 1 for stability,

$$
\begin{aligned}
1 &\geq \left\| 1 + b\Delta t \frac{e^{j\omega\Delta x} - 1}{\Delta x} \right\| \\
1 &\geq \left\| 1 + b\Delta t \frac{(cos(\omega\Delta x) + jsin(\omega\Delta x) - 1)}{\Delta x} \right\| \\
1 &\geq \left(1 + b\Delta t \frac{(cos(\omega\Delta x) - 1)}{\Delta x} \right)^2 \left(b\Delta t \frac{(sin(\omega\Delta x))}{\Delta x} \right)^2 \\
0 &\geq b\Delta t \frac{(cos(\omega\Delta x) - 1)}{\Delta x} + (b\Delta t)^2 \frac{(1 - cos(\omega\Delta x))}{\Delta x^2} \\
0 &\geq b(cos(\omega\Delta x) - 1) + (b^2\Delta t) \frac{(1 - cos(\omega\Delta x))}{\Delta x}
\end{aligned} \tag{44}
$$

When $b \geq 0$,

$$0 \geq (cos(\omega\Delta x) - 1) + (b\Delta t) \frac{(1 - cos(\omega\Delta x))}{\Delta x}$$

The first term is always negative as required. To check for the second term, $max(\frac{(1-cos(\omega\Delta x))}{\Delta x}) = 2$. But when it is 2, the first term is -2, resulting in,

$$2 \geq b\Delta t \frac{2}{\Delta x} \Rightarrow \Delta x \geq b\Delta t$$

When $b < 0$,

$$0 \leq (cos(\omega\Delta x) - 1) + (b\Delta t) \frac{(1 - cos(\omega\Delta x))}{\Delta x}$$

both terms must be positive when in fact both are negative. So it is never satisfied.
Similarly, for the backward difference CFL condition and combining the two the following is obtained,

$$\Delta x \geq |b|\Delta t \tag{45}$$

In our particular expression of two dimensions this condition has a factor of $\frac{1}{\sqrt{2}}$ applied to it resulting in the following final CFL condition,

$$\frac{\Delta x}{\sqrt{2}} \geq |b|\Delta t \tag{46}$$

The discretization of this part of the equation with edge extension is shown in the Code snippet Listing 3 below,

Listing 3: MATLAB Code Upwind Entropy Discretization

```
% Compute the Level Set formulation of normal using upwind entropy scheme
% and edge preservation
function norm = normal(psi, sign)
% Preserve the edges so identical values at extension of image edge
    DxF = psi([2:end end],:) - psi;
    DxB = psi - psi([1 1:end-1],:);
    DyF = psi(:,[2:end end]) - psi;
    DyB = psi - psi(:,[1 ,1:end-1]);

% Level set formulation of normal
```

```
    if sign == 1
        norm = ((max(DxF,0)).^2 + (min(DxB,0)).^2 + (max(DyF,0)).^2 ...
            + (min(DyB,0)).^2).^(1/2);
    elseif sign == -1
        norm = ((min(DxF,0)).^2 + (max(DxB,0)).^2 + (min(DyF,0)).^2 ...
            + (max(DyB,0)).^2).^(1/2);
    end
end
```

4.4 Overall Courant Friedrichs Lewy condition

The overall CFL condition is the minimum of the CFL condition of the two segments shown above.

$$\begin{aligned} |b_1|\Delta t &\leq \frac{\Delta x}{\sqrt{2}} \\ b_2 \Delta t &\leq \frac{\Delta x^2}{2} \end{aligned} \tag{47}$$

The overall resulting update equation,

$$\psi(x,y,t+1) = \psi(x,y,t) + \Delta t \left((1-\lambda)\left((I-u)^2 - (I-v)^2\right)\|\nabla\psi\| + \lambda\nabla.\left(\frac{\nabla\psi}{\|\nabla\psi\|}\right)\|\nabla\psi\| \right) \tag{48}$$

The value of b_1 is λ. value of b_2 is simply the multiplication of $(1-\lambda)$ with the maximum of the square of the image intensity at each pixel because the minimum value of the normalized image intensity is zero. The overall CFL condition would be the more restrictive of the time step derived from these two equations. The code for using this CFL condition is shown in the code snippet Listing 4.

