Image Inpainting with TV Regularization and the Bertalmio-Sapiro-Caselles-Ballester Algorithm

APMA 935 Final Project

Natalia Iwanski

April 18, 2014

1 Abstract

This project studies the process of image inpainting using variational techniques. Generally this involves a PDE model which can propagate information from known regions in an image, to unknown or damaged regions. Two different techniques will be examined, total variation (TV) inpainting and a technique developed by Bertalmio, Sapiro, Caselles and Ballester, referred to as BSCB inpainting. TV inpainting evolves unknown regions using curvature motion, combined with known values on the boundary. Since curvature motion propagates information in the normal direction of a contour, this technique works well for small regions and is able to continue the general gray level of an area, but it is unable to preserve the direction of contour lines in an image. BSCB inpainting on the other hand propagates information into an unknown region by following the direction tangent to the contour lines at the boundary, which are referred to as isophote directions. This enables the method to carry structural information about an image into an unknown region. However, it is unable to replicate texture, and becomes inefficient for large inpainting regions. Both methods are tested on several examples to compare their performance, and BSCB inpainting is applied to colour images as well. Results show that variational techniques are very useful tools in tackling the problem of image inpainting and are successful in repairing damage and removing objects in a wide variety of images.

2 Introduction

For many applications, it is useful to be able to automate the process of image inpainting, especially when restoring damaged paintings or photographs, or removing selected objects from images. Variational techniques provide a very good way to achieve this using only the initial image and a binary mask of the unknown region. They are able to fill in multiple unknown regions at once with varying structures and backgrounds.

Although the problem of image inpainting is different from the one of denoising, it is possible to adapt denoising techniques to be able to fill in missing regions of a picture. One such method is TV regularization, which uses the TV norm in minimizing the energy of an image.

Another technique is given by Bertalmio et al. in [1], (referred to as BSCB inpainting) which approximates the direction of contour lines arriving at the boundary of the inpainting region, also known as isophotes. These isophotes are continued into the inpainting region, and completed inside. This process can be modelled as a time-dependent PDE where the isophotes in the inpainting region are updated at each time step.

TV inpainting will be examined first, discussing how TV denoising can be adapted to fill in missing regions. Results will be shown for several different images, comparing successful and unsuccessful cases. BSCB inpainting will be examined afterwards, including a discussion of the underlying concepts of the method. It will be applied to several black and white images, and results will be compared against TV denoising, and will also be extended to several colour images.

3 TV Inpainting

TV inpainting comes from the TV denoising model first proposed by Rudin, Osher, and Fatemi in [6]. When denoising an image, $I(x, y)$, we want to be able to smooth regions with a large gradient, while keeping the image faithful to the original. This motivates the use of an energy term of the form:

$$E(I, I_0) = \int_\Omega |\nabla I| dx + \lambda \int_\Omega |I - I_0|^2 \tag{1}$$

where I_0 denotes the initial, noisy image and I represents the denoised image. Ω denotes the region the image is defined on and $\lambda \geq 0$ is a parameter which controls how true to the original image we would like to be. Hence the first term specifies the amount of noise in the image, and the second is the fidelity term, ensuring we remain close to the features of the original. Minimizing this energy then gives us the desired denoised image.

The first term in (1) is the TV norm, which for smooth images is defined as

$$||I||_{TV} = \int |\nabla I| dx$$

The TV norm is finite for images with line discontinuities and hence it is able to penalize the magnitude of the gradient without smoothing edges. Expression (1) can be minimized by writing out the corresponding Euler-Lagrange equations, and formulating them as the steady state of a time-dependent PDE, given by

$$\begin{aligned} \frac{\partial I}{\partial t} &= |\nabla I| \left(\text{div} \, \frac{\nabla I}{|\nabla I|} - 2\lambda(I - I_0) \right) \\ \frac{\partial I}{\partial n} &= 0 \end{aligned} \tag{2}$$

To adapt this model to image inpainting, we want to apply (2) to the missing regions in an image, denoted by D. Hence the fidelity term in (1) is no longer applicable in D, because we have no information about I_0. Setting $\lambda = 0$, when $(x, y) \in D$, we obtain the following energy term

$$E(I, I_0) = \int_\Omega |\nabla I| dx + \lambda \int_{\Omega \setminus D} |I - I_0|^2 \tag{3}$$

Furthermore, if we are not interested in denoising the entire image, then the fidelity term can be dropped altogether, and $E(I, I_0)$ can be computed only in D. Consequently, we obtain the following PDE to solve to steady state in D

$$\frac{\partial I}{\partial t} = |\nabla I| \left(\text{div} \, \frac{\nabla I}{|\nabla I|} \right) \tag{4}$$

with boundary conditions being enforced by setting any points outside of D equal to $I(x, y) \in \Omega \setminus D$ (Dirichlet boundary conditions).

