INVERSE PROBLEMS AND FORWARD PROPAGATION OF OPTICAL FLOW

by

JOHN MONTALBO

Presented to the Faculty of the Graduate School of
The University of Texas at Arlington in Partial Fulfillment
of the Requirements
for the Degree of

DOCTOR OF PHILOSOPHY

THE UNIVERSITY OF TEXAS AT ARLINGTON
August 2020

INVERSE PROBLEMS AND FORWARD PROPAGATION OF OPTICAL FLOW

The members of the Committee approve the doctoral dissertation of JOHN MONTALBO

Dr. Gaik Ambartsoumian ______________________

Dr. Souvik Roy ______________________

(Supervising Professors)

Dr. Suvra Pal ______________________

Dr. Hristo V. Kojouharov ______________________

Dean of the Graduate School ______________________

ACKNOWLEDGEMENTS

First, I would like to express my gratitude to my advisor Dr. Gaik Ambartsoumian for his inspiration, encouragement and invaluable guidance both academically and personally throughout my time at UTA. I am forever indebted to him for all of his support and encouragement.

I would also like to thank my co-advisor Dr. Souvik Roy. He has been an amazing ally and mentor throughout my entire graduate experience and I cannot thank him enough.

I wish to thank Dr. Suvra Pal, and Dr. Hristo Kojouharov for their interest in my research and for taking time to serve on my comprehensive committee and dissertation committee.

I would like to thank Dr. Jianzhong Su, Mrs. Lona Donnelly, Mrs. Shanna Banda, Mr. Michael Schmidtkunz, Dr. Carla Amaro-Jimenez, Dr. Tuncay Aktosun, Dr. James Alvarez, Dr. Minerva Cordero, Dr. Theresa Jorgensen, Dr. Christina Villalobos, Dr. Joseph Sifuentes, and Mrs. Libby Carrol for all of their support and assistance.

Thanks to my friends Talon, Dwight, Ian, Ariel, Zach, Justine, Izzet, Kiran, Khoa, Janessa, Crystal, Omomayowa, and so many other friends at UTA. I will never be able to properly articulate how much I value each and every one of you.

Thank you to my sisters Jessica and Jenna, mother Renee, father Juan, grandmothers Elida and Maria, uncle Estacio, aunt Irene, inlaws Sandra and Lauro, and my brother Henry. Your collective love and support has been so needed over these past few years. Thank you all for believing in me even when I didn't believe in myself.

Love to my grandfathers Eustacio and George, and my little brother Christian. Know that I did it for you.

Finally, I would like to thank my wife Suzy Luera. Without her constant love and support I would have not been able to come this far. You are my world.

July 31st, 2020

ABSTRACT

INVERSE PROBLEMS AND FORWARD PROPAGATION OF OPTICAL FLOW

John Montalbo, Ph.D.

The University of Texas at Arlington, 2020

Supervising Professors: Dr. Gaik Ambartsoumian & Dr. Souvik Roy

Optical flow is a concept originally introduced in computer vision that quantifies, and aids in the presentation of, motion (flow field) between two or more images. In essence, it is a solution of an inverse problem recovering a vector field between images through optimization techniques. This work studies the possibility of using optical flow and various techniques of forward propagation of the recovered flow field for a pair of image processing tasks in magnetic resonance imaging (MRI). It is shown that the proposed framework can be efficient in approximating missing image layers, as well as in generation of deliberately modified synthetic MRI images. We present the underlying mathematical hypotheses necessary for the applicability of the method, practical limitations associated with it, and potential mechanisms for its future improvements.

TABLE OF CONTENTS

CHAPTER 1

Introduction

1.1 Problems in Medical Imaging and Applications of Optical Flow

Imaging modalities such as magnetic resonance imaging (MRI) or ultrasound imaging (US) are incredibly powerful tools that aid in the preoperative planning and postoperative stages of medicine [13]. In the case of MRI, the 2-dimensional images of parallel slices at different depths are detailed and give great images of the body. Major abnormalities can be seen with relative ease by a trained doctor. However, within a certain class of medical problems, detailed information might be hard to see due to low contrast, objects might be time-consuming to identify, and/or such identification can even be subjective from doctor to doctor [13, 18].

For imaging scientists this creates a ripe area of interesting and challenging problems that can be solved using image classification and segmentation techniques. Image classification problems can be loosely defined as “determining the what”, whereas image segmentation “determines the where”. And because of the medical need, one could argue that these types of problems are extremely important and urgent in all of imaging science.

In the broader perspective, with the deluge of images that humans create everyday, there comes a real need to classify and sort them into understandable pieces for the use data collection and analysis purposes.

This thesis presents a mathematical framework equipped with various computational tools to assist with certain aspects of MRI.

In the medical imaging community there is a significant need to give doctors tools that may aid them in their daily practices, and the need to automatically segment and detect abnormalities is real [13,18]. To this end, the study of convolutional neural networks (CNNs) and their variants provide invaluable algorithms for the imaging scientist because of their incredible success in image segmentation and classification tasks [6,11,14,20,24]. U-net and other convolutional neural networks have been an incredible asset for image segmentation techniques tasks [6,20].

However, there is no "free lunch" when utilizing these algorithms. In order for modern machine learning techniques to be effective, they require a "large" data pool from which to learn [16]. Just how "large" this data pool needs to be is not an exact science. But as a general rule of thumb the amount of data that you should need to train your algorithm should be large enough to encompass the variation of the objects to be modeled. The number of data points that you need to properly model your problem can range in between a few hundred all the way to hundreds of thousands or more data points [16,20]. But what if you do not have access to a large pool of different images or significant instances of the objects you are trying to classify?

This is where data augmentation techniques come in handy. There are relatively simple ways to generate new cases from existing cases using affine and non-affine transformations. Some standard techniques are image scaling, rotation, reflection, shearing, and grid warping [22]. Depending on the problem in question some of these are more viable that others, but these serve as useful methods to generate new data points from existing ground truths.

Some of the existing data augmentation techniques, while simple to implement, fail to take into account existing geometry of the scene. They blindly rotate/skew/sheer an image without taking into account whether regions in question can (or should) rotate/skew/sheer (e.g., will such changes be anatomically viable?).

In many cases, access to such data is very limited and may take years or decades to accumulate (e.g. due to privacy laws and the need to de-identify medical images, expert manpower needed for manual segmentation of the images, etc). This is where new techniques and innovation can come in and help. With this in mind we take this idea and use an area of computer vision, called optical flow, to come up with a new technique for generating synthetic data. This new technique uses existing geometry of scenes and only allows variation for regions of existing images that have proven that motion is possible. It also has potential for incorporation of mathematical descriptions of viable transformations developed by synergistic teams of mathematicians and radiologists.

While we did not have time to utilize the data generated by our method to train a neural network, the obtained results in data augmentation are very promising. They can be used as an important first step in developing a platform for segmentation and analysis of clinical MRI. Another application of our method to clinical MRI is creation of additional image slices between existing ones to improve the resolution. A specific area of interest of our work was pelvic MR imaging, due to an ongoing collaboration between the University of Texas at Arlington and the University of Texas Southwestern Medical School departments of Radiology and Urology.

This thesis is arranged as follows; we first introduce the pelvic organ disorders to call attention to the need for more robust image processing techniques in this area. We then go through the necessary derivations for our modification of optical flow technique developed by Horn and Schunck (HS). We discuss the accuracy of our modified HS algorithm through a reconstruction technique and how this leads to a data creation framework. Next we discuss meaningful transformations of the motion field and how this process can be localized. We discuss when reconstruction methods will fail in the absence of "noise" and then introduce the Explicit Pixel Movement framework for codifying motion into images. We then discuss both standard and non-standard numerical methods for solving systems of ODE's and how they produce different results. And finally we discuss a specific set of transformations, that are area preserving, which is due to the well known brightness consistency condition. We also furnish plenty of codes and examples within the appendices for the curious reader.

1.2 Clinical Applications

Pelvic organ disorders (POD), or sometimes referred to as pelvic floor disorders, are a class of medical disorders that affect 1 out of every 4 women in the United States in their lifetime [13]. These conditions occur when the pelvic floor muscles and tissue tear or weaken and this erosion of muscle/tissue causes the organs in the pelvic region to move through the area in an unconstrained manner [13, 18], which wouldn't occur in a healthy pelvic region.

Various surgical and non-surgical options exist to help repair the damaged regions [13]. One of these options is the placement of synthetic implants, such as a sling or mesh to help keep a healthy boundary between the pelvic organs, to support organs from sagging, etc. [13]. However, sometimes these meshes and slings themselves cause problems of various types (infection, erosion into organs, etc) and must be removed [13]. Preoperative planning for the removal of these implants involves imaging the region with either ultrasound tomography or magnetic resonance imaging. Each of these techniques has its own pros and cons. In case of MRI, the problem is that mesh/sling and scar tissue are hard to differentiate in an MRI, i.e. they have similar contrast [13]. Distinguishing between the meshes/slings and scar tissue is time consuming and difficult even for trained radiologists. Therefore, there is a need for accurate computer algorithms that can automatically segment and classify the pelvic floor organs and implants in MRI.

The slices that are generated in an MRI sequence come in three distinctly orthogonal orientations. These are the **axial, sagittal,** and **coronal** views and are classified according to how the slices are taken in 3-D space (see Figure 1.1).

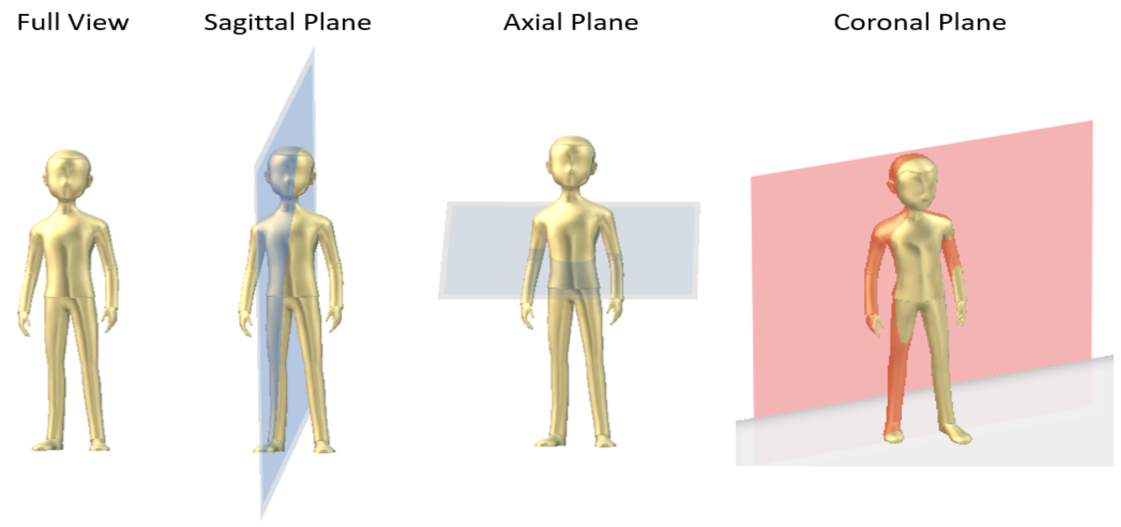

Figure 1.1. (Left) Full view. (Middle-Left) Sagittal plane. (Middle-Right) Axial plane. (Right) Coronal plane. Figure made in Paint 3D.