Listing 4: MATLAB Code Time Step using CFL Condition

```
%       Obtain tStep from CFL condition
tStep = min( 0.5/(max(I(:).^2)*(1-lambda)), 1/(lambda*sqrt(2)) );
```

5 Results

The image read into MATLAB was converted into a gray-scale, normalized and subsequently noise was added as required for testing. Two data sets used is distributed as part of the TOSCA (Toolbox for Surface Comparison and Analysis) [1, 2] [1]. A brief description from the dataset website is provided below:

- Two-dimensional articulated shapes (silhouettes). The data set contains 35 shapes: 5x2 scissors, 5x3 pliers, 5 pincers, 5 knives.
- Two-dimensional articulated shapes (silhouettes). The data set contains 15 shapes: 5 humans, 5 horses and 5 centaurs.

[1] http://tosca.cs.technion.ac.il/book/resources_data.html

The accuracy of the resulting segmentation on the noisy image after running the algorithm was compared with the ground truth by comparing each pixel to the a pixel on the segmented version of noiseless original image. This segmented version of the original image was formed by setting a threshold of 0.5 on the original gray-scale, normalized image. This brute force segmentation was compared visually with the original image and appeared to be a reasonable segmentation of the original binary images. The resulting accuracy which is the ratio of pixels that matched is shown in Table 1. The algorithm appears to work well for all noise types.

Dataset	**Noiseless**	**Gaussian**	**Salt & Pepper**	**Poisson**	**Speckle**
Myth	0.9977	0.9973	0.9970	0.9977	0.9973
Tools	0.9988	0.9985	0.9985	0.9987	0.9802

Table 1: Accuracy on the two datasets using 1000 iterations

Figure 1 shows the accuracy reached for different values of λ using salt and pepper noise. Although for other noise types, a lower value works perfectly fine, it is due to this curve, that the value of λ was chosen to be 0.25 for all noise types. The accuracy in Table 1 and all following graphs are derived using this same value of λ.

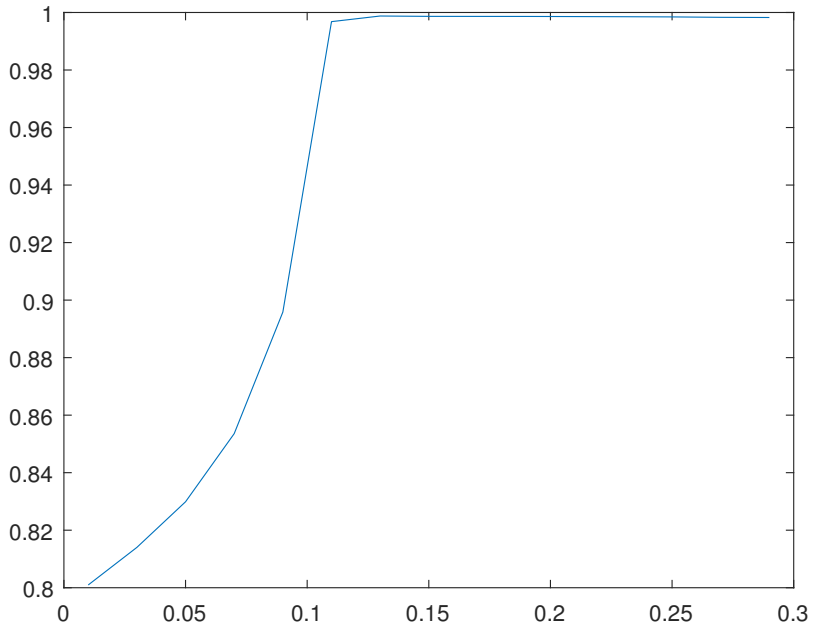
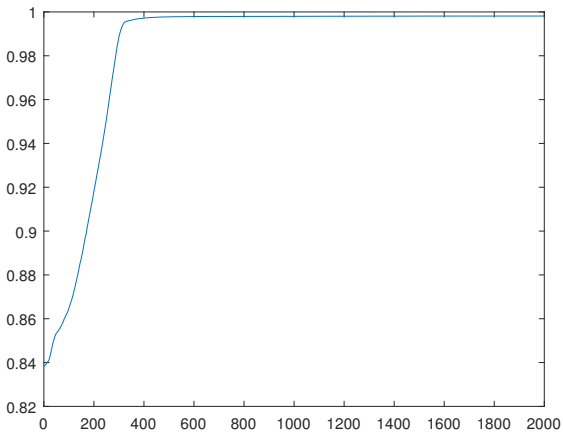
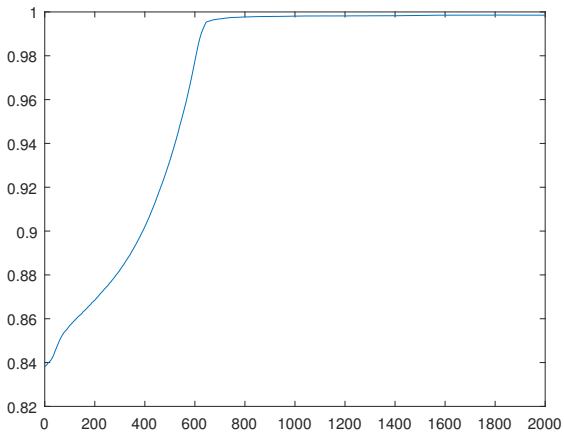