This PDE describes the curvature motion of the contours of I. Each level set of I evolves in the normal direction with a speed proportional to its curvature. It also satisfies the maximum principle, which implies that no new artificial structures are introduced while evolving to steady state [9]. Hence using TV regularization for inpainting will propagate features of known regions in a direction normal to their contours, and will also smooth unknown regions, preserving edges.

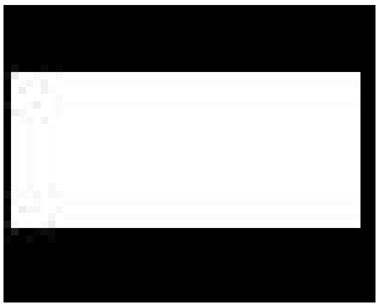
3.1 Numerical Implementation

Equations (2) and (4) can be solved numerically using gradient descent. A fixed point finite difference scheme can be applied with a semi-implicit iterative Gauss-Seidel method. Details can be found in [8]. If the PDE is only being solved in the region $D \subset \Omega$ then the boundary conditions are simply Dirichlet boundary conditions, with any points (x, y) outside of D equal to $I(x, y)$. If the PDE is being solved on the entire region Ω, implying that both denoising and inpainting are being performed, then Neumann boundary conditions must be specified, which can be found in [8]. The maximum number of iterations made by the scheme can be set as an input from the user, or as a tolerance level. In the implementation for this project, a tolerance level was set on the energy. After each iteration the energy of the image was computed using (1) or (3), and a tolerance was set on the difference in energy between two iterations.

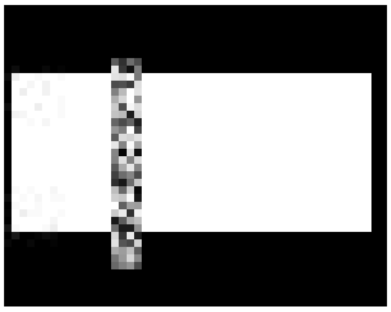
3.2 Examples

TV inpainting was run on several examples shown below. For all examples a step size of $\Delta t = 0.1$ was used. Initial masks were set to random values between 0 and 1, as in many cases this reduces the number of iterations for the algorithm [4]. However, in some instances, the colour of the mask can be strategically chosen based on the particular image being inpainted (Figure 3). The energy associated with each image is also plotted against iterations to check if it is progressing towards a minimum.

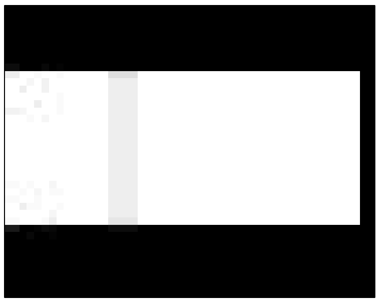
Figure 1 is a simple black and white image of a white bar on a black background with a mask splitting the bar in the middle. The final inpainted



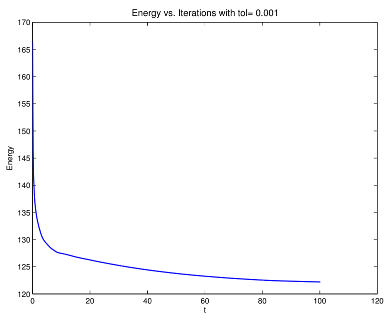
(a)



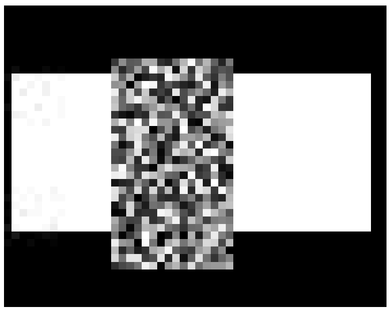
(b)



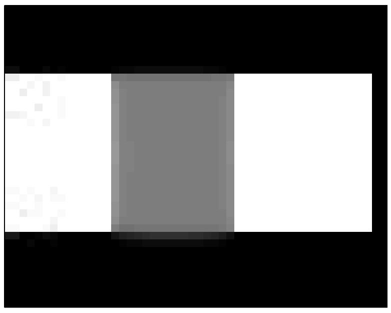
(c)