Typically a single view is chosen beforehand and a sequence of equally spaced images is generated in that view. This produces high resolution images in the chosen view orientation. The resolution in the direction normal to the slices (i.e. the distance between the slices) is usually very crude (4-5 times worse that the resolution within the slice).

Overcoming this hardware limitation, by (algorithmically) producing accurate approximations of 2D image slices in between the recovered slices that are far from each other, will be of substantial clinical value. A simple interpolation does not produce the desired result, as we will show later in this dissertation.

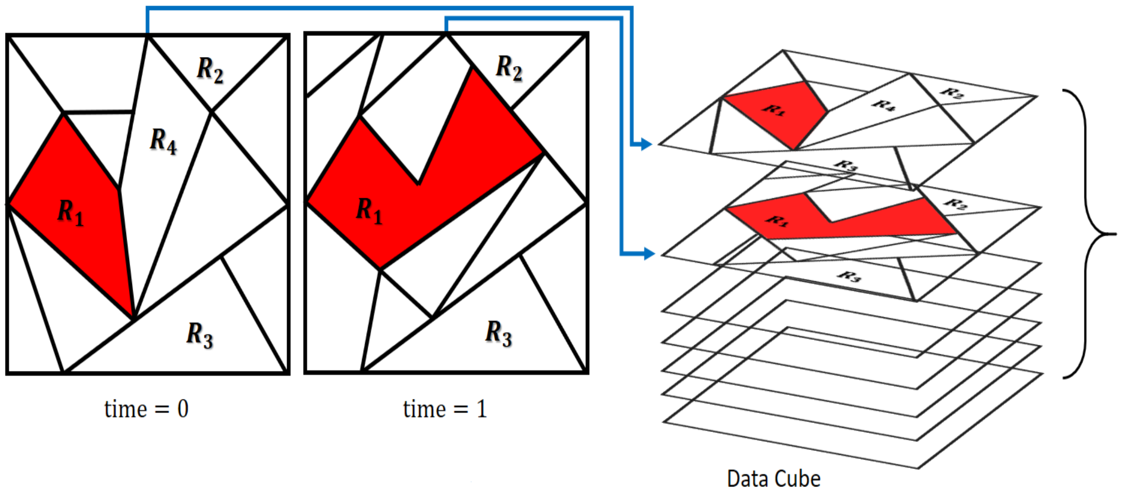

Figure 1.2. Successive images in a “data cube”. Regions of interest may change shape or disappear in between two images.

1.2.1 Clinical Data Used in the Dissertation

Most of the algorithms developed in this dissertation have been tested on pelvic floor MR images of real patients. These images have been obtained at the University of Texas Southwestern Medical Center, fully de-identified according to the requirements of all applicable privacy laws and then transferred to the University of Texas at Arlington as part of a Material Transfer Agreement between two institutions. I would like to thank Dr. Gaurav Khatri and Dr. Philippe Zimmern for making these images available for my research, as well as for their consultations with our research group.

1.2.2 Funding Acknowledgment

Part of my research has been supported by the US National Foundation grant NSF DMS-1616564. During (2016-2020) academic years I was also supported by a

GAANN (Graduate Assistance in Areas of National Need) fellowship funded by the Grant P200A150062 awarded to the Department of Mathematics at the University of Texas at Arlington.

CHAPTER 2

Optical Flow: Inverse Problems

2.1 Introduction

Optical flow is the study of motion within an image. It gives us a way to track and quantify how particles (pixels) change in time with respect to a sequence of images. This then allows for some very interesting applications such as

- image segmentation [8]
- video compression [17]
- motion tracking technology [12]

Helmholtz in his 1925 *Treatise on Physiological Optics* is often credited for starting this discussion of how humans see objects as distinct [9]. In his treatise he describes how small changes in the eyes must aid humans in the depth perception process. Through these small changes, the brain must be capturing information about how close or far away an object is.

And much of the discussion from 1925 onward had been about better articulating this process or about dealing with the problem as a more abstract concept. It is not until the 80's, when computational speed and storage had caught up, do we see explicit algorithms on how to find this motion field. In 1981 two papers are written that show how this motion field can be generated.

In 1981 Horn and Schunck (HS), laid out a framework for Optical Flow from a "first principles" standpoint [10]. Their approach uses a constraint on the brightness patterns and imposes a smoothness condition to arrive at an explicit iterative method to find the motion field. That is, they assume that pixels present in one time instance can be found in another, as long as the amount of time that has passed is very small. Moreover, they require that pixels in small regions of the image domain “move together with similar motion”.

Also in 1981 Lucas and Kanade (LK), tackled the problem in a different way [15]. They assumed that all pixels within a small region move with similar velocities, but solved the problem via a least squares approximation and managed to give another explicit method to find the motion field.

Much of the science after these two works deals with imposing different types of constraints. These constraints usually manifest themselves as changes in the cost functional, or by changing the regularization term.

In this work we are creating a general framework that can then be built upon by future researchers. We first propose a “proof of concept” approach to test whether optical flow can be used in the creation of synthetic data. Having shown results using simple models via the classical Horn and Schunck approach we then use the recovered flow field to solve inverse problems in MRI. We show how high resolution images can be built using what we call an explicit pixel movement structure. We show that using a hybrid mixture term we can approximate slices in between two high resolution images. Our hope is that by showing viability in our methods we can entice succeeding data scientists and image processing engineers to further this preliminary work.

2.2 Definitions, Notations and Formulation of the Problem

We think of an **image** $E(x, y)$ as a scalar valued function defined on a set of points $(x, y) \in \Omega \subseteq \mathbb{R} \times \mathbb{R}$. We call Ω the domain of the image. For every point (sometimes referred to as a **pixel**) in Ω we associate a value E, usually called the brightness or intensity of that pixel. We also view a sequence $\{E_n\}$ as a collection of

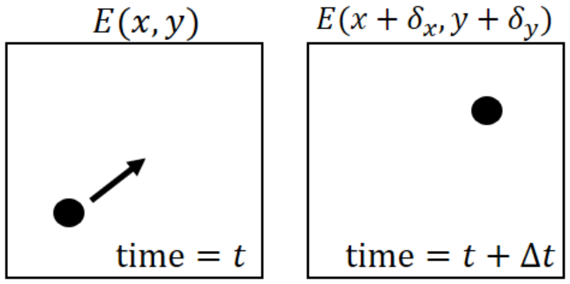

Figure 2.1. The brightness pattern $E(x, y, t)$ in the image frame corresponding to t can be found at location $(x + \delta_x, y + \delta_y)$ at time $t + \Delta t$.

n images, typically called a **video**. To enable us to use the tools of analysis, we will also consider a "continuous version" of that sequence, i.e. when the discrete values of n are instead replaced with a continuous time variable t. The intensity value of pixel (x, y) at moment t is then denoted as $E(x, y, t)$.

For the function $E(x, y, t)$ we will assume that $E(x, y, t)$ is known for all values of (x, y) for all t, and that the derivatives E_x, E_y, E_t exist with $E_x, E_y, E_t \in L^\infty(\overline{\Omega})$. Here $\overline{\Omega}$ is the closure of the image domain $\Omega = \{(x, y) \in (a, b) \times (c, d)\}$. And as a reminder, a function $f \in L^\infty(\overline{\Omega})$ if we can find $C < \infty$ such that $|f| \leq C$ almost everywhere. This space is called the space of **essentially bounded** functions.

Let us assume that our function $E(x, y, t)$ satisfies the following assumption: for any small Δt, one can find functions

$$\delta_x(x, y, t, \Delta t), \;\; \delta_y(x, y, t, \Delta t)$$

such that

$$E(x + \delta_x, y + \delta_y, t + \Delta t) = E(x, y, t). \tag{2.1}$$

That is, we assume that we can find the pixel value E with coordinates (x, y, t) in some other location $(x + \delta_x, y + \delta_y, t + \Delta t)$. This is referred to as the **brightness consistency constraint** (BCC) [1,3,5,10,21]. This ensures that after a small amount of time Δt has passed, we can find the pixel E that was located at point (x, y, t) in some other location $(x + \delta_x, y + \delta_y, t + \Delta t)$. Another requirement is that

$$\lim_{\Delta t \to 0} \delta_x(x, y, t, \Delta t) = 0, \tag{2.2}$$

$$\lim_{\Delta t \to 0} \delta_y(x, y, t, \Delta t) = 0, \tag{2.3}$$

and

$$\lim_{\Delta t \to 0} E(x + \delta_x, y + \delta_y, t + \Delta t) = E(x, y, t), \tag{2.4}$$

Assuming that the following limits exist, we now define the following functions

$$u(x, y, t) := \lim_{\Delta t \to 0} \frac{\delta_x(x, y, t, \Delta t)}{\Delta t} \tag{2.5}$$

$$v(x, y, t) := \lim_{\Delta t \to 0} \frac{\delta_y(x, y, t, \Delta t)}{\Delta t} \tag{2.6}$$

Remark 1. So far we have not assumed that E is smooth, and (2.5-2.6) hold even if $E(x, y, t)$ is not smooth.

The requirements that we have imposed are not too restrictive as to deem their study frivolous. For example, affine transformations for small time increments allow regions of images to stay relatively close to their starting positions. E.g., if we took an image $E(x, y, t)$ and moved all of its pixels in, say a cardinal direction (translation), then every pixel $E(x, y, t)$ can be found in another location $E(x + \delta x, y + \delta y, t + \Delta t)$.

If we enforce an additional condition that $E \in C^1(\Omega)$ we can use Taylor's theorem for multiple variables, and $E(x + \delta_x, y + \delta_y, t + \Delta t)$ can be expressed as

$$E(x + \delta_x, y + \delta_y, t + \Delta t) = E(x, y, t) + \frac{\partial E}{\partial x} \cdot \delta x + \frac{\partial E}{\partial y} \cdot \delta_y + \frac{\partial E}{\partial t} \cdot \Delta t + \text{H.O.T.} \tag{2.7}$$

Here by H.O.T. we mean higher order terms. Now, dividing (2.7) throughout by Δt and then taking $\Delta t \to 0$ we arrive at

$$\frac{\partial E}{\partial x}u + \frac{\partial E}{\partial y}v + \frac{\partial E}{\partial t} = 0, \tag{2.8}$$

which we will often write as

$$E_x u + E_y v + E_t = 0 \tag{2.9}$$

Remark 2. Even though we would like a sort of smoothness for our terms E_x, E_y, and E_t, when actually implementing these ideas in practice we may only have piece-wise smoothness.

