Curvature-based Image Registration:

Review and Extensions

By

Stuart Alexander MacGillivray

An essay
presented to the University of Waterloo
in fulfillment of the
essay requirement for the degree of
Master of Mathematics
in
Combinatorics and Optimization

Waterloo, Ontario, Canada, 2009

Abstract

Image registration is an optimization problem that, while ill-posed, has useful applications in medical analysis, criminology, and geomatics. In this paper, I present a number of the criteria and methodologies used in this optimization problem, with a particular focus on Modersitzki and Fischer's curvature-based non-linear registration technique. I present some minor exploration of alternatives using intensity scaling, using markers and, finally, the steepest descent. While the intensity scaling modification worked, it had little notable effect. Markers proved to be more useful but it is clear that alternate approaches, such as handling the markers in a pre-processing step or applying their penalty as part of the differential operators, could grant improvements. Finally, steepest descent held little promise; to have a hope of decent running time and useful deformations, the gradient and approximations used may need more investigation.

Acknowledgements

This paper would not have been possible without the help of many people along the way. First, I would like to thank my supervisor Dr. Stephen Vavasis for introducing me to this problem, providing me with ideas, and helping to ensure that I graduate.

Second, I would like to thank Levent Tuncel and Justin Wan for their helpful suggestions as readers.

Third, I'd like to thank Joanne for sticking with me through this chapter of my life. It's been rough, but thanks to her I've had the motivation to keep going.

Fourth, I'd like to thank Joan and Robert Meade. They introduced me to the joys of the creative arts. Their expertise in teaching theatre arts has helped me immeasurably.

Fifth, I'd like to thank my aunt Elizabeth for her advice over the years on essay-writing and presentations, along with being an audience for my attempts to explain math.

Finally, I'd like to thank all of my relatives. They've provided me with encouragement, cookies, and threats of doom to get this done!

This essay is dedicated to my Mom, who always believed in me, and encouraged me to laugh and keep on track through the tricky times.

TABLE OF CONTENTS

1. INTRODUCTION

Image registration is an optimization problem that, while ill-posed, has useful applications in many sciences. Medical analysis, criminology, and geomatics are merely a few fields that can make use of automated or semi-automated image registration. In this paper, we present a number of the criteria and methodologies used in this optimization problem, with a particular focus on Modersitzki and Fischer's curvature-based non-linear registration technique (Fischer and Modersitzki, 2003). Furthermore, we present some minor exploration of alternatives to the particular optimization method used by Modersitzki and Fischer.

2. PROBLEM

The core idea of image registration is straightforward. Given as input a reference image R and a deformable template image T, an image registration algorithm outputs a deformation $\mathbf{u}$. This deformation gives displacements for the vector of locations X and, when these displacements are applied to the template T, the modified template should more closely match the reference R. This is an optimization problem, aiming to minimize the difference between the deformed template $T(X-\mathbf{u}(X))$ and the original reference $R(X)$. In subsequent examples, X will be used to represent all indices in the image simultaneously. In other examples, x will be used for a single location in the image, with $\mathbf{u}(x)$ being the deformation at that location. When the single-point notation is used, Ω is the applicable region – the set of points in the image. In discussions of the deformation $\mathbf{u}$ outside formulae, boldface has been used for emphasis.

Despite the shared model, different methods of image registration will vary in optimization methods and, more importantly, the metric used to determine the difference. Some algorithms will place strict restrictions on the deformation $\mathbf{u}$, some will seek to focus on specific aspects of the image, and others will include a regularization term to ensure smoothness of the deformation. Some of these methods are reviewed in Chapter 2 of this report.

Applications for image registration range from medical to geographical. One medical application is three-dimensional modeling of the human brain, achieved through amalgamating and

adjusting slices of a paraffin-soaked slice. Another is present in radiology, automatically adjusting images for faster comparison; if applied to mammograms, for instance, cancerous cases could be identified more quickly. Image registration also has applications in identification; by matching faces or retinas to a database of references, rapid identification of a person could be achieved. Shamir et al. (Shamir, Ling, Rahimi, Ferrucci & Goldberg, 2009) have recently attempted analogous work with knees. Although the rate of success is low, the technique holds promise. Finally, image registration has been used in geomatics. Photogrammery is the technique of aligning aerial or satellite images to form a complete map, and an obvious use for image registration techniques.

While we acknowledge the wide variety of solution methods available to this problem, and discuss several of them in this report, image registration algorithms have not been exhausted. This paper examines the framework leading to the curvature-based non-parametric image registration problem as defined by Fischer and Modersitzki, and presents some extensions or replacements for their solver of the Euler-Lagrange conditions based on Discrete Cosine Transforms (DCT).

3. BACKGROUND

In order to find a useful deformation of the template, we must first determine what a 'good' deformation is. In general, two requirements must be satisfied. First, the deformed template and the reference image should match in some fashion. The type of matching depends on the algorithm used; spatial parameters of the image, specific points, or total differences are some of the possible measures of closeness.

The second requirement is some form of regularity in the deformation. Left unchecked, a deformation could be found that changes every pixel of the template to match every pixel of the reference, but such a process would be useless for analysis. The regularizer can require that the deformation be gradual, tend towards simple linear shifts of the entire template, or otherwise be smooth and semi-continuous in its application. It can take the form of either a hard constraint, permitting only certain types of deformation, or a softer penalty function, penalizing deformations for abnormalities.

Unfortunately, as presented, this problem is ill-posed. Much like the heat equations, there are too many equally valid solutions to the deformation. Even in simple cases, requiring only movement to make elements match, the question of transition versus rotation remains open. Which solution can be considered ‘best’ is uncertain, due to vague definitions in the problem. While the regularization terms help somewhat, most image registration implementations simply seek a 'good' deformation, rather than try for a badly defined 'best'.

3.1 Parametric Image Registration

Parametric image registration consists of techniques based on finite sets of parameters and/or image features. Foremost among these techniques is landmark-based image registration. A number of markers are specified in both the reference and the template, and a transformation is sought that allows these to align. This transformation could be a linear registration, a quadratic one, or ideally some other type of smooth registration.

Simply matching the markers and nothing else, however, can result in ill-formed solutions, as explained later. Evaluating the smoothness of the transformation could be more useful, and a modified landmark-based registration is presented later in this paper. Landmarks, however, are difficult to automatically locate. While some automation of marker finding is possible, human intervention may still be needed, reducing the autonomy of this method drastically.

Principal axes are put forward as an alternative; the centre of an image, along with the vectors along which its main axes lie, are easily found through basic numerical analysis. Finding a transformation between the axes of the reference and the template is easy, but has its weaknesses. In particular, the principal axes method holds too much ambiguity. With even different rectangles sharing the same features by this measure, the ability of the principal axes method to match images is limited at best.

Alternatively, the image features could be expanded to include the whole image, and the parameters to optimize could be restrained. Several methods restrict themselves to affine linear transformations, aiming to optimize only a few terms. Among these are some intensity-based

schemes using Gauss-Newton methods, and a few schemes using new distance measures. While intriguing, these go beyond the scope of this paper.