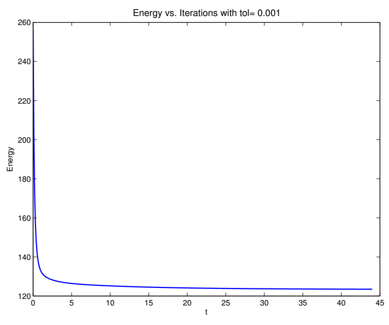
(d)



(e)



(f)

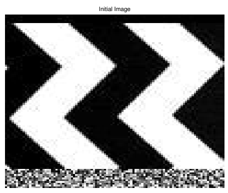


(g)

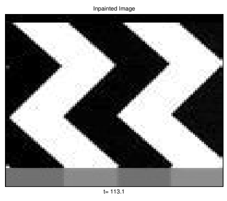
Figure 1: Results for inpainting (a) with a thin mask and thick mask, (b) and (e). The original image is 50 × 40 pixels in size. The tolerance level was set to 1e-3, and the final time for mask (b) was t=100.3 with a step size of $\Delta t = 0.1$. The final time for mask (e) was t=44.1. We see that TV inpainting performs well on the thin mask, but is unable to recover the original image with the wider mask.

result depends on the width of the region to fill in. For a thin mask the algorithm is able to propagate the surrounding white bar into the unknown region, preserving the edges of the bar. For the thicker mask however, we see that it brings in the darker region surrounding the bar. Due to the fact that the algorithm evolves according to curvature motion in the normal direction of the image contours, it does not specify which direction to continue an edge, and hence the white bar is separated into two.

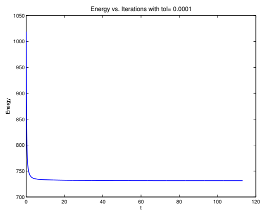
Figure 2 demonstrates the fact that TV inpainting is unable to preserve the angle of contour lines arriving at the boundary of the inpainting region. We see that the angled regions in the figure are straightened by the algorithm.



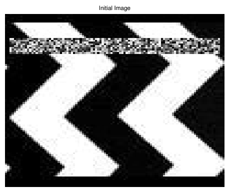
(a)



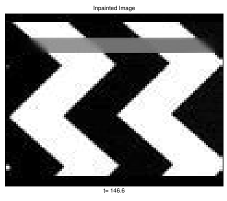
(b)



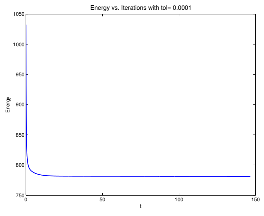
(c)



(d)



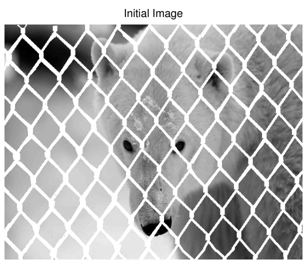
(e)



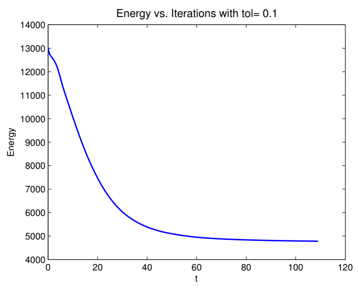
(f)

Figure 2: Results applied to 100×100 pixel image using two different masks. In both we see that TV inpainting cannot preserve the angles in the image.

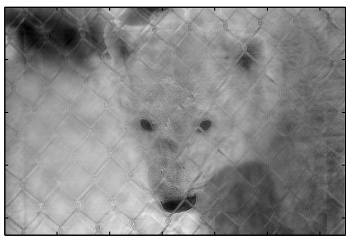
Figure 3 shows the result of removing a fence from an image covering the polar bear Storm. The algorithm performs very well, since it is very effective on thin regions. Comparing this result to the one obtained with Fourier methods, we see that TV inpainting is able to do a much better job. However, the gradient descent algorithm is quite slow on images and masks of this size (400×600 pixels). For this image in particular the final time was t=109.1 with a step size of 0.1 and tolerance level of 0.1. The process took about 15 minutes running on an Intel(R) PC (8G RAM, 2.80 GHz). Thus to use this on any larger regions or larger images, one would need to apply a faster method to solve (2) and (4) such as convex optimization methods.