We will often refer to the function

$$A(x, y, t) = E_x \cdot u + E_y \cdot v + E_t, \tag{2.10}$$

as the **advection function**, or sometimes the advection term.

The inverse problem associated with this optical flow construction can then be cast as follows: Given $E(x, y, t)$ (and hence knowing the terms E_x, E_y, E_t in (2.9)), find the associated $u(x, y, t), v(x, y, t)$ that satisfy (2.9).

Aside from the assumptions above we also assume that our unknowns $u, v, \nabla u, \nabla v \in L^2(\overline{\Omega})$ and that both $u = v = 0$ on $\partial\Omega$. That is, we are assuming that $u, v \in H_0^1$.

Again, to remind the reader, an L^p space is defined as the space of functions along with a measure space $(\Omega, \sum, \mu)$ whose norm as defined below is finite

$$\|f\|_p := \left(\int_\Omega |f|^p d\mu\right)^{1/p} < \infty.$$

The Sobolev space $W^{1,2}(\Omega)$ is defined as the space of all functions f on a domain S such that

$$W^{1,2}(\Omega) = \{f \in L^2(\Omega) \, : \, D^\alpha(f) \in L^2(\Omega) \, , \, |\alpha| = 1\},$$

where α is an index term. The Sobolev space $W^{1,2}$ is usually denoted as H^1, and an important subspace of H^1 is the space H_0^1 which is the closure of infinitely differentiable functions compactly supported in Ω in H^1.

2.2.1 Optimization

The main difficulty at this point is that (2.9) has no unique solution [2]. For example, suppose we have a constant image (an image that is colored by any one color throughout its domain) $E(x, y, t) = c$ for all $(x, y) \in \Omega$ and all values of $t > 0$ (where $c \in \mathbb{R}$). Then we have

$$E(x + \delta_x, y + \delta_y, t + \Delta t) = E(x, y, t) = c \qquad \forall (x, y) \in \Omega.$$

Then

$$\frac{\partial E}{\partial x} = \frac{\partial E}{\partial y} = \frac{\partial E}{\partial t} = 0 \qquad \forall (x, y) \in \Omega,$$

thus any functions (u, v) will satisfy

$$E_x u + E_y v + E_t = 0.$$

To make the problem well defined, one would have to impose additional restrictions. Horn and Schunck determined unknown functions (u, v) in (2.9) as minimizers of a global cost function $J(u, v)$ (described below) via regularization [10, 21]. That is, they find u and v in H_0^1 that minimize

$$J(u, v) = \int_a^b \int_c^d ((E_x \cdot u + E_y \cdot v + E_t)^2 + \lambda^2 (\|\nabla u\|^2 + \|\nabla v\|^2) \, dx \, dy. \qquad (2.11)$$

The first term in the integral is the square of the advection term (2.9) and in an ideal scenario should be identically zero. But in reality we will not have the first term be actually zero, so we use it as a penalty term to penalize the model from being away from zero. The second term was added by HS to assure that neighboring points have similar velocities. The term $\lambda > 0$ is a parameter used to control the effect of the 2nd term. Large values of λ are used for smooth motion, and small values of λ is for "jerking" motion.

The function $J(u, v)$ is strictly convex [21] and hence for the corresponding optimization problem, the minimization of $J(u, v)$, has a unique solution [4]. It can be shown that solving the minimization problem (2.11) comes down to solving its corresponding Euler-Lagrange equations (EL) [21]. These are given by

$$\frac{\partial J}{\partial u} = (E_x^2 u + E_x E_y v + E_x E_t) - \lambda^2 \Delta u = 0, \tag{2.12}$$

$$\frac{\partial J}{\partial v} = (E_x E_y u + E_y^2 v + E_y E_t) - \lambda^2 \Delta v = 0. \tag{2.13}$$

where $\Delta := \partial_{xx} + \partial_{yy}$. The expressions in (2.12) and (2.13) are referred to as the **Gateaux derivatives**.

Instead of solving (2.12) and (2.13) analytically we consider the discretized version and make the substitution $\Delta u \approx 5(\overline{u} - u)$ and $\Delta v \approx 5(\overline{v} - v)$. Here $(\overline{u}, \overline{v})$ are localized averages of (u, v) centered at point (x, y). A discussion about the details of this substitution is coming in the next section. Then we solve the discretized coupled EL equations, where every function is evaluated at (x, y, t), which leads to the following system

$$(E_x u + E_y v + E_t)E_x - 5\lambda^2(\overline{u} - u) = 0,$$

$$(E_x u + E_y v + E_t)E_y - 5\lambda^2(\overline{v} - v) = 0.$$

We will soon discuss why the approximations for $\Delta u, \Delta v$ are appropriate. But taking it to be true, we have

$$A \begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} 5\lambda^2 \overline{u} - E_x E_t \\ 5\lambda^2 \overline{v} - E_y E_t \end{bmatrix} \tag{2.14}$$

where

$$A = \begin{bmatrix} E_x^2 + 5\lambda^2 & E_x E_y \\ E_y E_x & E_y^2 + 5\lambda^2 \end{bmatrix}. \tag{2.15}$$

This leads us to

$$\begin{bmatrix} u \\ v \end{bmatrix} = \underbrace{\begin{bmatrix} E_x^2 + 5\lambda^2 & E_x E_y \\ E_y E_x & E_y^2 + 5\lambda^2 \end{bmatrix}^{-1}}_{A^{-1}} \begin{bmatrix} 5\lambda^2 \overline{u} - E_x E_t \\ 5\lambda^2 \overline{v} - E_y E_t \end{bmatrix}. \tag{2.16}$$

The matrix inverse above is well defined since

$$|\det(A)| = |25\lambda^4 + 5\lambda^2(E_x^2 + E_y^2)| > 0.$$

2.2.2 The 5-point Stencil

We now turn our attention to the approximations of the $\Delta u, \Delta v$ from the previous section. Let us consider the following discretization formulas for $u(x, y, t)$ using Taylor's formula (here we assume a spatial discritization of (x, y) and step size of h)

$$\begin{aligned} u(i,j,k) &:= u(x_i, y_j, t_k) \\ u(i+1,j,k) &= u(i,j,k) + h \cdot u_x(i,j,k) + \frac{h^2}{2!} \cdot u_{xx}(i,j,k) + \mathcal{O}(h^3) \\ u(i-1,j,k) &= u(i,j,k) - h \cdot u_x(i,j,k) + \frac{h^2}{2!} \cdot u_{xx}(i,j,k) - \mathcal{O}(h^3) \\ u(i,j+1,k) &= u(i,j,k) + h \cdot u_y(i,j,k) + \frac{h^2}{2!} \cdot u_{yy}(i,j,k) + \mathcal{O}(h^3) \\ u(i,j-1,k) &= u(i,j,k) - h \cdot u_y(i,j,k) + \frac{h^2}{2!} \cdot u_{yy}(i,j,k) - \mathcal{O}(h^3). \end{aligned} \tag{2.17}$$

Remark 3. For the rest of this dissertation we normalize and define $h = 1$, as well as denote $x_i = i$.

Adding together the previous equations and ignoring the higher order error terms, we then have

$$u(i,j,k) + u(i+1,j,k) + u(i-1,j,k) + u(i,j+1,k) + u(i,j-1,k) =$$
$$5 \cdot u(i,j,k) + u_{xx}(i,j,k) + u_{yy}(i,j,k).$$

We define

$$\overline{u} := \frac{1}{5}\left(u(i,j,k) + u(i+1,j,k) + u(i-1,j,k) + u(i,k+1,j) + u(i,j-1,k)\right). \tag{2.18}$$

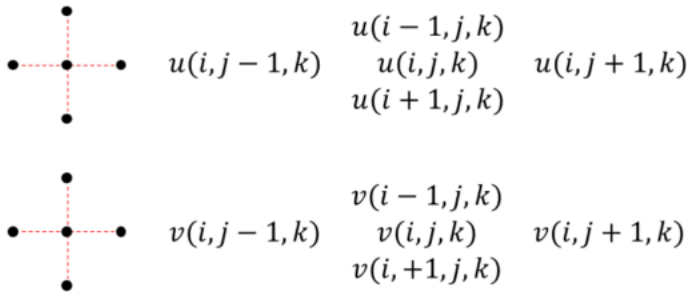

Figure 2.2. The 5-point stencil. We approximate u and v through its "neighboring" points.

Thus we have

$$\overline{u} = u + \frac{1}{5}(u_{xx} + u_{yy})$$

that is, $5(\overline{u} - u) = u_{xx} + u_{yy}$, which leads to

$$\Delta u = \left(\frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}\right) u \approx 5(\overline{u} - u). \tag{2.19}$$

A similar calculation with the function $v(i,j,k)$ would lead to

$$\Delta v = \left(\frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}\right) v \approx 5(\overline{v} - v). \tag{2.20}$$

The approximation that we make for our terms $\Delta u, \Delta v$ are different from the approximations that HS make in their paper [10].