3.2 Non-Parametric Image Registration

In non-parametric image registration, by contrast, we neither focus on specific points nor demand an affine linear transformation. While such a transformation may be preferred, non-linear deformations are possible. As such, we aim to minimize the metric

$$J[R,T; u] := D[R,T; u] + \alpha S[u]$$

where D is a metric for the difference between the reference and the deformed template, and S is a measure of the deformation's smoothness. For the difference, the sum of squared differences is a popular metric which can be quickly computed over Ω.

$$D[R,T; u] := 1/2 \, || \, R - T_u \, ||^2 = 1/2 \int_\Omega (T(x-u(x)) - R(x))^2 \, dx$$

The smoothness measure, on the other hand, varies widely with the specific non-parametric method used. Finally, depending on the problem to be resolved, appending a penalty term is possible if particular solutions are to be avoided.

3.2.1 Elastic

The elastic smoothness measure is motivated by the physics of objects being deformed. For this method, the smoothness measure

$$S^{elas}[u] := \int_\Omega \mu/4 \sum_{j,k} (\delta_{xj} u_k + \delta_{xk} u_j)^2 + \lambda/2 \, (\text{div } u)^2 \, dx$$

is used, where λ and μ are the so-called Lamé constants, reflecting material properties. The smoother in this case represents the stress on each point in the simulated object. This method is used in fields where the images being registered are drawn from objects with elastic properties, such as slices of a brain used for 3D reconstruction.

Elastic-based registration has its advantages and its disadvantages. It has rapid implementations, making it quick to use, and its physical motivations make it useful in some cases. However, any deformations will be small and local, as opposed to more global transformations such

as an overall displacement. Lastly, it can be too rigid for some images. If the object being modeled is more mutable, then elastic smoothness is too constraining.

3.2.2 Fluid

In some cases, elastic registration is too rigid for the problem. For cases where the material imaged is more prone to change, the fluid smoothness parameter is used. This is relatively straightforward to understand; instead of directly taking the smoothness of a deformation **u**, the fluid smoothness measure finds the velocity **v** of the displacement field **u**, and takes the elastic smoothness of **v**. To resolve this velocity, an arbitrary time step is implemented, with time represented as t. This step can be matched to subsequent iterations.

$$S^{fluid}[u] := S^{elas}[v]$$

$$v(x,t) = \delta_t u(x,t) + \nabla u(x,t)v(x,t)$$

This fluid model was proposed by Christensen (Christensen, 1994), and is used in cases where the object being modeled is fluid-like; it does not accurately represent elastic objects, and can be used to obtain deformations completely altering the nature of an image. Some classical examples include turning a hand into a disc, or a circle into the letter C.

3.2.3 Diffusion

Switching entirely from the physically motivated model, we come to diffusion registration. Rather than attempt to emulate physical properties as in the elastic or fluid models, this method attempts to evaluate more closely the smoothness of the displacement itself. The regularization term for this, as introduced by Fischer and Modersitzki, is the sum of the norm of the gradients of **u** in each dimension.

$$S^{diff}[u] := \tfrac{1}{2} \sum_j \int_\Omega \| \nabla u_j \|^2 \, dx$$

This regularizer, ideally, reduces penalties for smooth deformations. Furthermore, its setup allows easily computed Euler-Lagrange equations. This makes several optimization methods more useful; the conditions are as follows.

$$f(x,u(x)) + \alpha\Delta u(x) = 0,\ x \in \Omega$$

$$f(x,u(x)) := (R(x) - T_u(x))\nabla T_u(x)$$

The diffusion registration method is named as such because the partial differential equations it approximates can be viewed as a generalized diffusion equation. Furthermore, these PDEs can be solved through an iterative process. Beyond this, variants exist. The Thirion's demons method for matching of images can be implemented, or the basis of the regularizer can be focused on velocity rather than displacement – a fluid-like model, as opposed to an elastic-like one.

This method is not without its drawbacks. While evaluating the gradients is sensible, the resulting method is not physical. Each component of the resulting displacement is akin to a solution to a heat equation, but a unified model for the resulting displacement field is not clear. Fortunately, the non-physicality is seldom significant in most real-world applications.

Diffusion registration has its advantages as well. Spatial directions are decoupled, allowing block diagonalization. Additive operator splitting, as proposed by Fischer and Modersitzki, permits a linear-complexity solution technique for each block. Hence, diffusion registration is quick, particularly on high-dimensional data.

3.2.4 Curvature

Modersitzki and Fischer were motivated to look beyond diffusion registration by desire for a measure with a certain kernel. As affine linear transformations involve no deformation of the image, the smoothness measure should not penalize their use. As such, a measure with a kernel including such linear transforms was desired. As such transforms resulted in nonzero smoothness under diffusion, Modersitzki and Fischer needed a new measure.

$$S^{curv}[u] := \tfrac{1}{2} \sum_j \int_\Omega (\Delta u_j)^2 \, dx$$

By taking the Laplacian of **u** instead of the gradient, a Cx+b transform has a smoothness measure of zero, fulfilling the stated goal. This measure shares several properties with diffusion registration, not the least of which is its computational efficiency.

3.2.5 Other

Other options exist beyond the unified model presented here. Droske and Ring (Droske & Ring, 2006) present a level-set method based on segmentation and edge-alignment, while Keeling (Keeling, 2007) has done work with optical flow and new intensity-scaling distance measures. Most of these fall outside the scope of this paper, and our focus will be on the diffusion and curvature-based methods.

4. SOLVING THE SYSTEM

Once our measure has been determined, the goal of image registration is to solve for the best deformation **u** that can minimize the distance measure while fulfilling certain properties. The measure and properties, to some extent, determine how this problem will be solved. The parametric measures mentioned above can sometimes be solved directly, while other measures sometimes require more procedural algorithms.

Most parametric measures, by design, lend themselves easily to direct solution of a system. With a fixed number of variables, solutions can be straightforward. Axes-based registration in particular is fast to solve; after finding the principal axes of the template and reference, an affine linear transformation to make the axes match is simple to solve. Likewise, landmark-based methods constrained to linear or quadratic deformations can quickly be solved for the required variables. However, the faults of the parametric methods show themselves here. Quadratic transformations in particular can technically satisfy the requirements of landmark-based registration while being otherwise useless for image registration. While the example below matches the fingertip landmarks exactly, the rest of the deformed image is in no state to be compared to a reference.