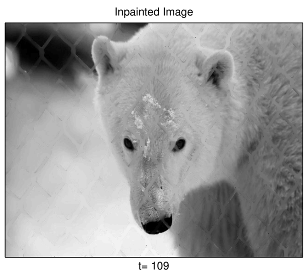
(a)



(b)



(c)



(d)

Figure 3: Results for removing the fence from Storm (648 × 432 pixels). The algorithm took approximately 15 minutes to run with a tolerance level of 0.1. Even better results can be obtained with a lower tolerance. Figure (c) was obtained using Fourier methods, and we can see that TV inpainting easily outperforms the Fourier method in this case.

4 BSCB Inpainting

TV inpainting is unable to propagate structural information into an unknown region because of the fact that it moves in a direction normal to the isophotes of an image. Bertalmio et al. address this issue in [1] by implementing a method which propagates information along the direction of isophotes so contour lines are continued into an unknown region. Since level sets of an image are orthogonal to the gradient of the image, these isophote directions are denoted by $\nabla^{\perp} I$. Figure 4 shows how the directions ∇I and $\nabla^{\perp}$ relate to the isophotes of I.

If we denote the inpainting region by D and its boundary as ∂D, then the algorithm is designed to carry both the isophote directions and the gray values along ∂D into D. It evolves an image according to the PDE:

$$\frac{\partial I}{\partial t} = \nabla(\Delta I) \cdot \nabla^{\perp} I \tag{5}$$

where the the boundary conditions are drawn from outside of D in the known region of the image. In this case the information being propagated

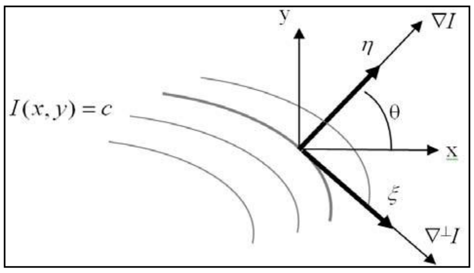


Figure 4: Image isophotes, $I(x, y) = c$, and their normal and tangent vectors, denoted by ∇I and $\nabla^{\perp}$ respectively. Figure obtained from [5].

along isophote directions is the"smoothness" of the surrounding image, represented by the Laplacian.

The steady state is achieved when $\frac{\partial I}{\partial t} = 0$, which means that the Laplacian of I is constant in the isophote direction, giving a smooth continuation of the Laplacian in D [3]. Using the Laplacian as a representation of smoothness ensures the continuity of $I(x, y, t)$ for $t = 1, ..T$ and its derivatives.

So the model first estimates the variation of smoothness in an image, and projects this variation onto the isophote directions. The projection is then used to update the values of I inside the inpainting region D. As contour lines are extended into D, a few iterations of diffusion are regularly applied to ensure that the contours do not begin crossing each other. Bertalmio et al. use anisotropic diffusion which is exactly the diffusion process used in TV inpainting in equation (4). The diffusion process is only applied in the region D, with boundary values being drawn from outside of D using the known values of I. The authors refer to this region as $D^{\epsilon} \subset \Omega$.

The goal is to be able to connect level sets corresponding to the same gray values together. As a result it performs best when D is a proper subset of Ω, since the boundary values can be drawn from $\Omega \setminus D$. This case is examined in the numerical implementation and examples shown, in sections 4.1-4.3.

4.1 Numerical Implementation

Bertalmio et al. give a step by step procedure for implementing the inpainting algorithm. They first let $I_0(i, j)$ denote the 2D discrete gray level image to be inpainted, with missing regions denoted by the set $D \subset \Omega$. The algorithm then constructs a family of images $I(i, j, n)$, such that as $n \to \infty$, $I(i, j, n) = I_R(i, j)$ where R denotes the restored image. Updates at each time step are performed on all pixels $I(i, j) \in D$, and are given by

$$I^{n+1}(i, j) = I^n(i, j) + \Delta t I_t^n(i, j) \qquad (6)$$

where $I_t^n(i,j)$ denotes the projection of the Laplacian onto the isophote direction at (i,j). To compute the update we first need to find the isophote direction at (i,j), given by

$$\frac{\overrightarrow{N(i,j,n)}}{|\overrightarrow{N(i,j,n)}|} := \frac{(-I_y^n(i,j), I_x^n(i,j))}{\sqrt{(I_x^n(i,j))^2 + (I_y^n(i,j))^2 + \epsilon}} \tag{7}$$

where I_x and I_y can be approximated with finite difference formulas. To ensure that $\overrightarrow{N}$ is being computed using points on ∂D or outside D, pixels surrounding (i,j) are scanned to find the direction of the boundary. If there are un-updated values to the left of (i,j) for example, then values from the right will be used to find $\overrightarrow{N}$ with a one-sided finite difference formula (Figure 5). Hence a combination of two point forward and backward differences are used, or 3-point centred differences if the points surrounding $I(i,j)$ have all been updated at least once.