2.3 Solutions and Numerical Scheme

Now back to the matters at hand, we are almost done with the derivation of our numerical scheme to calculate the optical flow. We just need to finish the computation from here

$$\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} E_x^2 + 5\lambda^2 & E_xE_y \\ E_yE_x & E_y^2 + 5\lambda^2 \end{bmatrix}^{-1} \begin{bmatrix} 5\lambda^2\overline{u} - E_xE_t \\ 5\lambda^2\overline{v} - E_yE_t \end{bmatrix}.$$

By direct computation of the right-hand side we have

$$\begin{bmatrix} E_x^2 + 5\lambda^2 & E_xE_y \\ E_yE_x & E_y^2 + 5\lambda^2 \end{bmatrix}^{-1} \begin{bmatrix} 5\lambda^2\overline{u} - E_xE_t \\ 5\lambda^2\overline{v} - E_yE_t \end{bmatrix} =$$

$$= \frac{1}{(E_x^2 + E_y^2)(E_y^2 + 5\lambda^2) - E_x^2E_y^2} \cdot \begin{bmatrix} E_y^2 + 5\lambda^2 & -E_xE_y \\ -E_yE_x & E_x^2 + 5\lambda^2 \end{bmatrix} \cdot \begin{bmatrix} 5\lambda^2\overline{u} - E_xE_t \\ 5\lambda^2\overline{v} - E_yE_t \end{bmatrix}$$

$$= \frac{1}{(E_x^2 + E_y^2)(E_y^2 + 5\lambda^2) - E_x^2E_y^2} \cdot \begin{bmatrix} (E_y^2 + 5\lambda^2)(5\lambda^2\overline{u} - E_xE_t) - E_xE_y(5\lambda^2\overline{v} - E_yE_t) \\ -E_yE_x(5\lambda^2\overline{u} - E_xE_t) + (E_x^2 + 5\lambda^2)(5\lambda^2\overline{v} - E_yE_t) \end{bmatrix}$$

$$= \frac{1}{25\lambda^4 + 5\lambda^2(E_x^2 + E_y^2)} \cdot \begin{bmatrix} 5\lambda^2E_y^2\overline{u} - E_xE_tE_y^2 + 25\lambda^4\overline{u} - 5\lambda^2E_xE_t - 5\lambda^2E_xE_y\overline{v} + E_xE_tE_y^2 \\ -5\lambda^2E_xE_y\overline{u} + E_x^2E_yE_t + 5\lambda^2E_x^2\overline{v} - E_x^2E_yE_t + 25\lambda^4\overline{v} - 5\lambda^2E_yE_t \end{bmatrix}$$

$$= \frac{1}{25\lambda^4 + 5\lambda^2(E_x^2 + E_y^2)} \cdot \begin{bmatrix} 5\lambda^2E_y^2\overline{u} + 25\lambda^4\overline{u} - 5\lambda^2E_xE_t - 5\lambda^2E_xE_y\overline{v} \\ -5\lambda^2E_xE_y\overline{u} + 5\lambda^2E_x^2\overline{v} + 25\lambda^4\overline{v} - 5\lambda^2E_yE_t \end{bmatrix}$$

$$= \frac{1}{5\lambda^2 + E_x^2 + E_y^2} \cdot \begin{bmatrix} E_y^2\overline{u} + 5\lambda^2\overline{u} - E_xE_t - E_xE_y\overline{v} \\ -E_xE_y\overline{u} + E_x^2\overline{v} + 5\lambda^2\overline{v} - E_yE_t \end{bmatrix}.$$

Thus we have

$$\begin{bmatrix} u \\ v \end{bmatrix} = \frac{1}{5\lambda^2 + E_x^2 + E_y^2} \cdot \begin{bmatrix} (5\lambda^2 + E_y^2)\overline{u} - E_xE_y\overline{v} - E_xE_t \\ -E_xE_y\overline{u} + (E_x^2 + 5\lambda^2)\overline{v} - E_yE_t \end{bmatrix},$$

which means

$$(5\lambda^2 + E_x^2 + E_y^2)u = (5\lambda^2 + E_y^2)\overline{u} - E_xE_y\overline{v} - E_xE_t \tag{2.21}$$

$$(5\lambda^2 + E_x^2 + E_y^2)v = -E_xE_y\overline{u} + (E_x^2 + 5\lambda^2)\overline{v} - E_yE_t. \tag{2.22}$$

At this point we have our solutions for (u, v) from the coupled equations above. However, to be consistent with the expressions that Horn and Schunck derived in their paper, we will subtract $-(5\lambda^2 + E_x + E_y^2)\overline{u}$ and $-(5\lambda^2 + E_x^2 + E_y^2)\overline{v}$ from the previous equations. We then end up with

$$\begin{aligned}(5\lambda^2 + E_x^2 + E_y^2)u - (5\lambda^2 + E_x^2 + E_y^2)\overline{u} &= \\ &(5\lambda^2 + E_y^2)\overline{u} - E_xE_y\overline{v} - E_xE_t - (5\lambda^2 + E_x^2 + E_y^2)\overline{u}\end{aligned}$$

$$\begin{aligned}(5\lambda^2 + E_x^2 + E_y^2)v - (5\lambda^2 + E_x^2 + E_y^2)\overline{v} &= \\ &- E_xE_y\overline{u} + (E_x^2 + 5\lambda^2)\overline{v} - E_yE_t - (5\lambda^2 + E_x^2 + E_y^2)\overline{v}\end{aligned}$$

which can be written

$$(5\lambda^2 + E_x^2 + E_y^2)(u - \overline{u}) = -E_x^2\overline{u} - E_xE_y\overline{v} - E_xE_t$$

$$(5\lambda^2 + E_x^2 + E_y^2)(v - \overline{v}) = -E_xE_y\overline{u} - E_y^2\overline{v} - E_yE_t.$$

This means

$$u - \overline{u} = \frac{-E_x(E_x\overline{u} + E_y\overline{v} + E_t)}{5\lambda^2 + E_x^2 + E_y^2},$$

$$v - \overline{v} = \frac{-E_y(E_x\overline{u} + E_y\overline{v} + E_t)}{5\lambda^2 + E_x^2 + E_y^2},$$

and we finally arrive at

$$u = \overline{u} - \frac{E_x(E_x\overline{u} + E_y\overline{v} + E_t)}{5\lambda^2 + E_x^2 + E_y^2}, \tag{2.23}$$

$$v = \overline{v} - \frac{E_y(E_x\overline{u} + E_y\overline{v} + E_t)}{5\lambda^2 + E_x^2 + E_y^2}. \tag{2.24}$$

At this point we have the solutions that we sought via (2.23) and (2.24). From here the HS-method solves (2.23) and (2.24) via a Gauss-Seidel iterative scheme [10]

$$u^{n+1} = \overline{u}^n - \frac{E_x(E_x\overline{u}^n + E_y\overline{v}^n + E_t)}{5\lambda^2 + E_x^2 + E_y^2}, \tag{2.25}$$

$$v^{n+1} = \overline{v}^n - \frac{E_y(E_x\overline{u}^n + E_y\overline{v}^n + E_t)}{5\lambda^2 + E_x^2 + E_y^2}, \tag{2.26}$$

In their iterative scheme HS compute the next estimates for the flow field (u^{n+1}, v^{n+1}) via the previous iterates (u^n, v^n). In practice we usually initialize the values $(\overline{u}^1, \overline{v}^1) = (0, 0)$ and run the algorithm for only a few values of n (typically between 10-15 iterations).

2.4 Verifying the Accuracy of the Recovered Vector Field

So now that we have our velocity vectors (u, v) we know the values that satisfy

$$E_x u + E_y v + E_t = 0.$$

Since we know

$$\frac{\partial E}{\partial t} = \lim_{\Delta t \to 0} \frac{E(x, y, t + \Delta t) - E(x, y, t)}{\Delta t}$$

then for a pair of images E_1, E_2 separated by a time increment of Δt within an image sequence, we can put

$$E_1(x, y, t) = E(x, y, t), \qquad E_2(x, y, t) = E(x, y, t + \Delta t).$$

Then

$$\frac{\partial E}{\partial t} \approx \frac{E_2 - E_1}{\Delta t}. \tag{2.27}$$

Thus we have

$$
\begin{aligned}
& E_x u + E_y v + \frac{E_2 - E_1}{\Delta t} \approx 0 \\
\Rightarrow \qquad & \frac{E_2 - E_1}{\Delta t} \approx -(E_x u + E_y v) \\
\Rightarrow \qquad & E_2 - E_1 \approx -\Delta t (E_x u + E_y v)
\end{aligned}
\tag{2.28}
$$

We obtain

$$E_2 \approx E_1 - \Delta t (E_x u + E_y v). \tag{2.29}$$

Using (2.29) we can check whether our numerical scheme is being computed correctly, since we happen to know beforehand what the actual image 2 was supposed to be. We will also denote the quantity on the right hand side of (2.29) as

$$\tilde{E}_2 := E_1 - \Delta t (E_x u + E_y v). \tag{2.30}$$

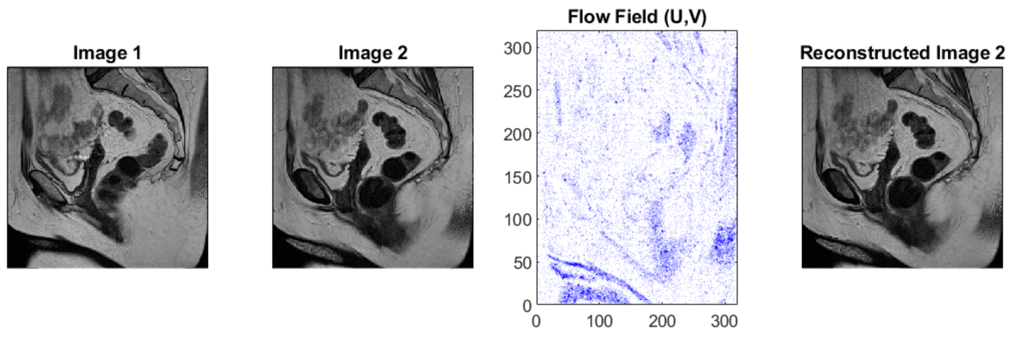

Figure 2.3. (Left) Original Image 1. (Middle Left) Original Image 2. (Middle Right) Flow field generated. (Right) A reconstructed version of Image 2 via (2.29).

In the figure above we run our modified HS algorithm with the parameters listed in the table below. These parameters are proportional to the actual resolutions in the MRI used to obtain the images.

δ_x	.6875 mm
δ_y	.6875 mm
Δt	4.4 mm
Iterations	5
λ	10^{-7}

With the setup prescribed as above we are able to achieve a relative error as

$$\text{Relative Error} = \frac{\|\tilde{E}_2 - E_2\|_p}{\|E_2\|_p} = \begin{cases} = .0335 & \text{if } p = 2 \\ = .0450 & \text{if } p = F \end{cases}$$

Here $\|\cdot\|_2$ is the induced matrix norm and $\|\cdot\|_F$ denotes the Frobenious norm. Just to remind the reader the 2-norm and the Frobenius norm are calculated as

$$\|A\|_2 = \sqrt{\lambda_{max}(A^*A)}$$
$$\|A\|_F = \left(\sum_{i=i}^{m}\sum_{j=1}^{n}|a_{i,j}|^2\right)^{1/2}$$

where $\lambda_{max}(\cdot)$ denotes the largest eigenvalue of a matrix.

CHAPTER 3

Optical Flow: Modified Forward Propagation

So far we have shown that the optical flow algorithm provided by Horn and Schunk can produce an accurate vector field between two images. Through reconstruction of the second image (using the first one together with the flow field) we can verify how well the algorithm is performing. This is a benchmark for the rest of this dissertation because what we discuss in Chapter 3 and 4 is how we can use this information in creative ways. More specifically, we have two goals in mind. Here we present the goals of the following chapters and their associated sections for the reader.