Figure 1: Accuracy vs λ

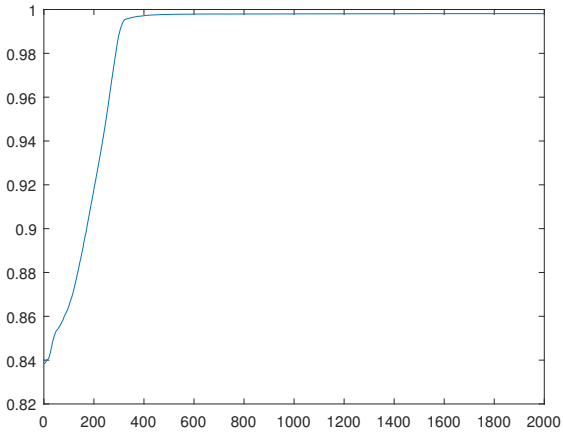
Figure 2 shows the way accuracy evolves over iterations for different types of noise for a particular image. It shows the quicker learning rate for a noiseless image and the slower rate for Gaussian, and Speckle in particular. The reason for the apparent quicker learning rate for Salt and Pepper noise despite it being the most problematic is the value of λ was tuned to fit the Salt and Pepper noise and the same value used over all different noise types.
Salt and pepper noise was the most challenging noise for the algorithm. The accuracy vs iteration graphs in Figure 3a and 3b shows the accuracy of the implementation in segmenting a scissor image