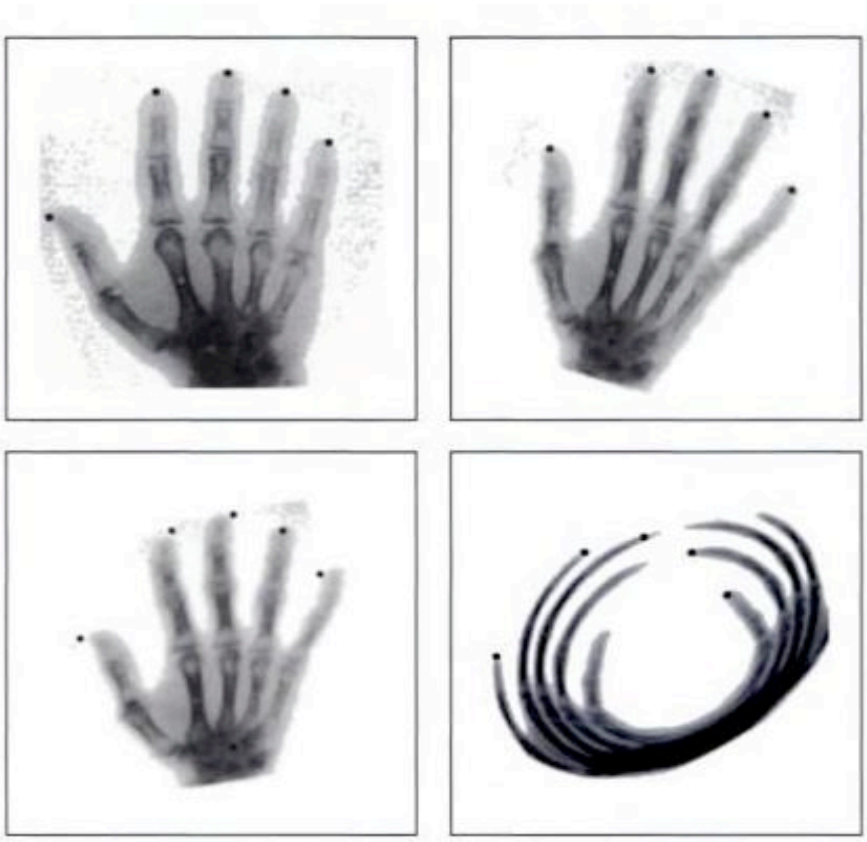

FIG. 4.1 Landmark-based image registration. TOP LEFT: reference, and TOP RIGHT: template with landmarks (black dots), BOTTOM LEFT: parametric linear registered, BOTTOM RIGHT: parametric quadratic registered.

(Citation: Modersitzki, J. Numerical methods for Image Registration. Page 31.)

Another approach, used in an implementation of curvature-based registration by Fischer and Modersitzki, involves solving the Euler-Lagrange conditions for the image registration metric. With the gradient of the objective function and some approximation, the deformation can theoretically be solved for a gradient of zero. In practice, too much approximation is necessary to obtain an ideal deformation. To resolve this, a fixed point iteration is used, using the gradient of the objective function on a given deformation to obtain the next one. This is repeated until various conditions are met. If the calculation of the various deformations can be done efficiently, this method can be fast. It is not without its disadvantages, however; in particular, any critical points of the objective

function can be traps for the algorithm. As the Euler-Lagrange conditions are the only ones checked, the iteration would stop upon reaching any stationary point.

Steepest descent is another optimization technique with some use in this field. It requires several iterations, like fixed-point. Instead of solving the Euler-Lagrange equations at each step, however, the algorithm finds a direction in which to improve the deformation, calculates how far in that direction to go, and produces an updated deformation that results in a lower objective value. This process is repeated until sufficient progress has been made. Steepest descent methods, however, have two weaknesses. The first is that the algorithm, while less prone to the weaknesses of the earlier fixed-point iteration, can still fall prey to critical points. The second is that steepest descent will sometimes take steps that are too small. While the algorithm could reach the same result as another method in this scenario, it would take much longer.

One novel approach for image registrations is Thirion's demons. Inspired by Maxwell's demons from thermodynamics, the idea is to place several 'demons' at various locations on the template. These demons then decide whether moving a given particle of the template would reduce the difference between the reference and the template. Over multiple iterations, the demons effectively sort the elements of the template to produce a final deformation. In practice, a force field is often computed to decide on the direction of movement, and diffusion registration is often used to regularize the smoothness of the method. This approach is akin to a piecewise steepest descent. The main drawbacks are the smoothness of the method, and the difficulty in choosing some of the parameters.

5. FISCHER-MODERSITZKI APPROACH

As mentioned above, we characterize all deformations of an image by a measure

$$J[u] := D[R,T;u] + \alpha S[u]$$

where D is a distance measure between the reference and deformed template, and S is a measure of the deformation. We seek to minimize this joint measure, using any of a number of numberical schemes. A simple necessary condition for a minimizer **u** is that the Gâteaux derivative of the objective

function vanishes. The derivative of the joint functional splits into a sum, with the derivative of the distance measure being $-f(x,u(x))$.

$$f(x,u(x)) := (R(x) - T_u(x))\nabla T_u(x)$$

The derivative of the smoothness parameter depends on the smoothness measure used. Diffusion and curvature smoothness were designed with this in mind. For diffusion, the derivative of the smoothness is simply the Laplacian of u. For curvature, it is the Laplacian squared. The Laplacian can be approximated with a discrete operator. The diffusion example, being more straightforward, is as follows for two dimensions.

$$\Delta u_j(X) \approx S^{diff,d} * u_j(X)$$

$$S^{diff,2} := \begin{pmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{pmatrix}$$

This stencil can be generalized into a matrix $A^{diff,d}$ applied to U, a vector of values of **u**.

$$\Delta[u_j](X) \approx A^{diff,d} U_j$$

The curvature extension of this is in turn straightforward.

$$A^{curv,d} := (A^{diff,d})^2$$

Finally, we have modified Euler-Lagrange conditions for the diffusion case and, by replacing A, the curvature case.

$$f(X,U) + \alpha I_d \otimes A^{diff,d} U = 0$$

With the derivative approximated, the next logical step is to use some form of numerical methods to approach such a minimizer. One option, as proposed by Henn (1997), is a gradient-based steepest descent method. Using the gradient of the distance measure, projecting it as appropriate, the permutation ideally approaches an optimal state. We present an attempt to apply this method to curvature-based image registration.

Another scheme, first proposed by Thirion in 1995, is a demons-based approach, as outlined earlier in this paper.