1. Creation of new information from the computed flow field.
    (a) Section 3.1: Transformations of (u, v).
    (b) Section 3.2: Localization of Change.
    (c) Section 3.3: Analysis of $\|E_x u + E_y v + E_t\|$.
2. Improving the MRI resolution by utilizing the flow field.
    (a) Section 4.1: Deficiencies in Blind Interpolation Processes.
    (b) Section 4.2: The Explicit Pixel Movement Framework.
    (c) Section 4.3: Explicit Movement in MRI.
    (d) Section 4.4: Methods of Forward Propagation.
    (e) Section 4.5: Area Preserving Transformations.

For our first goal we would like to use the computed flow field to generate new data. So in Section 3.1 we discuss how we may slightly modify our flow field to this end. In Section 3.2 we discuss how we can localize this process. Then in Section 3.3

we talk about the need for adding noise into synthetic images to make sure that the brightness consistency constraint is satisfied.

As for our second goal, we would like an automated process that will create high resolution images between two given images. Sections 4.1, 4.2, 4.3, 4.4 and 4.5 lead the reader trough the creation of what we ultimately call the Forward Hybrid Propagation Model, which allows us to achieve high quality images to be used to "fill in" the gap between two images. We discuss how simpler linear methods fail, why HS is not enough, how to overcome technical challenges, and how to "mix" new information into data frames.

3.1 Transformations of u and v

In the reconstruction equation $\tilde{E}_2 = E_1 - \Delta t(E_x \cdot u + E_y \cdot v)$ we assume the knowledge of accurate estimates for (u, v) via the algorithm of Horn and Schunck. One could also use one of the other variants of the optical flow problem to solve for (u, v), however different approaches make different assumptions [4,21]. Regardless of how the components are computed, a natural question to ask is what type of transformations on u, v are interesting from a data generation standpoint. Considering this, the first type of transformation that we employed was the following

$$u \to \varepsilon u, \qquad v \to \varepsilon v, \tag{3.1}$$

where $\varepsilon \in [0, 1]$. This seems like a natural choice because if $\varepsilon = 0$ we have

$$u \to 0, \qquad v \to 0,$$

i.e. there is no movement in the scene. This makes complete sense because in our equation it is equivalent to

$$\tilde{E}_2 = E_1 - \Delta t(E_x \cdot 0 + E_y \cdot 0) = E_1.$$

On the other hand if $\varepsilon = 1$ we have

$$u \to u, \qquad v \to v,$$

which means that there is no change in your velocity functions, so the scene plays out as usual. Analytically, if we change $(u, v) \to (\varepsilon u, \varepsilon v)$ we have

$$\begin{aligned}\tilde{E}_2 &= E_1 - \Delta t(E_x \cdot u + E_y \cdot v)\\ &= E_1 - \Delta t(\varepsilon E_x u + \varepsilon E_y v)\\ &= E_1 - \varepsilon \Delta t(E_x \cdot u + E_y \cdot v)\end{aligned}$$

We note that $E_1 - E_2$ can be computed as

$$E_2 - E_1 = -\Delta t(E_x \cdot u + E_y \cdot v).$$

So we have

$$\begin{aligned}\tilde{E}_2 &= E_1 + \varepsilon(E_2 - E_1),\\ &= \varepsilon E_2 + (1 - \varepsilon)E_1.\end{aligned}$$

Thus we may view the function $F(u, v) = (\varepsilon u, \varepsilon v)$ as a homotopy from E_1 to E_2 as ε goes from 0 to 1. In fact, when $\varepsilon = 1/2$ we have

$$\tilde{E}_2 = \frac{E_1 + E_2}{2}.$$

For different values of ε we can get different images as shown here

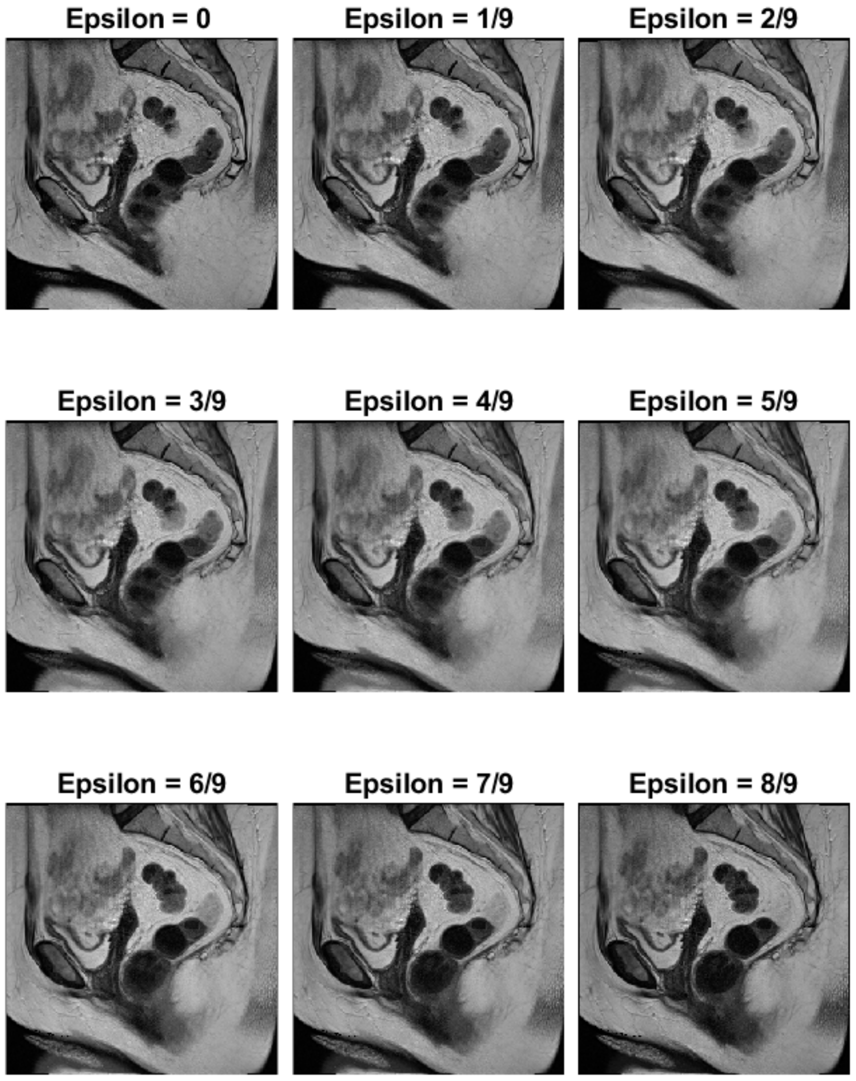

Figure 3.1. At the top left we start with a value of $\varepsilon = 0$ and continue increasing in a serpentine pattern.

In this case we are starting at an image E_1 and evolving our ε in

$$\tilde{E}_2 = E_1 - \Delta t(\varepsilon E_x \cdot u + \varepsilon E_y \cdot v)$$

from $\varepsilon = 0$ until $\varepsilon = 1$ which makes $\tilde{E}_2$ close to E_2. These images are taken from an MRI scan of a patient in sagittal view. For reference, images 1 and 2 are taken directly one after another in the same orientation.

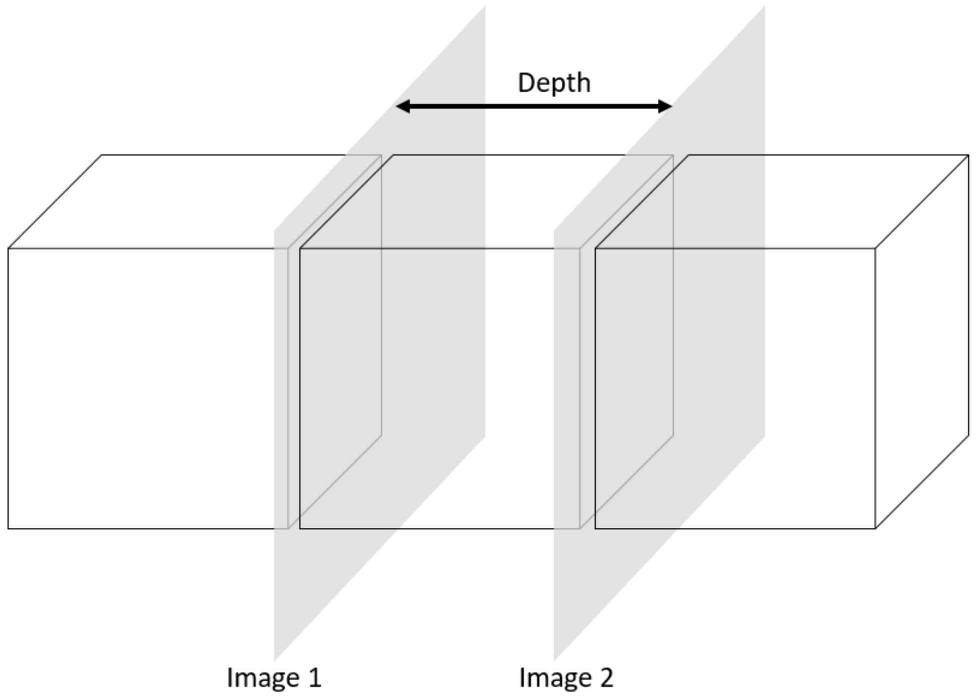

Figure 3.2. Image pairs from a sequence in an MRI scan tell information about a cross-section of a volume after a change in depth.

3.2 Localization of Change

Now that we have shown that the transformations $u \to \varepsilon u$ and $v \to \varepsilon v$ will produce good results, we explore what will happen if the changes were not made globally. That is, what if we could localize changes in the components u and v so that only a portion of the image changes? This would give us freedom to localize changes, instead of having global variation.