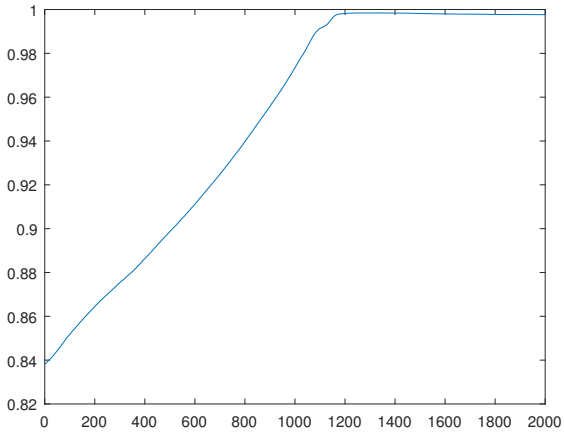
(a) Noiseless



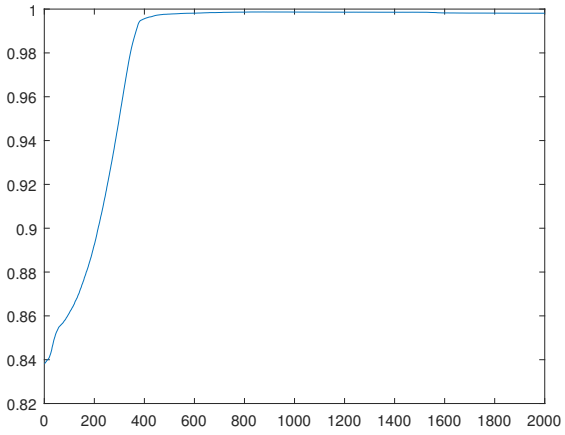
(b) Gaussian noise



(c) Poisson noise



(d) Speckle noise

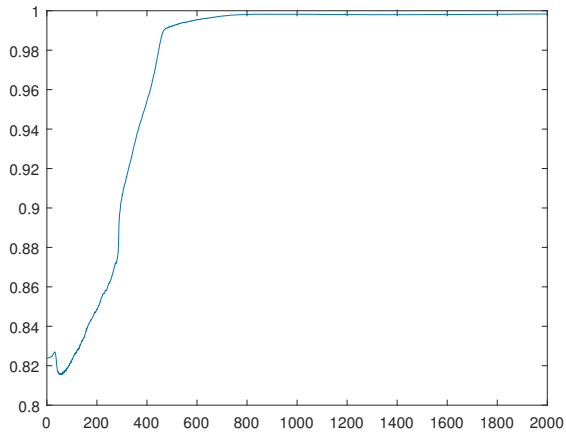


(e) Salt and Pepper noise

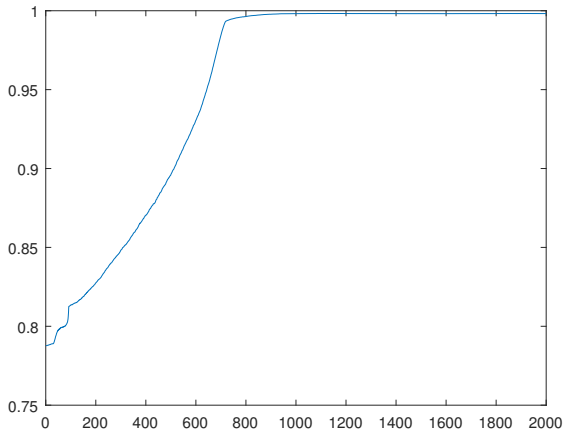
Figure 2: Accuracy vs iteration for different noise types

successfully. The convergence to the peak value is faster for a single contour as opposed to multiple contour but the peak value reached by multiple contour was slightly higher (< 0.001) at the stopping

value. The slight decreases in the accuracy is due to the fact that the accuracy metric used here does not correspond to the energy that is being minimized.



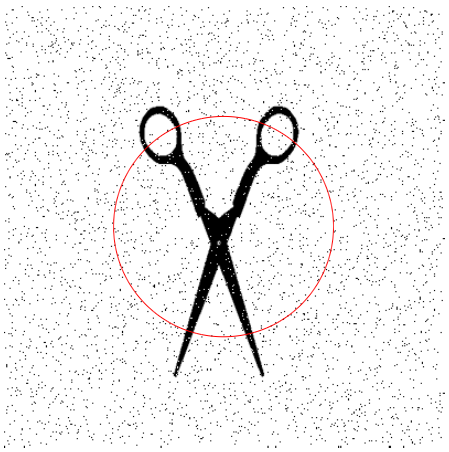
(a) Initialized ψ using 1 per row



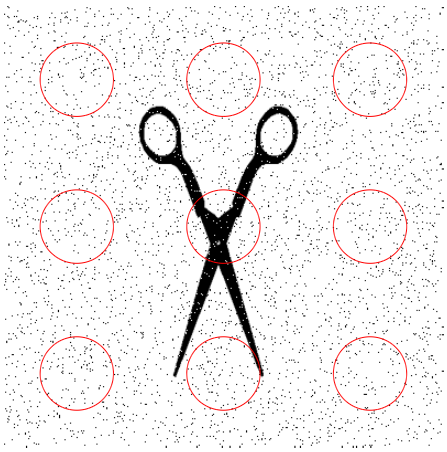
(b) Initialized ψ using 3 per row

Figure 3: Accuracy vs iteration for salt and pepper noise using different number of initial contours

Figure 4a and 4b shows the original image with salt and pepper noise added with different numbers of initial contours. The generated initial contour is shown in red. The circular rings at different arbitrary values of the level set function are shown in Figure 5. The code for generating these initial level sets ψ is shown in Listing 1