Finally, the approaches principally used by Fischer and Modersitzki are based on the Euler-Lagrange equations as discussed above. Rather than following the gradient, this scheme attempts to solve the conditions directly, using fixed-point iteration. Representing our differential operator with A, and starting with some initial guess such as $u^{(0)} = 0$, $u^{(k+1)}$ is defined by

$$\alpha A[u^{(k+1)}](x) = -f(x,u^{(k)}(x)) \text{ for all } x$$

This scheme can be further stabilized with a parameter τ as follows. With $F^{(k)}$ defined as $f(X,U^{(k)})$ and n being the number of points to be considered, the curvature approach can be modified to become

$$(I_n + \alpha\tau A^{curv,d})U_j^{(k+1)} = U_j^{(k)} + \tau F_j^{(k)}$$

The most computationally intensive step of this scheme is, as expected, finding a way to solve $(I_n + \alpha\tau A^{curv,d})^{-1}$ or otherwise solve for $U_j^{(k+1)}$. Where curvature-based registration is concerned, Fischer and Modersitzki present a clever solution to this dilemma. By using a Discrete Cosine Transform on U and F, the revised effects of A^{curv} effectively form a diagonal matrix, making inversion trivial.

$$D^{diff,2} := \text{diag}(d_{j1,j2}, j_1 = 1\ldots n1, j_2 = 1\ldots n2)$$

$$d_{j1,j2} = -4 + 2\cos((j_1 - 1) \pi / n_1) + 2\cos((j_2 - 1) \pi / n_2)$$

$$D^{curv,2} = (D^{diff,2})^2.$$

Piecing these elements together, the following straightforward algorithm can be used.

Inputs: R, T, α, τ. Assumptions: $U^{(0)} = 0$. j covers all dimensions.

For k = 0,…

$F_j^{(k)} = (T(X - U^{(k)}(X)) - R(X)) \delta_{xj} T(X - U^{(k)})$

$G_j = DCT(U_j^{(k)} + \tau F_j^{(k)})$

For p=1..d, $i_p = 1\ldots n_p$

$V_{j, i1,\ldots,ip} = G_{i1,\ldots,ip} [1 + \tau\alpha d^2_{i1,\ldots,ip}]^{-1}$

End,

$U_j^{(k+1)} = DCT^{-1}(V_j)$

End.

MATLAB code implementing this is presented in the appendix, terminating after a certain number of iterations or when the difference between iterations is trivial.

6. MODIFICATIONS

In order to further explore this algorithm, we attempted several modifications based on our research.

6.1 Intensity scaling

The first was a simple extension implementing basic intensity scaling. Before starting the curvature based registration process, the reference image had its values scaled so that average intensities of reference and template were on par. This simple correction was easy to implement, but had negligible effect on the end result. While more involved methods are possible, global intensity scaling seems largely unnecessary. A 'correct' deformation of the image seems to have a minimal sum of squared differences from the template, with or without a scaling of intensity.

6.2 Landmarking

Another modification of Fischer and Modersitzki's algorithm was parametrically motivated. While landmark-based registration was not implemented as presented, a penalty term was applied to the objective function. This would allow a user to specify markers for the reference and template. Our choice of penalty term to implement this regularizing variable times the squared norm of the distance between the reference marker and the deformed template marker. This is a geometric distance, as opposed to a difference in pixel intensity; as a result it can be compared directly with the deformation u, but must be scaled carefully. Denoting the marker locations with arrays Rm and Tm:

$$P(u) := (\beta/2 \, \| (Rm - Tm) - u(Tm) \|^2)$$

The gradient of this penalty term, as required in the Euler-Lagrange equations, is the distance between deformed template marker and reference marker, in each dimension. As the Euler-Lagrange equations apply at each point, this penalty term is applied at the location of the template marker, for each pair of markers specified.

$$d/dx_j\ P(u) = \beta * [(Rm_j - Tm_j) - u(Tm)_j]$$

As presented, this gradient consists of a constant term and a simple linear term based on the deformation. This allows rapid computation of this penalty term when added directly to the distance function.

Testing of this implementation revealed the need to specify β carefully. In cases with a marker close to the edge of the image, the modified fixed point iteration overcompensated and exhibited divergent behaviour. With careful selection of markers and regularizing constants, however, the resulting deformations closely matched the markers in question. The resulting deformed templates matched the references more closely at those points, but the smoothness of the deformation suffered as a result.

A second attempt was made to achieve greater smoothness. Rather than apply the penalty term to the distance function, it was applied to the smoothness. Adjusting the Euler-Lagrange equations to include the penalty on the left side was difficult. The constant part of the gradient of the penalty – Rm-Tm – was included on the right hand side, added to the distance function. On the left, β was added to the differential operator on **u** at the marker locations. This, however, added difficulties in the inversion – with this modification, the diagonalization of A under DCT was not certain. By applying the Sherman-Morrison-Woodbury formula, however, the DCT formula could be preserved and simply adjusted for the markers.

$$(A + UCV)^{-1} = (I + A^{-1}U(C^{-1} + VA^{-1}U)^{-1}V) A^{-1}$$

UCV = β on diagonal at markers, 0 elsewhere

The results of this second attempt again revealed the need to select β carefully, while considering the other parameters. Varying the locations of markers made significant differences as well. In the following test cases, markers were used on the thumb, pinky finger, and either side of the wrist. In successful cases, this resulted in the markers being dragged closer to their correct locations, as constrained by the curvature-based smoothness. The first two tests were terminated after 400 iterations; the other three were allowed to run for 20000 iterations in case convergent behaviour emerged. In the final case presented here, convergence to a small tolerance was achieved after

12969 iterations of the algorithm. That said, useful results tend to present themselves quickly; the tolerance currently used in testing needs reformulating.


Reference R


Template T


$\alpha=1\ \tau=2\ \beta=4$


$\alpha=1\ \tau=2\ \beta=0.4$


$\alpha=1\ \tau=2\ \beta=4$


$\alpha=50\ \tau=0.02\ \beta=40$


$\alpha=10\ \tau=0.2\ \beta=1$

Evidence seems to indicate that β must be set relatively small compared to α, in order to prevent overly disrupting the differential operator. Furthermore, it can be shown that α has a significant effect on the smoothness of the image, as anticipated. However, the markers as implemented show promise. Further tweaking could be of use, along with automation of the marker selection. At present, human intervention is still required. Furthermore, the marker term includes an extra matrix inversion of size k by k, where k is the number of markers. Despite this slowdown, the landmarks show promise and could be of use for stabilization of image registration.

6.3 Steepest Descent

Finally, an effort was made to construct a curvature based steepest descent solver. We used the gradient approximation from the Euler-Lagrange conditions of the basic fixed point iterations, and a simple line search method to determine the severity of the descent at any given iteration. This line search required sufficient descent, based on the joint functional of curvature based registration. For simplicity, an approximation of the Lagrangian of the deformation was used. Using the same

convergence requirements as in our implementation of fixed point iteration, we set to testing the steepest descent method.

After some experimentation, the steepest descent method proved suboptimal. The best test run came by setting α to 0.01, so that smoothness of the deformation would be a secondary concern at best. At present, the steepest descent seems to rate smoothness too highly; even with a low α, the process terminated after five iterations with little significant change.

Reference Template Steepest Descent, α=0.01

Alternate methods of calculating gradient, direction of descent, and sufficient descent are almost certainly required. The current methods of approximation seem insufficient. Furthermore, when directly compared with standard curvature-based image registration, steepest descent is slower. On the 60x60 image used above, the time taken for five iterations of steepest descent was equivalent to the time required for three hundred iterations of fixed-point solving.