With this idea in mind we would like to choose a transformation scheme that will affect our components $u(x, y)$ and $v(x, y)$ smoothly around a point $(x, y) \in \Omega$. To this end, we employ a Gaussian centered at (h, k)

$$f(x, y) = e^{-\sigma^2 \cdot \left((x-h)^2 + (y-k)^2\right)}, \tag{3.2}$$

which will determine the support of the change. The points (h, k) can be chosen randomly or can be predetermined beforehand. In our numerical experiment we picked $(h, k) \in U\{(x, y) \in \mathbb{N} \times \mathbb{N} | \ \ 1 \leq x \leq n \ \text{ and } \ 1 \leq y \leq m\}$, where $U(B)$ is a uniform distribution of set B. We say points, plural, because in practice we would like to generate many different images so building an algorithm with this in mind is more effective than doing things one at a time. Therefore, we are actually talking about a family of Gaussians

$$f_s(x, y) = e^{-\sigma_s^2 \cdot \left((x-h_s)^2 + (y-k_s)^2\right)}, \tag{3.3}$$

where $s \in \{1, 2, 3, ..., K_{\text{end}}\}$ and K_{end} is the number of locations that one would like the algorithm to modify. This would allow different regions to be affected by the change. The **variance** σ_s^2 can likewise be randomly chosen per each trial, or it can be predetermined. The variation should be large enough to affect a non-trivial region around (h_s, k_s), but not so large as to affect the entire image.

```
Initialize:
$K_{\text{end}}, \sigma_s^2, E_1, E_x, E_y, u, v, \Delta t$
For $s \in \{1, \dots, K_{\text{end}}\}$
  Pick
  $(h,k) \in U\{(x,y) | \, 1 \le x \le n$ and $1 \le y \le m\}$
  $f_s(x,y) = e^{-\sigma_s^2[(x-h_s)^2+(y-k_s)^2]}$
  $u(i,j) = f_s \cdot u(i,j)$
  $v(i,j) = f_s \cdot v(i,j)$
  $\tilde{E}_2 = E_1 - \Delta t \cdot (E_x u + E_y v)$
end
```

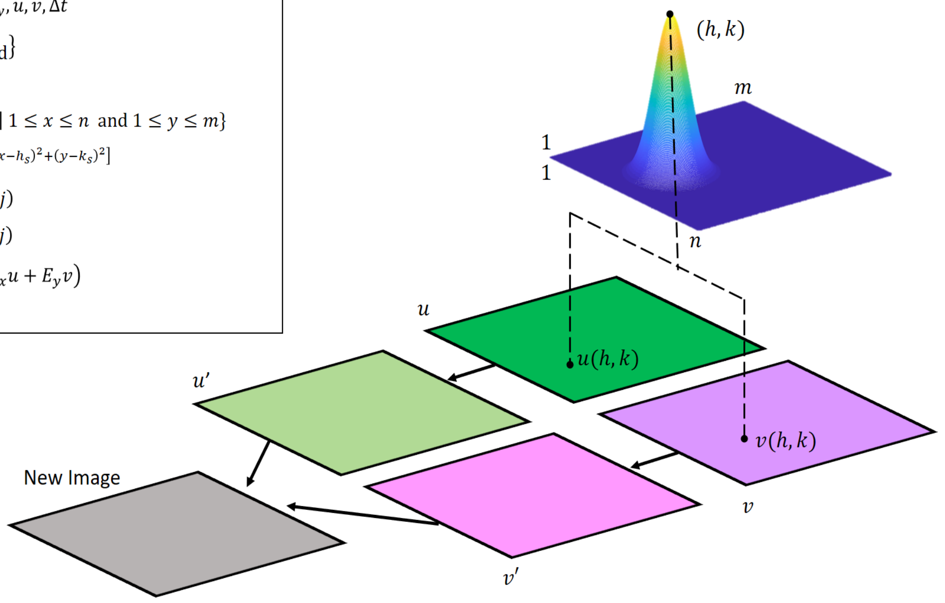

Figure 3.3. Using a localized approach allows us to only change a subset of pixels within our image.

3.3 Analysis of $\|E_x u + E_y v + E_t\|$

We now turn our attention to the advection term

$$A(x,y,t) = E_x \cdot u + E_y \cdot v + E_t.$$

In an ideal scenario we would have $A(x,y,t)$ be zero. This would determine how “faithful” we are to satisfying our brightness consistency constraint

$$E(x+\delta_x, y+\delta_y, t+\Delta t) = E(x,y,t).$$

But because of errors that accumulate via numerical implementation, non-optimal choices of λ^2, and other various violations in assumptions, in practice this will not usually be the case. However, the advection equation

$$A(x,y,t) = 0, \tag{3.4}$$

is still useful, because it will tell you whether or not the reconstruction was successful. Since we may always have some sort of imperfections in our implementation we would expect $E_x u + E_y v + E_t = \eta$, where η is a “small” amount of error. The size of η is an indicator of the quality of the reconstruction (the smaller it is, the better is the quality).

We now introduce a simple phantom to illustrate how things can be “evaluated” by the advection term and also to show how noise has an effect on our advection term. To this end, we take a black and white image of a circle (which was generated in MATLAB) and move it 1 pixel in the north-east direction. To familiarize the reader with the layout of the upcoming experiments and tables, we will first show the 1 pixel movement, a 25 pixel movement, and a 50 pixel movement. We start out our discussion with no noise term added.

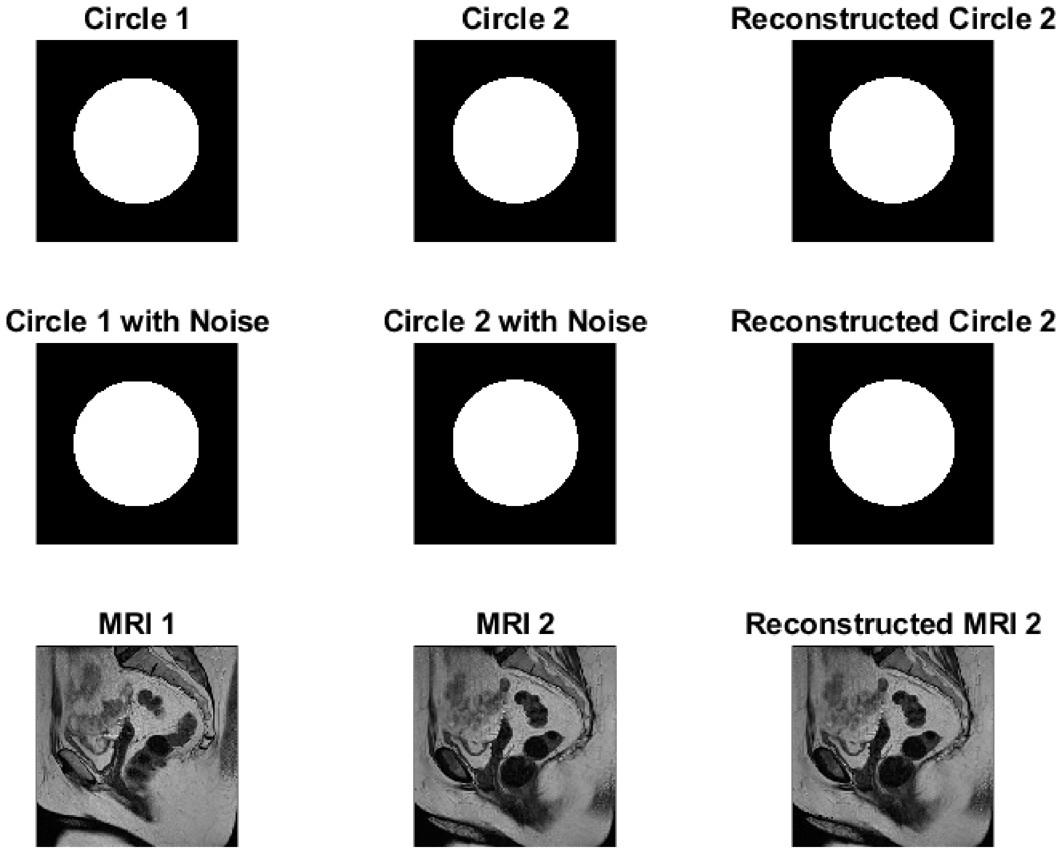

Figure 3.4. (Top-left) Original circle. (Top-middle) Original Circle 2. (Top-right) Reconstructed Circle 2. (Middle-left) Original circle w/noise = 0 . (Middle-middle) Original circle 2 w/noise = 0. (Middle-right) Reconstructed Circle 2. (Bottom-left) MRI 1. (Bottom-middle) MRI 2. (Bottom-right) Reconstructed MRI 2.

With a 1 pixel movement it is really hard to see any difference between the images. We can however keep track of norms of different terms and use that instead of visual inspection to emphasize differences. To this end, we will introduce the following norms

$$\text{NormExEy} = \sqrt{\|E_x\|^2 + \|E_y\|^2} \tag{3.5}$$

$$\text{NormEt} = \|E_t\| \tag{3.6}$$

$$\text{NormAdvec} = \|E_x \cdot u + E_y \cdot v + E_t\| \tag{3.7}$$

$$\text{NormDiff} = \|\tilde{E}_2 - E_2\| \tag{3.8}$$

$$\text{NormRel} = \|\tilde{E}_2 - E_2\| / \|E_2\| \tag{3.9}$$

The term NormExEy tells us the magnitude of overall “variation” in an image. The second term NormEt gives us an idea of how “different” image 1 and image 2 are. The third term, NormAvec, tells us how far away we are from satisfying the BCC. The fourth term, NormDiff, is the difference between the reconstructed image 2 and the actual image 2. The reconstructions in the last column of Figure (3.4), rows 1 and 2, are being compared to the image in the first row second column. The last term, NormRel, is the relative error of the reconstructed image 2 and the actual image 2.

Here is the table with all of these values from Figure (3.4). Note that the first two numerical columns in Table (3.1) are the same, this is done on purpose to setup a “table template” that will be used many times throughout this section. As we progress through this section we will a noise term which will cause the middle column to change, so the choice to add a noise = 0 column is purposeful. The hope is that this aids the reader with understanding the format of subsequent tables in this section.

	Circles	Circles w/noise = 0	MRI
NormExEy	21.5407	21.5407	10.9188
NormEt	7.5515	7.5515	5.7264
NormAdvec	7.5169	7.5169	0.2514
NormDiff	7.5169	7.5169	3.4232
NormRel	0.0447	0.0447	0.0283

Table 3.1. Norm data from Figure (3.4)

The main thing to take away from this table is that the advection error is large, even in this simple case. All we have done is moved our circle a small amount. Still our advection error is large. The MRI pairs on the other hand have a relatively small advection error, and the reconstruction is doing well.

Before we continue, we can actually explain why the terms NormAdvec and NormDiff are very close to each other in the case of circles without noise.

Let us assume that

$$\text{NormAdvec} = \|E_x \cdot u + E_y \cdot v + E_t\| = \eta_1,$$

$$\text{NormDiff} = \|\tilde{E}_2 - E_2\| = \eta_2.$$

In the case where we have no noise term (either artificially added or coming naturally from the scene) with the chosen images of circles we have that $E_x = E_y = 0$ almost everywhere. This is because we are using piece-wise constant images. This means that

$$\text{NormAdvec} = \|E_x \cdot u + E_y \cdot v + E_t\| = \left\|\frac{E_2 - E_1}{\Delta t}\right\| = \eta_1,$$

and using $\tilde{E}_2 = E_1 - \Delta t \cdot (E_x u + E_y v)$,

$$\text{NormDiff} = \|\tilde{E}_2 - E_2\| = \|E_1 - E_2\| = \|E_2 - E_1\| = \eta_2.$$

Since we use $\Delta t = 1$, we arrive at

$$\eta_1 = \|E_2 - E_1\| = \eta_2$$

When we add artificial noise to our images (or if our images come with their own noise, as is the case with the MRIs) we actually make the derivatives to be non-zero, which then translates to our errors $\eta_1 \neq \eta_2$.