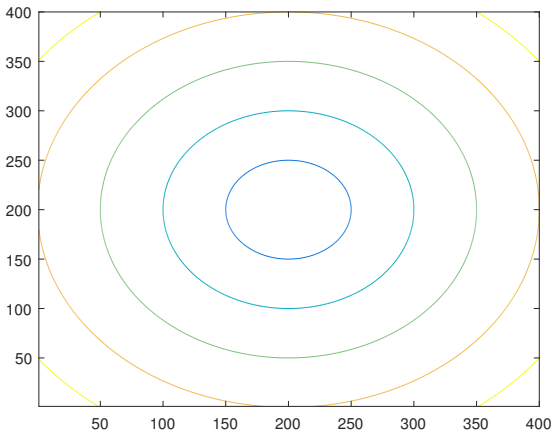
(a) Initialized ψ using 1 per row



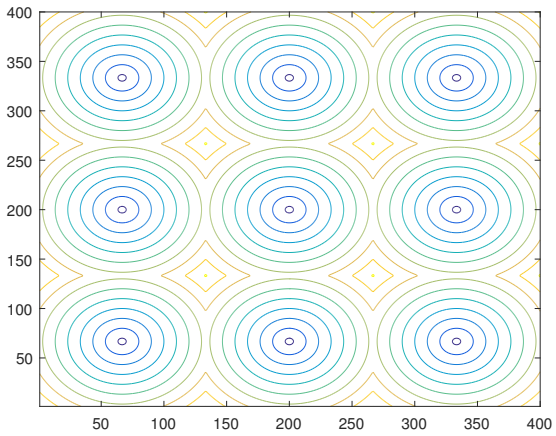
(b) Initialized ψ using 3 per row

Figure 4: Image and zero level set in red with salt and pepper noise

Figure 6 refers to the binary segmented image. Figure 6b is the initially segmented image while Figure 6c is the final segmented image.
A final comparison shows a real non-binary image with salt and pepper added with the obtained binary segmentation in Figure 7. The red outline in the noise image is the zero level set obtained



(a) Initialized ψ using 1 per row



(b) Initialized ψ using 3 per row

Figure 5: Initial Level Sets

(a) Original Image

(b) Initial Segmentation

(c) Final Segmentation from image with Salt and Pepper Noise

Figure 6: Segmented images

in the image of the house. The value of λ was the same for this image as for the other parts, 0.25, but the number of iterations was 10000.

6 Conclusion

This project showed how PDEs are actually implemented and discretized for using in a real world problem, image segmentation. The overall implementation showed the particular significance of the CFL condition and how choosing different parameters affect the speed of convergence with the desired segment and the accuracy at that point.
The main issue faced was in handling the salt and pepper noise. The contour would evolve perfectly fine for the other noise types. Once the correct CFL condition was used, the value of λ, the weight to term κN, could be pushed up without blowing up the evolution.
With more time, I would have liked to compare this particular image segmentation algorithm to others like the Geodesic Active Contour. The accuracy could have been tested on a larger dataset than the one used here which had 50 binary images.

(a) House with salt and pepper noise with zero level set in red

(b) Final binary segmented house

Figure 7: Segmented house image

References

[1] A. M. Bronstein, M. M. Bronstein, R. Kimmel, *Numerical geometry of non-rigid shapes*, Springer, 2008.

[2] A. M. Bronstein, M. M. Bronstein, A. M. Bruckstein, R. Kimmel, *Analysis of two- dimensional non-rigid shapes*, Intl. J. Computer Vision (IJCV), Vol. 78/1, pp. 67-88, June 2008.

[3] Chan, Tony F and Vese, Luminita A, *Active contours without edges*, IEEE Transactions on image processing, Vol.10, No. 2, pp, 266-277, IEEE, 2001

[4] Kass, Michael, Andrew Witkin, and Demetri Terzopoulos. *Snakes: Active contour models.* International journal of computer vision 1, no. 4 (1988): 321-331.