7. DISCUSSION AND CONCLUSIONS

Some of our modifications to the curvature based implementation achieved greater success than others. The initial test of intensity scaling proved less than fruitful; while the modification works, it had little effect on the end result. While more involved methods could be used, the ability of the basic algorithm to overcome simple intensity problems seems sufficient. Markers proved to be more useful; while the basic implementation had problems, the idea seems sound. Alternate approaches, such as handling the markers in a pre-processing step, could grant improvements. Finally, steepest descent held only minor promise; to have a hope of decent running time and useful deformations, the gradient needs to be redone, with a different approximation used.

Some of these results show that there is room for improvement. More elaborate intensity scaling could be used, and more complicated test cases should be examined. In photography cases where one photo is overexposed, average brightness could vary across the photo. An intensity algorithm looking for such cases could improve curvature registration. For markers, a worthwhile area of research could be the automation of landmark selection. If such processes are combined with a penalty term on curvature registration, the result could be a relatively smooth deformation that matches up the most important features of the images. Shifting the focus of the penalty to the differential operator helped the algorithm in resolving the penalties more directly; this second attempt requires further testing, particularly in conjunction with automated marker selection. Thirdly, steepest descent could work if the gradient were computed in some fashion other than the approximation used for the normal DCT-based algorithm.

Outside our three extensions, other modifications could be made to the curvature algorithm. Although steepest descent gave disappointing results, modifications to it combined with a Thirion's demons approach could prove interesting when used with curvature-based registration. Modifications to use curvature smoothness on a fluid-based velocity field have been explored by Fischer and Modersitzki, but could possibly be combined with other penalty terms or a secondary smoothness function. Alternate difference measures could be examined, as well. While these are only ideas, the successes thus far show that room remains for exploration of further extensions to curvature-based registration.

References

Christensen, G.E. (1994). Deformable Shape Models for Anatomy. PhD thesis, Sever Institute of Technology, Washington University.

Droske, M. & Ring, W. (2006). A Mumford-Shah level-set approach for geometric image registration. SIAM Journal on Applied Mathematics, 66(6), 2127-2148.

Fischer, B. & Modersitzki, J. (2008) Ill-posed medicine - an introduction to image registration. Inverse Problems, 24(3), Art. No. 034008.

Fischer, B. & Modersitzki, J. (2004). A unified approach to fast image registration and a new curvature based registration technique. Linear Algebra and its Applications, 380, 107-124.

Fischer, B. & Modersitzki, J. (2003) Curvature based image registration. J. Mathematical Imaging and Vision, 18, 81-85.

Henn, S. (2006). A translation and rotation invariant Gauss-Newton like scheme for image registration. BIT Numerical Mathematics, 46(2), 325-344.

Henn, S. (2006). A full curvature based algorithm for image registration. Journal of Mathematical Imaging and Vision, 24(2),195-208.

Henn, S. (2005). A multigrid method for a fourth-order diffusion equation with application to image processing. SIAM Journal on Scientific Computing, 27(3), 831-849.

Henn, S. & Witsch, K. (2005). Image registration based on multiscale energy information. Multiscale Modeling & Simulation, 4(2), 584-609.

Keeling, S. L. (2007). Generalized rigid and generalized affine image registration and interpolation by geometric multigrid. Journal of Mathematical Imaging and Vision, 29(2-3), 163-183.

Keeling, S.L.(2007). Image similarity based on intensity scaling. Journal of Mathematical Imaging and Vision, 29(2), 21-34.

Knobbly ID. (2009). The Economist, 390 (8625), pp.83-84.

Larrey-Ruiz, J., Verdu-Monedero, R., & Morales-Sanchez, J. (2008). A Fourier domain framework for variational image registration. Journal of Mathematical Imaging and Vision, 32(1), 57-72.

Larrey-Ruiz, J. & Morales-Sanchez, J. (2006). Optimal parameters selection for non-parametric image registration methods in Advanced Concepts for Intelligent Vision Systems, Proceedings, 4179, 564-575.

Modersitzki, J. (2003). Numerical Methods for Image Registration, Oxford University Press.

Mumford, D. & Shah, J. (1989). Optimal approximation by piecewise smooth functions and associated variational problems. Communiciations on Pure and Applied Mathematics 42(5), 577-685.

Shamir, L., Ling, S., Rahimi, S., Ferrucci, L., & Goldberg, I.G. (2009). Biometric identification using knee X-rays. International Journal of Biometrics, 1(3), 365-370.

Vercauteren, T.,Pennec, X., Perchant, A., & Ayache, N. (2009). Diffeomorphic demons: Efficient non-parametric image registration. NeuroImage 45, S61–S72.