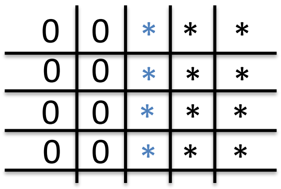


Figure 5: Pixel values for an an image $I(x,y)$ along the boundary of the inpainting region, ∂D. The blue stars represent ∂D, the black stars represent known values of I outside of D and the zeros represent pixels in the inpainting region that have not been updated by the algorithm yet. In this case pixel $(i,j) \in D$ would be updated using a forward difference formula drawing values from the right.

Once the isophote direction is known we can compute the difference in smoothness at $I(i,j)$ given by

$$\overrightarrow{\delta L^n}(i,j) := (L^n(i+1,j) - L^n(i-1,j), L^n(i,j+1) - L^n(i,j-1)) \tag{8}$$

where $L^n(i,j) = I^n(i-1,j) + I^n(i+1,j) + I^n(i,j-1) + I^n(i,j+1) - 4I(i,j)$, and a uniform spatial grid is assumed with $h = 1$. The projection of δL^n onto the isophote direction is then given by

$$\beta^n(i,j) := \delta L^n(i,j) \cdot \frac{\overrightarrow{N(i,j,n)}}{|\overrightarrow{N(i,j,n)}|}$$

For numerical stability, β^n is multiplied by $|\nabla I|$ (or $|\nabla^{\perp} I|$, since they are equal). It is important to note that $|\nabla I|$ is not computed with centred differences because it will lead to problems with numerical stability around steep or flat gradients. Instead Bertalmio et al. use slope limiters which give

$$|\nabla I^n(i,j)| = \begin{cases} \sqrt{(I^n_{xbm})^2 + (I^n_{xfM})^2 + (I^n_{ybm})^2 + (I^n_{yfM})^2} & \text{if } \beta^n > 0 \\ \sqrt{(I^n_{xbM})^2 + (I^n_{xfm})^2 + (I^n_{ybM})^2 + (I^n_{yfm})^2} & \text{if } \beta^n < 0 \end{cases} \tag{9}$$

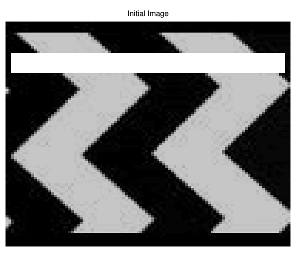
The (i, j) are dropped for simplicity, b represents a backward difference, f represents a forward difference, and m, M denote the min and max between I_x or I_y and 0. As equations, are applied to pixels lying on ∂D, known values from outside D are used to update the isophote directions and Laplacians. Bertalmio et al. stress that this is a crucial element of their model, since it is these values drawn from a small dilation of D (denoted by D^ϵ), that allow the algorithm to approximate the contour directions and gray values along the boundary and propagate them forward into the inpainting region.

The numerical implementation of the diffusion process is identical to that in section 3. Typically a set number of iterations, A, of BSCB inpainting are performed followed by B iterations of diffusion in the same inpainting region D.

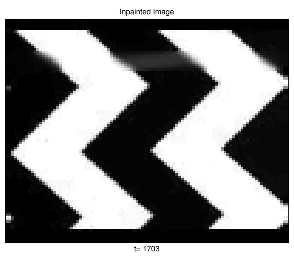
4.2 Examples

All examples in this section were run with time step $\Delta t = 0.1$, and with $A = 15$ iterations of BSCB inpainting followed by $B = 2$ iterations of diffusion, just as in [1]. The norm of image I, $||I||$, is plotted against iterations to see if it is progressing towards a steady state. When $\frac{\partial I}{\partial t} = 0$ in (5), $||I||$ should stop changing over time. Also, all mask values were set to 1.3 in the image so that the algorithm could detect which values had been updated or not.