Next we will add a small amount of noise (uniformly additive 1% noise) to the circles and then show the same table as above, but with updated values.

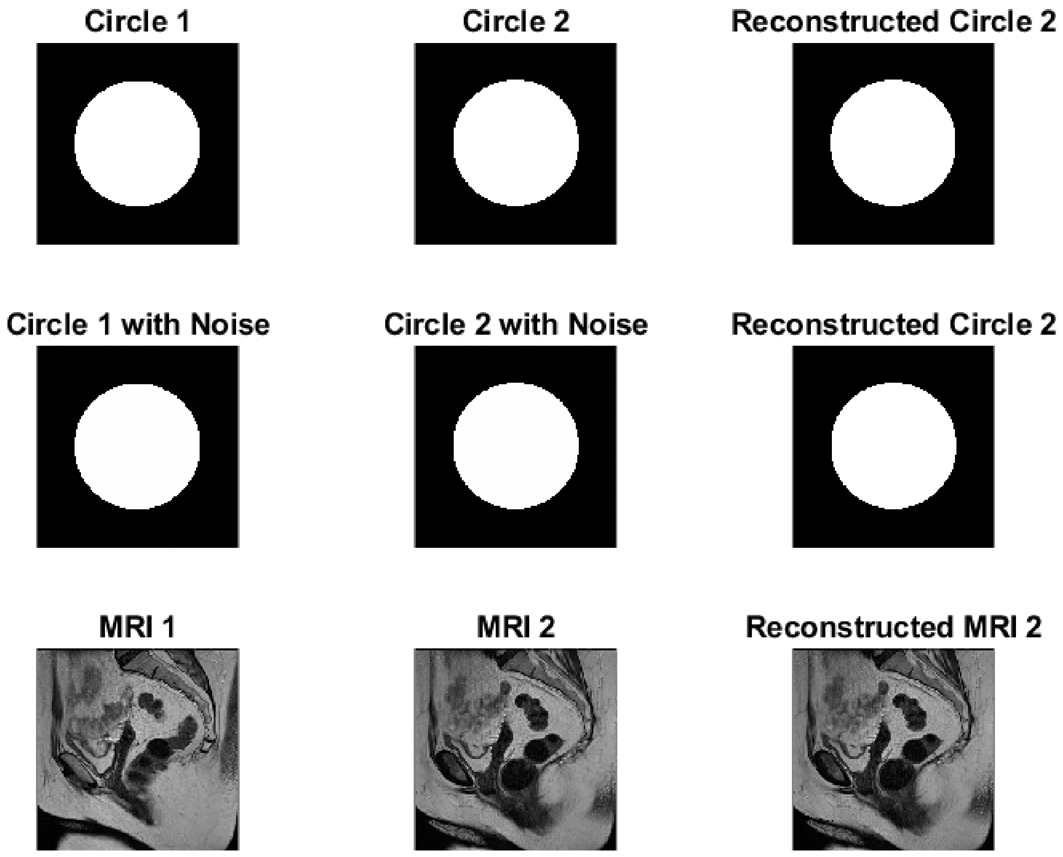

Figure 3.5. (Top-left) Original circle. (Top-middle) Original Circle 2. (Top-right) Reconstructed Circle 2. (Middle-left) Original circle w/noise = 1%. (Middle-middle) Original circle 2 w/noise=1%. (Middle-right) Reconstructed Circle 2. (Bottom-left) MRI 1. (Bottom-middle) MRI 2. (Bottom-right) Reconstructed MRI 2.

Figure (3.5) and Figure (3.4) look practically the same. The amount of noise being added to the circles is 1% of uniformly random noise. This is because our movement is very small and the noise term that is being used is also very small. This results in images that are very close to their denoised partners visually. However the effects can be seen in the following table (3.2).

The only column that has changed is the 2nd column. The NormExEy term has changed because now there are more spatial changes in our images. The second term

	Circles	Circles w/noise = 0.01	MRI
NormExEy	21.5407	21.5472	10.9188
NormEt	7.5515	7.5515	5.7264
NormAdvec	7.5169	3.1224e-04	0.2514
NormDiff	7.5169	1.6009	3.4232
NormRel	0.0447	1.8583e-06	0.0283

Table 3.2. Norm data from Figure (3.5)

NormEt has not changed because the same noise term is added to circle 1 and circle 2. But our advection term, NormAdvec, has gone down substantially. This means that our BCC is almost perfectly being satisfied. The difference between the reconstructed circle 2 and the original circle 2 is smaller, but the relative error between the two, NormRel, has gone down a lot.

Before we move our circles by a larger amount, we will stay with this example and add more noise. We will spare the image sequences and instead give a table of the results, again these are all with a 1 pixel movement in the top-right direction.

Circles	Noise = 0	Noise = 0.01	Noise = 0.05	Noise = 0.10
NormExEy	21.5407	21.5472	21.6301	21.8394
NormEt	7.5515	7.5515	7.5515	7.5515
NormAdvec	7.5169	3.1224e-04	9.2171e-05	2.6838e-06
NormDiff	7.5169	1.6009	8.0166	16.0075
NormRel	0.0447	1.8583e-06	5.4855e-07	1.5452e-08

Table 3.3. Norm data from Figure (3.5), but with three different noise terms

We see that adding more noise to an image has many different effects on the norm terms. Fist of all, it translates to more variation in our simple image and therefore our NormExEy term grows. The NormEt term is not changed throughout this process because the same noise term is being added to circle 1 and circle 2. If we were to add different noise terms then this term would change.

We will now move the circles 25 pixels in the top-right direction, and we will add a 5% noise term to aid in the reconstruction. We show the results of the images first and then the table of the relevant data.

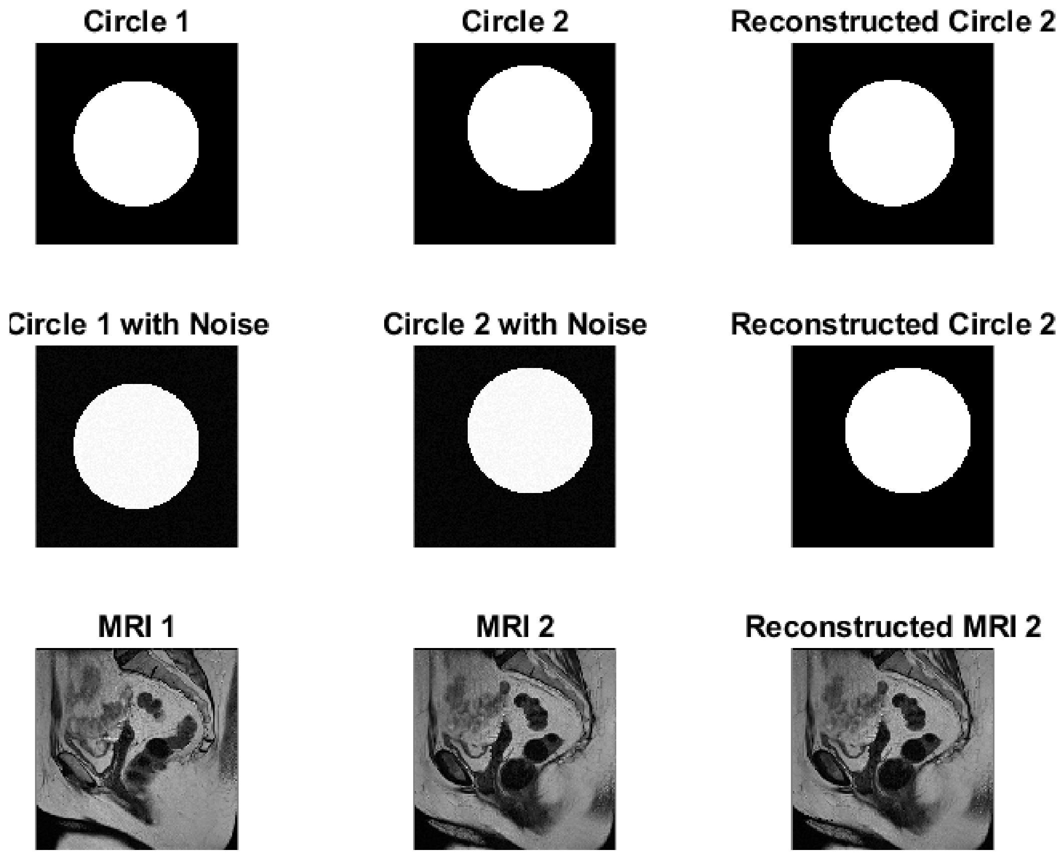

Figure 3.6. (Top-left) Original circle. (Top-middle) Original Circle 2. (Top-right) Reconstructed Circle 2. (Middle-left) Original circle w/noise=5%. (Middle-middle) Original circle 2 w/noise=5%. (Middle-right) Reconstructed Circle 2. (Bottom-left) MRI 1. (Bottom-middle) MRI 2. (Bottom-right) Reconstructed MRI 2.

Lastly we move our images 50 pixels and put the noise level at 5%.