APPENDIX A:
MATLAB CODE

```
%  CurvImgReg – Takes reference and template as parameters, along with
regularization constants, outputs the curvature-based deformation used.
function [q,uFinal] = curvImgReg(R,T,alpha,tau)
    [sizex,sizey] = size(R);
    [verx,very] = size(T);
    uFinal = zeros(sizex,sizey,2);
    if (sizex ~= verx) || (sizey ~= very)
        print('Error: Images must be the same size.');
        return
    end
    udiff = 1; tol = 0.0001; maxit = 300; %% Initialization; tolerance here
    q = 1; %% Counter for number of iterations
    while udiff > tol && q <= maxit
        uOld = uFinal;
        uFinal = Mcurv3(R,T,uOld,alpha,tau);
        udiff = 0;
        for i = 1:sizex
            for j=1:sizey
                for k=1:2
                    udiff = udiff + (uOld(i,j,k)-uFinal(i,j,k))^2;
                end
            end
        end %% This loop should be reworked, but it's serviceable.
        if mod(q,20) == 0
        %    uFinal - this would allow output of intervening steps for
        %    testing.
        end
        q = q+1;
    end
    return
end

% Mcurv3 – Runs one step of the fixed-point iteration.
function newU = Mcurv3(R,T,u,alpha,tau)
    [sizex,sizey] = size(R);
    newU = zeros(sizex,sizey,2);
    Fgrad = feval3(R,T,u);

    for l = 1:2
        temp = u(:,:,l) + tau*Fgrad(:,:,l);

        G = dct2(temp);

        V = zeros(sizex,sizey);

        for i = 1:sizex
            for j = 1:sizey
                d = -4 + 2*cos((i-1)*pi/sizex) + 2*cos((j-1)*pi/sizey);
                V(i,j) = G(i,j) * (1 + tau*alpha*(d^2))^-1;
            end
        end
        newU(:,:,l) = idct2(V);

    end
```

```
    return
end

% feval3 – Computes the gradient of the distance function.
function f = feval3(R,T,u)
    [sizex,sizey] = size(R);
    f = zeros(sizex,sizey,2);
    temp = zeros(sizex,sizey);
    tempx = zeros(sizex,sizey);
    tempy = zeros(sizex,sizey);

    for i = 1:sizex
        for j=1:sizey
            temp(i,j) = interpolate2(T,(i-u(i,j,1)),(j-u(i,j,2))) - R(i,j);
            [tempx(i,j),tempy(i,j)] = intergrad(T,(i-u(i,j,1)),(j-u(i,j,2)));
            [s1,s2] = intergrad(T,i-0.5,j-0.5);
            [s3,s4] = intergrad(T,i+0.5,j+0.5);
            if u(i,j,1) == 0
                tempx(i,j)=(s1+s3)/2;
            end
            if u(i,j,2) == 0
                tempy(i,j) = (s2+s4)/2;
            end
        end
    end
%%    f(:,:,1) = temp*tempx; An earlier misunderstanding.
%%    f(:,:,2) = temp*tempy; Depreciated code.
    for i = 1:sizex
        for j=1:sizey
            f(i,j,1) = temp(i,j) * tempx(i,j);
            f(i,j,2) = temp(i,j) * tempy(i,j);
        end
    end

    return;
end

% interpolate2 – Implementation of bilinear interpolation.
function bilin = interpolate2(T, x, y)

    [maxx, maxy] = size(T);
    if (x > maxx)
        x = maxx;
    end
    if (x < 1)
        x = 1;
    end
    if (y > maxy)
        y = maxy;
    end
    if (y < 1)
        y = 1;
    end
    %% The boundval was stupid.  If it's outside the boundaries, use the
    %% closest point.  Fixed.
    bilin = 0;
    if (x < maxx) && (y < maxy)
        bilin = bilin + (x - floor(x))*(y - floor(y))*(T(floor(x)+1,
```

```
floor(y)+1));
    end
    if (x < maxx)
        bilin = bilin + (x - floor(x))*(floor(y)+1-y)*(T(floor(x)+1,
floor(y)));
    end
    if (y < maxy)
        bilin = bilin + (floor(x)+1-x)*(y - floor(y))*(T(floor(x),
floor(y)+1));
    end
    bilin = bilin + (floor(x)+1-x)*(floor(y)+1-y)*(T(floor(x), floor(y)));
    return;

% intergrad – Takes the derivative of bilinear interpolation.
function [gradx, grady] = intergrad(T, x, y)
    constvar = 0;
    [maxx, maxy] = size(T);
    if (x > maxx) || (x < 1) || (y > maxy) || (y < 1)
        gradx = constvar;
        grady = constvar;
        return;
    end
    gradx = 0; grady=0;
    if (x < maxx) && (y < maxy)
        grady = grady + (x - floor(x))*(1)*(T(floor(x)+1, floor(y)+1));
        gradx = gradx + (1)*(y - floor(y))*(T(floor(x)+1, floor(y)+1));
    end
    if (x < maxx)
        gradx = gradx + (1)*(floor(y)+1-y)*(T(floor(x)+1, floor(y)));
        grady = grady + (x - floor(x))*(-1)*(T(floor(x)+1, floor(y)));
    end
    if (y < maxy)
        gradx = gradx + (-1)*(y - floor(y))*(T(floor(x), floor(y)+1));
        grady = grady + (floor(x)+1-x)*(1)*(T(floor(x), floor(y)+1));
    end
    gradx = gradx + (-1)*(floor(y)+1-y)*(T(floor(x), floor(y)));
    grady = grady + (floor(x)+1-x)*(-1)*(T(floor(x), floor(y)));
    return;

% CurvIntReg – Simple implementation of intensity scaling.
% Uses the same subroutines as normal curvature based registration.
function [q,uFinal,uOld] = curvIntReg(R,T,alpha,tau)
    [sizex,sizey] = size(R);
    [verx,very] = size(T);
    uFinal = zeros(sizex,sizey,2);
    %uFinal = 0.5*ones(sizex,sizey,2);
    if (sizex ~= verx) || (sizey ~= very)
        print('Error: Images must be the same size.');
        return
    end
    udiff = 1; tol = 0.001; maxit = 300; %% Initialization; tolerance here

    Rtot = 0; Ttot = 0; %% Begin intensity correction.
    for i=1:sizex
        for j=1:sizey
            Rtot = Rtot + R(i,j);
            Ttot = Ttot + T(i,j);
        end
```

```
    end

    R2 = (Ttot/Rtot)*R;
    %% I'm not really happy with this yet; I'm going to get 'darker than
    %% dark' pixels in some cases.
    %% A more dynamic setup would be better, but that'd require redoing
    %% most of the algorithm - the gradient of the new difference function
    %% alone would be a pain and a half to compute.

    q = 1; %% Counter for number of iterations
    while udiff > tol && q <= maxit
        uOld = uFinal;
        uFinal = Mcurv3(R2,T,uOld,alpha,tau);
        udiff = 0;
        for i = 1:sizex
            for j=1:sizey
                for k=1:2
                    udiff = udiff + (uOld(i,j,k)-uFinal(i,j,k))^2;
                end
            end
        end
        if mod(q,20) == 0
        %    uFinal - Use for intermediate steps.
        end
        q = q+1;
    end
    return
end

% CurvMark – Landmarks-based penalty term.  Takes a
% pair of lists of locations, one each for Reference and Template.
% Beta is a parameter for the importance of markers; setting with care.
function [q,uFinal,uOld] = curvMark(R,T,markR,markT,alpha,tau,beta)
    [sizex,sizey] = size(R);