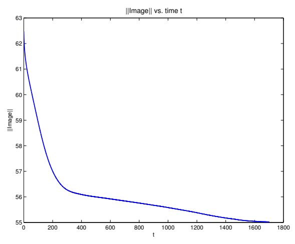
The method is first tested on the same image used in Figure 2 for TV inpainting. We saw that TV inpainting was unable to recover the angles and gray values of the contours surrounding the inpainting region D. BSCB inpainting however is able to successfully propagate surrounding contours into the unknown region. However, it does take a large number of iterations, in this case the final time is t=1700 with a step size of 0.1. For any larger regions than this however, the algorithm is unsuccessful. Methods that have superseded BSCB inpainting such as one developed by Telea in [7] state that the reason BSCB is unable to propagate isophote directions in larger regions is because isophote and Laplacian approximations degrade over time. They are subject to numerical diffusion, which helps to stabilize numerical solutions to PDEs but causes information to be lost as the inpainting algorithms progress in time.



(a)

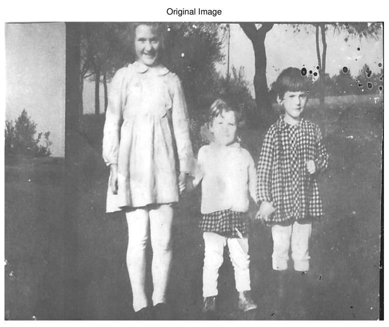


(b)

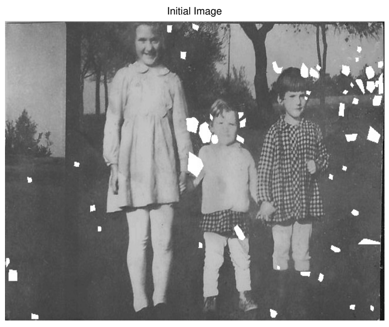


(c)

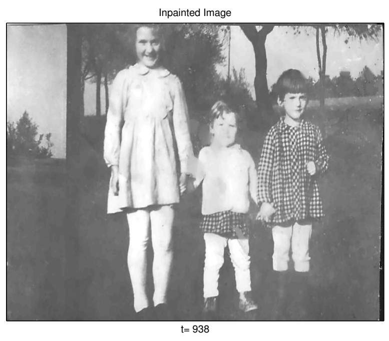
Figure 6: Results with BSCB using the same image as in Figure 2. This 100×100 pixel image, was run with t=1703, which took approximately 10 minutes.



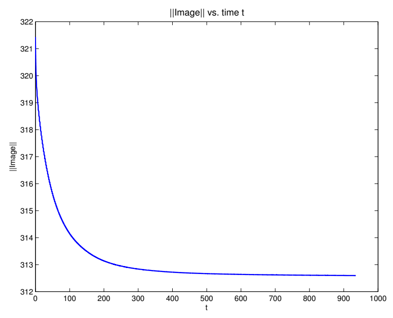
(a)



(b)



(c)



(d)

Figure 7: Results with BSCB using an old damaged photo (size: 646×417 pixels). The algorithm was run with final time t=938, which took approximately 30 minutes.

However, BSCB inpainting does work well in small regions, especially for applications which repair scratches or fading in old photographs. An example is given in Bertalmio's paper, and another one is shown in Figure 7. The photograph is quite degraded so some of the larger inpainting regions still remain somewhat blurred but small spots and scratches are successfully removed.

Figure 7 shows that the BSCB algorithm works well on any shape of inpainting region. As a result it can be used to remove text covering an image, an example of which is shown in Figure 8.