Listing 5: Entire MATLAB Code for generating accuracy vs noise graphs

```
function project_main()
    close all;
    clear all;
    % Loaded image resized into a square pixels
    n = 400;

    folder = 'tools/';

    direc = dir(strcat(folder,'*.bmp'));

    % Set time parameters
    % tStep = 0.1;
    iter = 2000;
```

```
% Set scaling parameters for energy terms
lambda= 0.25;

% nContours: number of contours in each row column
nContours = 1;
radius = n/nContours/4;

% Noise trigger: if 'true' noise added
% Noise types 1:gaussion, 2:salt & pepper, 3:localvar, 4:poisson, 5:speckle
noiseTrigger = true;
noiseAll = {'gaussian'; 'salt & pepper'; 'poisson'; 'speckle'};
noiseType = noiseAll{2};

% Create two matrices where each point has the value of the coordinate
% split into corresponding positions in X and Y matrices
X = zeros(n,n);
Y = zeros(n,n);
for i = 1 : n
    for j = 1 : n
        X(i,j) = j;
        Y(i,j) = i;
    end
end

% Generate contours with equally spaced centers
k=1;
psiInit(1,:,:)=zeros(n,n);
for i = 1:nContours
    for j = 1:nContours
        f1=1+(i-1)*2;
        f2=1+(j-1)*2;
        centerX = n/nContours/2*f1;
        centerY = n/nContours/2*f2;
        psiInit(k,:,:) = sqrt((X-centerX).^2 + (Y-centerY).^2)-radius;
        k=k+1;
    end
end


for cur_noise=1:length(noiseAll)+1
    if cur_noise > length(noiseAll)
        noiseTrigger=false;
    else
        noiseType=noiseAll{cur_noise};
    end
```

```
% Load image: Change filename with director to image file
filename = direc(1).name;
image = imread(strcat(folder,filename));

% Convert to greyscale if RGB
if size(image,3)==3
    image = rgb2gray(image);
end


% Resize image and nomalize
image = imresize(image, [n, n]);
image = double(image);
image = image./max(image(:));

% Average image intensity of max and min
% image_avg = sum(image(:))/(n*n);
image_avg = (max(image(:))+min(image(:)))/2;

% Pixels having intensity higher than average
radius_dyn = sum(image(:)>image_avg);

% radius: radius of each contour
% radius=radius_dyn;

cleanI = image;


% Add noise if specified and normalize as chosen in specifications
if noiseTrigger
    image = imnoise(image, noiseType);
end


% Find the minimum of each point to get overall initial contour
psi = squeeze(min(psiInit,[],1));

% Compute segmentation with Chan_Vese
I = image;


for i = 1:iter
%     Average intensity inside contour
    u = sum(sum(I.*(psi < 0)))/sum(psi(:) < 0);

%     Average intensity outside contour
```

```
            v = sum(sum(I.*(psi > 0)))/sum(psi(:) > 0);

        %       Obtain tStep from CFL condition
            tStep = min( 0.5/(max(I(:).^2)*(1-lambda)), 1/(lambda*sqrt(2)) );

        %       Determine the psi at the next time step using the update equation
            psi = psi +  tStep.*((1-lambda)*((I-u).^2.*normal(psi,1)...
                - (I-v).^2.*normal(psi,-1)) + lambda.*kappaNormal(psi)) ;

            acc(i) = sum(sum((psi>0) == (cleanI>0.5)))/(n*n);
        end

        figure(cur_noise);
        plot(1:iter,acc);
        avg_acc(cur_noise) = sum(acc)/length(acc);
    end

    disp(avg_acc)
end

% Compute the Level Set formulation of normal using upwind entropy scheme
% and edge preservation
function norm = normal(psi, sign)
% Preserve the edges so identical values at extension of image edge
    DxF = psi([2:end end],:)-psi;
    DxB = psi - psi([1 1:end-1],:);
    DyF = psi(:,[2:end end])-psi;
    DyB = psi - psi(:,[1 ,1:end-1]);

% Level set formulation of normal
    if sign == 1
        norm = ((max(DxF,0)).^2 + (min(DxB,0)).^2 + (max(DyF,0)).^2 ...
            + (min(DyB,0)).^2).^(1/2);
    elseif sign == -1
        norm = ((min(DxF,0)).^2 + (max(DxB,0)).^2 + (min(DyF,0)).^2 ...
            + (max(DyB,0)).^2).^(1/2);
    end
end

% Compute the level set formulation of kappa * Normal using a central
% difference scheme and edge preservation
function kn = kappaNormal(psi)
% Preserve the edges so identical values at extension of image edge
    psix = ( psi([2:end end],:)-psi([1 1:end-1],:) )./2;
    psiy = ( psi(:,[2:end end])-psi(:,[1 ,1:end-1]) )./2;
    psixy = ( psix(:,[2:end end])-psix(:,[1 ,1:end-1]) )./2;
    psixx = ( psi([2:end end],:)- 2.*psi + psi([1 1:end-1],:) );
```

```
    psiyy = ( psi(:,[2:end end])- 2.*psi + psi(:,[1   ,1:end-1]) );

% Level set formulation of kappa*normal
    kn = (psixx.*(psiy.^2) - 2.*psix.*psiy.*psixy + psiyy.*(psix.^2))...
        ./(max(eps, ((psix.^2)+(psiy.^2))));
end
```