    [verx,very] = size(T);
    [nummarks,d] = size(markR);
    [vermarks,verd] = size(markT);
    uFinal = zeros(sizex,sizey,2);
    %uFinal = 0.5*ones(sizex,sizey,2);
    if (sizex ~= verx) || (sizey ~= very) || (nummarks ~= vermarks) || (d ~=
verd)
        print('Error: Images must be the same size.');
        return
    end
    udiff = 1; tol = 0.0001; maxit = 400; %% Initialization; tolerance here
    q = 1; %% Counter for number of iterations
    while udiff > tol && q <= maxit
        uOld = uFinal;
        uFinal = McurvMark(R,T,uOld,markR,markT,nummarks,alpha,tau,beta);
        udiff = 0;
        for i = 1:sizex
            for j=1:sizey
                for k=1:2
                    udiff = udiff + (uOld(i,j,k)-uFinal(i,j,k))^2;
                end
            end
        end
        if mod(q,20) == 0
```

```
        %    uFinal – use for intervening steps.
        end
        q = q+1;
    end
    return
end

% McurvMark – Landmark variant of Mcurv3.
function newU = McurvMark(R,T,u,markR,markT,nummarks,alpha,tau,beta)
    [sizex,sizey] = size(R);
    newU = zeros(sizex,sizey,2);
    Fgrad = feval2marks(R,T,u,markR,markT,nummarks,beta);

    for l = 1:2
        temp = u(:,:,l) + tau*Fgrad(:,:,l);

        G = dct2(temp);

        V = zeros(sizex,sizey);

        for i = 1:sizex
            for j = 1:sizey
                d = -4 + 2*cos((i-1)*pi/sizex) + 2*cos((j-1)*pi/sizey);
                V(i,j) = G(i,j) * (1 + tau*alpha*(d^2))^-1;
            end
        end
        newU(:,:,l) = idct2(V);

    end
    return
end

% feval2marks – gradient of the distance function with new penalty.
function f = feval2marks(R,T,u,Rmarks,Tmarks,nummarks, beta)
    [sizex,sizey] = size(R);
    f = zeros(sizex,sizey,2);
    temp = zeros(sizex,sizey);
    tempx = zeros(sizex,sizey);
    tempy = zeros(sizex,sizey);


    for i = 1:sizex
        for j=1:sizey
            temp(i,j) = interpolate2(T,(i-u(i,j,1)),(j-u(i,j,2))) - R(i,j);
            [tempx(i,j),tempy(i,j)] = intergrad(T,(i-u(i,j,1)),(j-u(i,j,2)));
            [s1,s2] = intergrad(T,i-0.5,j-0.5);
            [s3,s4] = intergrad(T,i+0.5,j+0.5);
            if u(i,j,2) == 0
                tempx(i,j)=(s1+s3)/2;
            end
            if u(i,j,2) == 0
                tempy(i,j) = (s2+s4)/2;
            end
        end
    end

    for i = 1:sizex
```

```
        for j=1:sizey
            f(i,j,1) = temp(i,j) * tempx(i,j);
            f(i,j,2) = temp(i,j) * tempy(i,j);
        end
    end

    for i = 1:nummarks
        diffx = (Tmarks(i,1) - u(Tmarks(i,1),Tmarks(i,2),1) - Rmarks(i,1));
        diffy = (Tmarks(i,2) - u(Tmarks(i,1),Tmarks(i,2),2) - Rmarks(i,2));
        dist = diffx^2 + diffy^2;  %% Derivative of this would be the
distance in each direction.  So:
        f(Tmarks(i,1),Tmarks(i,2),1) = f(Tmarks(i,1),Tmarks(i,2),1) -
beta*diffx;
        f(Tmarks(i,1),Tmarks(i,2),2) = f(Tmarks(i,1),Tmarks(i,2),2) -
beta*diffy;
    end

    return;


%% curvMark2: Completely implements the second landmark-based method.
%% Oversight in subsequent code effectively doubles beta; left unfixed
%% because beta's more an estimate at this point anyway.

function [q,uFinal] = curvMark2(R,T,markR,markT,alpha,tau,beta)
    [sizex,sizey] = size(R);
    [verx,very] = size(T);
    [nummarks,d] = size(markR);
    [vermarks,verd] = size(markT);
    uFinal = zeros(sizex,sizey,2);
    %uFinal = 0.5*ones(sizex,sizey,2);
    if (sizex ~= verx) || (sizey ~= very) || (nummarks ~= vermarks) || (d ~=
verd)
        print('Error: Images must be the same size.');
        return
    end
    udiff = 1; tol = 0.0001; maxit = 400; %% Initialization; tolerance here
    q = 1; %% Counter for number of iterations
    [markin1,markin2,markin3] =
MarkInversion(sizex,sizey,nummarks,markT,tau,alpha,beta);
    while udiff > tol && q <= maxit
        uOld = uFinal;
        uFinal =
McurvMark2(R,T,uOld,markR,markT,nummarks,alpha,tau,beta,markin1,markin2,marki
n3);
        udiff = 0;
        for i = 1:sizex
            for j=1:sizey
                for k=1:2
                    udiff = udiff + (uOld(i,j,k)-uFinal(i,j,k))^2;
                end
            end
        end
        if mod(q,20) == 0
        %    uFinal - Use for intervening steps.
        end
        q = q+1;
    end
```

```
    return
end


%% MarkInversion: This function works out the basics of the Sherman-
%% Morrison-Woodbury formula, and produces a matrix by which we multiply
%% A^-1 to get (A + 2*beta*markers)^-1.  Code differs from algorithm in
%% paper, on request - effectively doubles beta here and elsewhere.  It
%% scales.

function [AiU,B,V] = MarkInversion(sizex,sizey,nummarks,markT,tau,alpha,beta)
    C = eye(nummarks);
    U = zeros(sizex*sizey,nummarks);
    V = zeros(nummarks,sizex*sizey);

    for i=1:nummarks
        U((markT(i,1)-1)*sizey+markT(i,2),i) = sqrt(2*beta);
        V(i,(markT(i,1)-1)*sizey+markT(i,2)) = sqrt(2*beta);
    end

    DU = dct2(U);
    for i = 1:sizex
        for j = 1:sizey
            d = -4 + 2*cos((i-1)*pi/sizex) + 2*cos((j-1)*pi/sizey);
            DU((i-1)*sizey+j,:) = DU((i-1)*sizey+j,:) * (1 +
tau*alpha*(d^2))^-1;
        end
    end

    AiU = idct2(DU);

    block = C + (V * AiU);
    B = inv(block);
    %% curvmod = - AiU * inv(block) * V;   Code depreciated
    return
end


% McurvMark2: Second attempt at landmarks, splits the penalty function to F
% and to S.  Uses the matrix stencil calculated elsewhere to save time -
% time probably lost by unraveling and raveling U.  A more efficient way to
% handle this would be nice.

function newU =
McurvMark2(R,T,u,markR,markT,nummarks,alpha,tau,beta,markin1,markin2,markin3)
    [sizex,sizey] = size(R);
    newU = zeros(sizex,sizey,2);
    Fgrad = feval3marks(R,T,u,markR,markT,nummarks,beta);

    for l = 1:2
        temp = u(:,:,l) + tau*Fgrad(:,:,l);

        G = dct2(temp);

        V = zeros(sizex,sizey);
```

```
        for i = 1:sizex
            for j = 1:sizey
                d = -4 + 2*cos((i-1)*pi/sizex) + 2*cos((j-1)*pi/sizey);
                V(i,j) = G(i,j) * (1 + tau*alpha*(d^2))^-1;
            end
        end
        newU(:,:,l) = idct2(V);

    end
    %% If there's a better way to unravel and reravel U, I'd love to hear
    %% it.
    AU = zeros(sizex*sizey,2);
    for i=1:sizex