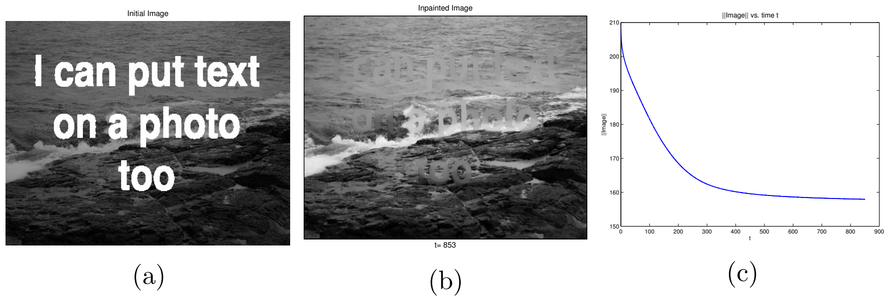


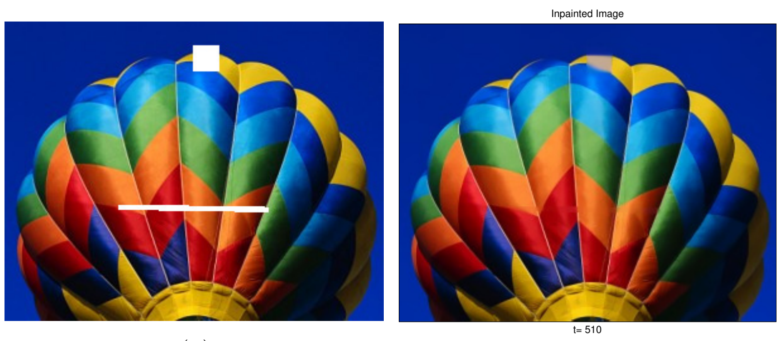
Figure 8: Results of BSCB for inpainting a picture with text (size: 500×334 pixels). The algorithm was run with final time t=853, which took approximately 45 minutes. We can see that it is able to fill in the gray values of the image very well, but it is unable to replicate the texture of the water and rocks.

As previously mentioned, the BSCB can give poor results when applied to large inpainting areas. Bertalmio et al. consider D large for any region spanning 30 pixels or more, which is quite restrictive. The algorithm is also not designed to replicate texture or pattern as can be seen in Figure 8. Other algorithms for reproducing texture must be combined with the BSCB algorithm in order to achieve optimal results [3].

4.3 Extension to Colour Images

BSCB inpainting can be extended to colour images, since they are represented by $M \times N \times 3$ arrays, where the three dimensions represent RGB values. Each colour dimension ($M \times N$) can be processed separately by the algorithm and can be formed back into an $M \times N \times 3$ array. Bertalmio et al. use a different colour model from RGB, but it can also be broken down into 3 arrays, so for this project only the RGB model is used. However a significant drawback to processing colour images with this algorithm is that the computation time of the algorithm will increase dramatically. Black and white images, approximately 500×500 pixels in size took approximately 30-45 mins to process, so colour images will take at least three times longer, if not more. Hence for applications such as this, fast and more efficient algorithms are necessary. Telea introduces an improvement of Bertalmio's method in [7] using a so called Fast Marching Method. Bertalmio et al. also improve on their work by demonstrating that the steady state PDE model in (5) has a connection with fluid dynamics. As a result efficient and well-developed methods from fluid dynamics can be applied to speed up the inpainting process [2].

Figure 9 shows the ability of the algorithm to fill in several different colours within a fairly small region, and keep them well divided along their contours. However, there is some blurring evident especially with the square mask at the top of the parachute. For this image, some of the RGB dimensions do better than others, so certain RGB layers may need to be run longer. In this case implementing the algorithm with a tolerance on the norm of I would be useful.



(a)

(b)

Figure 9: Results of BSCB for inpainting a colour picture. (size: 300 $\times$ 240 pixels). The algorithm was run with final time t=513, which took approximately 10 minutes for all three RGB layers. We can see that it does well in the thinner inpainting region, but performs slightly worse for the square mask at the top of the parachute.

In Figure 10 the inpainting algorithm is used to remove wrinkles from around Einstein's eyes and forehead. This example further shows the applicability of the method to retouching photos. One thing to note is that the algorithm is unable to inpaint regions such as eyes or ears, that require previous information unrelated to the continuation of contour lines.

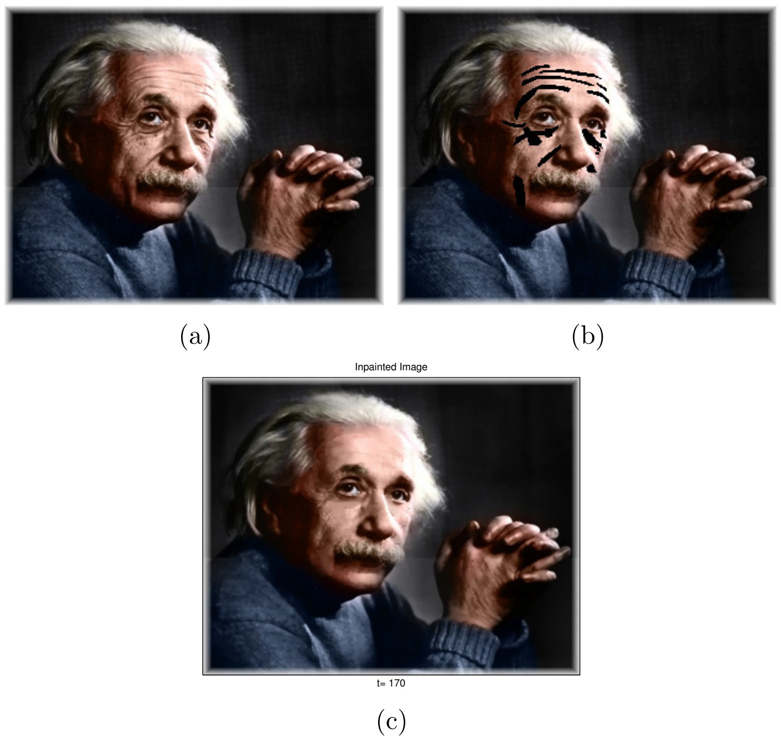


Figure 10: Results of BSCB for retouching a colour picture. (size: 308 $\times$ 266 pixels). The algorithm was run with final time t=170 for each RGB layer. The process took approximately 5 minutes to reveal a younger looking Einstein!

5 Conclusion

Both TV and BSCB inpainting are able to give very good results on a variety of images and applications. They are able to remove scratches or damage from degraded photographs, they can be used to remove text from images, and even to remove specific objects such as a fence, or wrinkles from images. The BSCB algorithm is able to propagate RGB values and structural information related to isophotes into missing regions, while preserving a certain level of smoothness within an image. Both algorithms however, give poor results when applied to large inpainting regions of 30 pixels or more. For TV inpainting this is due to the fact that it does not propagate gray values in the direction of isophotes. For BSCB inpainting this is a result of the fact that isophote approximations degrade as they progress into missing regions. However this issue can be resolved with more efficient numerical algorithms that are better able to preserve the isophote directions.

Another major issue with both algorithms is computation time. For TV inpainting convex optimization methods such as projected gradient descent or the primal dual algorithm [9] can be applied to improve efficiency. For BSCB inpainting, methods from fluid dynamics can be applied [2] or similar methods based on the same principles [7].

Implementing these newer methods would definitely be a useful direction for future work. It would also be interesting to apply the algorithms to inpainting video or sound files, since they are already quite versatile for a wide range of images.

References

[1] M. Bertalmio, G. Sapiro, V. Caselles, and C. Ballester. *Image Inpainting* Proc. SIGGRAPH, pp. 417-424, New Orleans, LA, July 2000.

[2] M. Bertalmio, A.L. Bertozzi, and G. Sapiro. *Navier-Stokes, Fluid Dynamics, and Image and Video Inpainting.* Proc. IEEE Computer Vision and Pattern Recognition, pp. 355-362, Dec. 2001.

[3] M. Bertalmio, L. Vese, G. Sapiro, and S. Osher. *Simultaneous Structure and Texture Image Inpainting* IEEE Transactions on Image Processing, 12, pp. 882-895, 2003.

[4] T.F. Chan and J. Shen. *Variational Image Inpainting*, Comm. Pure Applied Math, 58, pp. 579-619, 2005.

[5] O. Naouel, and M. Khiredinne Kholladi, *An Image Inpainting Algorithm based on K-means Algorithm* 2013 World Congress on Multimedia and Computer Science, pp. 47-52, 2013.

[6] L. Rudin, S. Osher and E. Fatemi. *Nonlinear Total Variation Based Noise Removal Algorithms.* Physica D, 60, pp. 259-268, 1992.

[7] A. Telea. *An Image Inpainting Technique Based on the Fast Marching Method* Journal of Graphics Tools, 9, pp.25-36, 2004.

[8] L. Vese, `http://www.math.ucla.edu/~lvese/285j.1.09f/ROFScheme_TimeDep.pdf`

[9] L. Younes, *Mathematical Image Analysis: AMS 493 Course Notes*

6 Appendix: Description of Matlab Files

There are two main files from which the examples for this report are generated:

- `TVInpaintExamples`
- `BSCBExamples`

`TVInpaintExamples` calls `TVInpaint` and `TVtol`. `TVtol` contains the gradient descent implementation.
`BSCBExamples` calls `BSCBInpaint`, `BSCBIter`, `TVIter`, and `laplacian`. In the case of a colour image `BSCBInpaintColour` is called instead of `BSCBInpaint`. `BSCBIter` contains the main algorithm given in [1].
The image inputs to `TVInpaintExamples`, and `BSCBExamples` are in the folder `Images`. `Images` contains all of the images being inpainted and their binary masks.
There are two folders where outputs are sent: `TVOutput`, and `BSCBOutput`. All figures generated are sent as eps files to these folders.