        for j=1:sizey
            AU((i-1)*sizey+j,:) = newU(i,j,:);
        end
    end
    AU = AU - markin1*(markin2*(markin3*AU));
    for i=1:sizex
        for j=1:sizey
             newU(i,j,:) = AU((i-1)*sizey+j,:);
        end
    end
    return
end


% feval3marks: Further modification of f(x,u(x)); this applies only the
% constant part of the landmark penalty function.
function f = feval3marks(R,T,u,Rmarks,Tmarks,nummarks, beta)
    [sizex,sizey] = size(R);
    f = zeros(sizex,sizey,2);
    temp = zeros(sizex,sizey);
    tempx = zeros(sizex,sizey);
    tempy = zeros(sizex,sizey);


    for i = 1:sizex
        for j=1:sizey
            temp(i,j) = interpolate2(T,(i-u(i,j,1)),(j-u(i,j,2))) - R(i,j);
            [tempx(i,j),tempy(i,j)] = intergrad(T,(i-u(i,j,1)),(j-u(i,j,2)));
            [s1,s2] = intergrad(T,i-0.5,j-0.5);
            [s3,s4] = intergrad(T,i+0.5,j+0.5);
            if u(i,j,2) == 0
                tempx(i,j)=(s1+s3)/2;
            end
            if u(i,j,2) == 0
                tempy(i,j) = (s2+s4)/2;
            end
        end
    end

    for i = 1:sizex
        for j=1:sizey
            f(i,j,1) = temp(i,j) * tempx(i,j);
            f(i,j,2) = temp(i,j) * tempy(i,j);
        end
```

```
    end

    for i = 1:nummarks
        diffx = (Tmarks(i,1) - Rmarks(i,1));
        diffy = (Tmarks(i,2) - Rmarks(i,2));
        dist = diffx^2 + diffy^2;  %% Derivative of this would be the
distance in each direction.  So:
        f(Tmarks(i,1),Tmarks(i,2),1) = f(Tmarks(i,1),Tmarks(i,2),1) -
2*beta*diffx;
        f(Tmarks(i,1),Tmarks(i,2),2) = f(Tmarks(i,1),Tmarks(i,2),2) -
2*beta*diffy;
    end

    return;


% CurvSteep – implementation of Steepest Descent.  Needs work.
function [q,uFinal] = curvSteep(R,T,alpha)
    [sizex,sizey] = size(R);
    [verx,very] = size(T);
    uFinal = zeros(sizex,sizey,2);
    %uFinal = 0.5*ones(sizex,sizey,2);
    if (sizex ~= verx) || (sizey ~= very)
        print('Error: Images must be the same size.');
        return
    end
    udiff = 1; tol = 0.0001; maxit = 40; %% Initialization; tolerance here
    %% Tolerated number of iterations is low; if it's working,
    %% it doesn't get even that far.
    q = 1; %% Counter for number of iterations
    while udiff > tol && q <= maxit
        uOld = uFinal;
        [uFinal] = McurvSteep(R,T,uOld,alpha);
        udiff = 0;
        for i = 1:sizex
            for j=1:sizey
                for k=1:2
                    udiff = udiff + (uOld(i,j,k)-uFinal(i,j,k))^2;
                end
            end
        end
        if mod(q,20) == 0
        %    uFinal - Intervening steps can be output here.
        end
        q = q+1;
    end
    return
end

% McurvSteep – one step of the steepest descent algorithm.
function [newU] = McurvSteep(R,T,u,alpha)
    [sizex,sizey] = size(R);
    Fgrad = feval2(R,T,u);
    Adiff = Adiff2(sizex,sizey);
    A = Adiff'*Adiff;
    tempU = zeros(sizex*sizey,2);
    desc = zeros(sizex,sizey,2);
```

```
    for i = 1:sizex
        for j=1:sizey
            tempU((i-1)*sizey+j,:) = u(i,j,:);
        end
    end

    AU = A*tempU;
    cond = 0;
    for i = 1:sizex
        for j = 1:sizey
            for l = 1:2
                desc(i,j,l) = alpha*AU((i-1)*sizey+j,l) - Fgrad(i,j,l);
                % Still not sure about the direction of descent...
            end
            cond = cond - desc(i,j,1)^2 - desc(i,j,2)^2;
        end
    end

    theta = 0.9;%%2/(1 + sqrt(5));
    rho = 10;

    j = Jfunct(R,T,u,alpha);
    newU = u - rho*desc;
    j2 = Jfunct(R,T,newU,alpha);
    if j2-j > rho*cond/4
        while j2-j > rho*cond/4 && rho > 0.0001
            rho = rho*theta;
            newU = u - rho*desc;
            j2 = Jfunct(R,T,newU,alpha);
        end
    else
        while j2-j <= rho*cond/4 && rho < 10000
            rho = rho/theta;
            newU = u - rho*desc;
            j2 = Jfunct(R,T,newU,alpha);
        end
        rho = rho*theta;
    end
    newU = u - rho*desc;
rho
    return
end

% Adiff2 – Using the matrix stencil, creates a matrix to approximate
% the differential operators on u.  The square of this is the curvature
% version.  Legacy code from earlier direct attempts before DCT
% was implemented.
function A = Adiff2(n1,n2)
    M1 = zeros(n1,n1);
    M2 = zeros(n2,n2);
    M1(1,1) = 1;
    M1(n1,n1) = 1;
    for i = 2:n1
        M1(i-1,i) = 1;
        M1(i,i-1) = 1;
    end
    M2(1,1) = 1;
```

```
    M2(n2,n2) = 1;
    for i = 2:n2
        M2(i-1,i) = 1;
        M2(i,i-1) = 1;
    end
    n1n2 = n1 * n2;
    A = -4*eye(n1n2) + kron(eye(n2),M1) + kron(M2,eye(n1));
end

% Jfunct – calculates the joint functional under curvature
% registration.  Reworking of the smoothness operator may be
% required – I don't like where the math's going.
function Jfin = Jfunct(R,T,u,alpha)
    [sizex, sizey] = size(R);
    D = 0;
    S = 0;

    for i=1:sizex
        for j=1:sizey
            D = D + (R(i,j) - interpolate2(T,i-u(i,j,1),j-u(i,j,2)))^2;
        end
    end

    uExt = [u(1,1,:) u(1,1:sizey,:) u(1,sizey,:);
        u(1:sizex,1,:) u(1:sizex,1:sizey,:) u(1:sizex,sizey,:);
        u(sizex,1,:) u(sizex,1:sizey,:) u(sizex,sizey,:)];
    udiff = -4*uExt(2:sizex+1,2:sizey+1,:) + uExt(1:sizex,2:sizey+1,:) +
uExt(3:sizex+2,2:sizey+1,:) + uExt(2:sizex+1,1:sizey,:) +
uExt(2:sizex+1,3:sizey+2,:);
    ucurv = udiff.*udiff;
    S = sum(sum(sum(ucurv)));

    %% Below is loop-based implementation of the same. Replaced for speed.
%    for i=1:sizex
%        for j=1:sizey
%            for l=1:2
%                a=4*u(i,j,l);  %% Using diff,2.
%                if i>=2
%                    a=a-u(i-1,j,l);
%                else
%                    a=a-u(i,j,l);
%                end
%                if i<=sizex-1
%                    a=a-u(i+1,j,l);
%                else
%                    a=a-u(i,j,l);
%                end
%                if j>=2
%                    a=a-u(i,j-1,l);
%                else
%                    a=a-u(i,j,l);
%                end
%                if j<=sizey-1
%                    a=a-u(i,j+1,l);
%                else
%                    a=a-u(i,j,l);
%                end
%                S = S+a^2; %% Because curvature, that's why.
```

```
%             end
%         end
%     end

    Jfin = D + alpha*S;

end
```