	Circles	Circles w/noise = 0.05	MRI
NormExEy	21.5407	21.5785	10.9188
NormEt	66.4386	66.4386	5.7264
NormAdvec	66.1581	7.3121e-04	0.2514
NormDiff	66.1581	8.0188	3.4232
NormRel	0.3937	4.3518e-06	0.0283

Table 3.4. Norm data using from Figure (3.6) with 5% noise

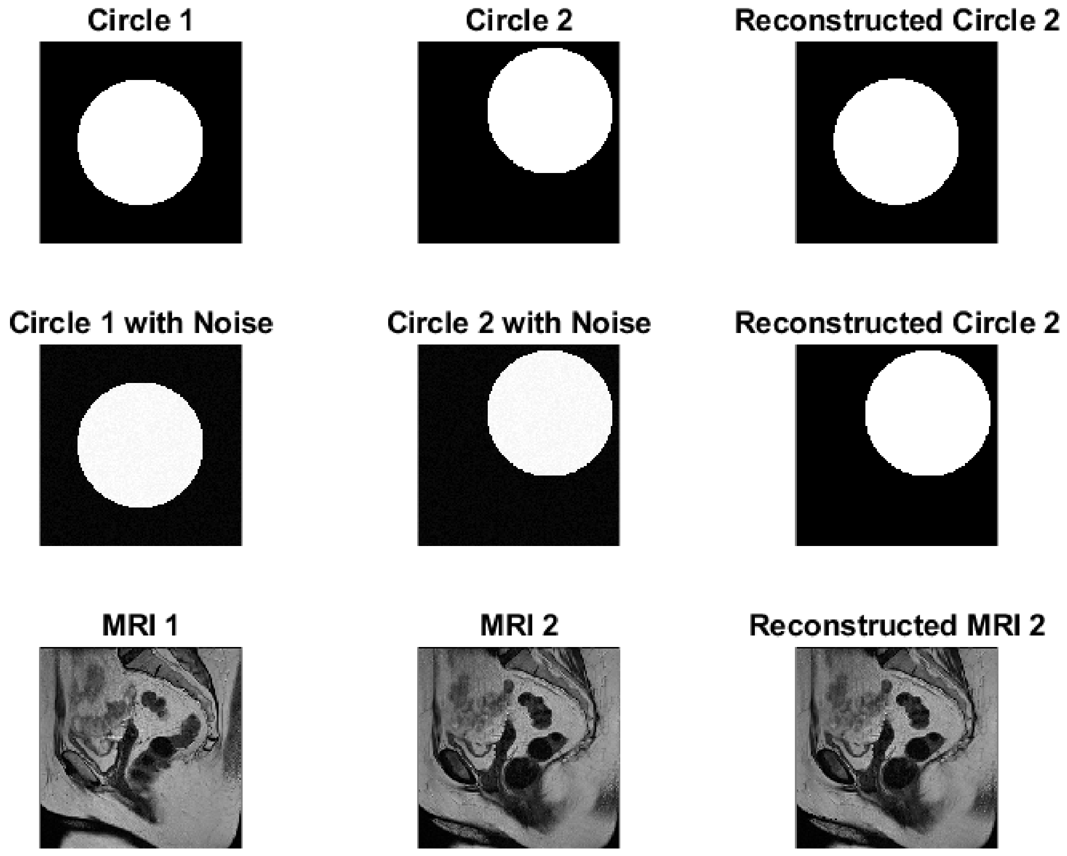

Figure 3.7. (Top-left) Original circle. (Top-middle) Original Circle 2. (Top-right) Reconstructed Circle 2. (Middle-left) Original circle w/noise=5%. (Middle-middle) Original circle 2 w/noise=5%. (Middle-right) Reconstructed Circle 2. (Bottom-left) MRI 1. (Bottom-middle) MRI 2. (Bottom-right) Reconstructed MRI 2.

	Circles	Circles w/noise = 0.05	MRI
NormExEy	21.5407	21.5794	10.9188
NormEt	104.2776	104.2776	5.7264
NormAdvec	104.0771	0.0093	0.2514
NormDiff	104.0771	8.0080	3.4232
NormRel	0.6194	5.5326e-05	0.0283

Table 3.5. Norm data from Figure (3.7)

3.3.1 Discussion of Results

We can see that without noise the reconstruction is not good in the case of circular phantoms. This is because our reconstruction method relies on the fact that $E_x \cdot u + E_y \cdot v + E_t = 0$. For the images we are trying to work with (piecewise constant images, 1 in the support and 0 otherwise) the spatial derivatives E_x and E_y would be zero almost everywhere, which then eliminate u and v in (2.30) used for forward propagation. This then causes our $\tilde{E}_2 = E_1$, which is not what we are trying to do. So the addition of noise is an important pre-processing step, when dealing with simple phantoms. It is not needed in the case of the MRIs, for example, because with those images they naturally come with 1) texture from the different parts of the anatomy and 2) noise from their acquisition.

We are also capable of reconstructing the image (the piece-wise constant circles) without noise by subtracting it out at the end of our algorithm. Namely, let us assume the same noise term (say φ) is being added to the images $W_1 = E_1 + \varphi$ and $W_2 = E_2 + \varphi$. One can then run our algorithm on the noised images W_1 and W_2 and once complete, simply put $\tilde{E}_2 := \tilde{E}_2 - \varphi$. In other words, if the original images do not have enough spatial variation, one can artificially add certain amount of noise to make the algorithm applicable. Then, once the new images are generated, the artificially added noise can be subtracted.

One more useful piece of information that is given by the tables in this section is the fact that knowledge of the norms NormExEy and NormEt does not provide enough information to tell before computing the components u and v whether the BCC is satisfied or not (e.g. see Table 3.3 on page 30). That is, if all you were given from an image sequence was the first two rows of the norms table, it is impossible to know which image pairs satisfy the BCC and which do not.

CHAPTER 4

Approximation of Intermediate Slices in MRI

We now turn our attention to the need of improving the resolution between the slices and how the techniques developed so far can be modified to achieve this goal. We first begin with a discussion concerning why standard interpolation techniques will not work for our purposes. Then we briefly discuss why the HS formulation is not enough to solve the problem alone. Next we discuss a way to add motion into a scene via Pixel Movement. And finally we discuss the Hybrid Methods that incorporate several processes working together to arrive at the completion of our second goal.

4.1 Deficiencies in Blind Interpolation Processes

Let us explore why a simple interpolation does not produce the types of images that we would like. That is, given two image slices we would like to produce an approximation for the intermediate slices between the given two.

Suppose we are given the two images in Figure 4.1. What we have done is taken a circular mass and moved it in the north-western direction from one slice to another. What we would like for the “approximation of the images in the intermediate slices” is the singular mass shifted by a progressively increasing fraction of the distance between Image 1 and Image 2.

To test whether a straightforward interpolation will suffice, we can use, for example, MATLAB’s built-in interp3 command to try and get intermediate slices. But this produces blurred images for the intermediate slices and fails to give the type of successive images that we would like to have. This is not a deficiency in MATLAB, but rather a lack of proper association between points in one image and points in another. We discuss this in more detail in the text that follows.

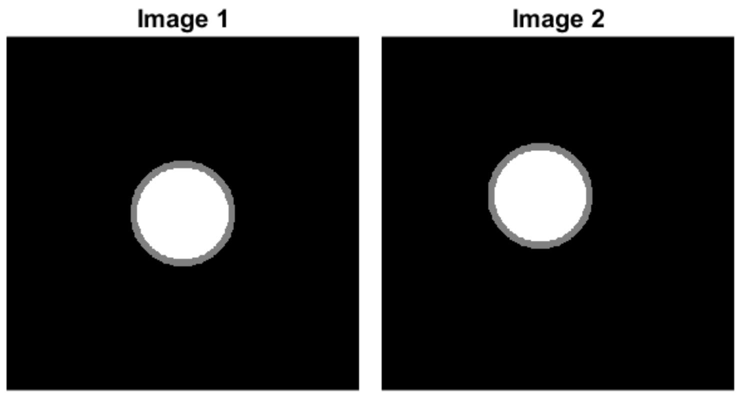

Figure 4.1. Take a circle and move it in the north-western direction.

In the previous figure, we have taken a circle and moved it in the north-western direction. We put these images into a volume at depth $z = 1$ for Image 1 and $z = 2$ for Image 2. We used interp3 to get the intermediate depths between $z = 1$ and $z = 2$, but what comes out are images of this type (see next figure).

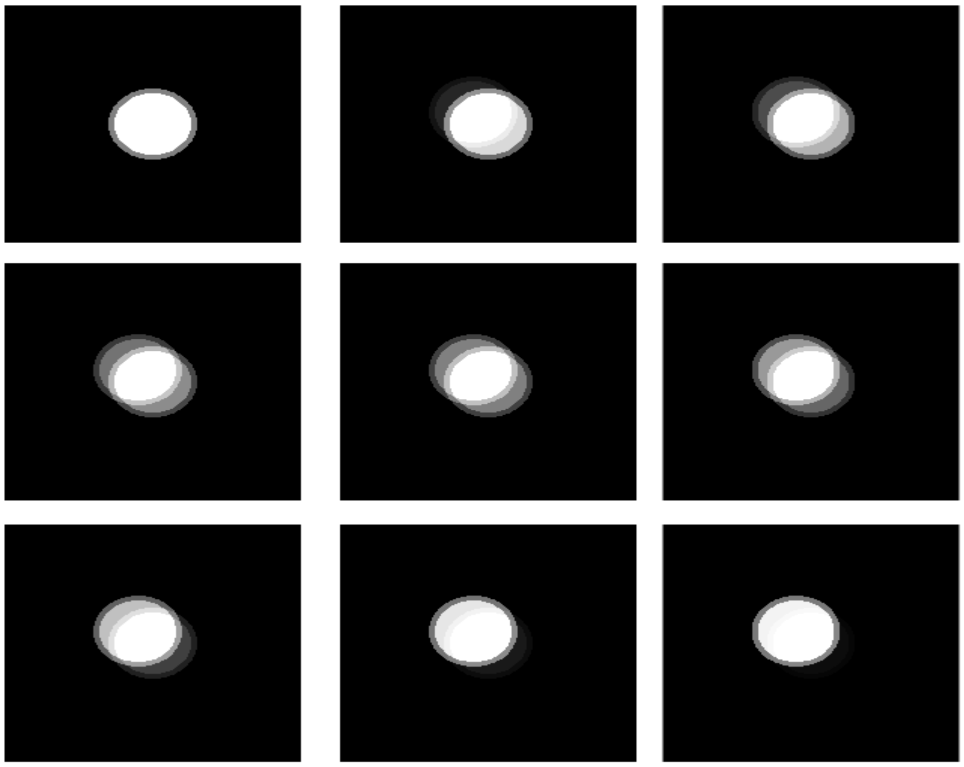

Figure 4.2. Interpolated images from the previous given two. We see that the intermediate slices have an additive blur from one image to another. (Top left) $z = 1$, (Bottom Right) $z = 2$. To be read from top left, to top middle, to top right, then middle left, middle middle, etc..

Intuitively, if we were to couple each pixel in Image 1 with a unique “corresponding” shifted pixel in Image 2, then a simple linear interpolation would provide a “perfect” approximation of the shifted images in the intermediate slices between the two images. But that type of information about pairwise correspondence of points is certainly not available. Instead, the interpolation process tries to approximate the values of the function in the new slices using correspondence between entire regions (supports of the discs) and the functional values there. The result is the “blurry” approximation that is not adequate for our needs.

Let us now discuss why the (HS) formulation is not enough either. To give you an idea of what happens if we simply try and use (HS) formulation alone we offer the following figure.

$$E_2 = E_1 - \Delta t \underbrace{\left(E_x u + E_y v\right)}_{A}$$

$$E_2 = E_1 - \Delta t A$$

$$\Delta t A = E_1 - E_2$$

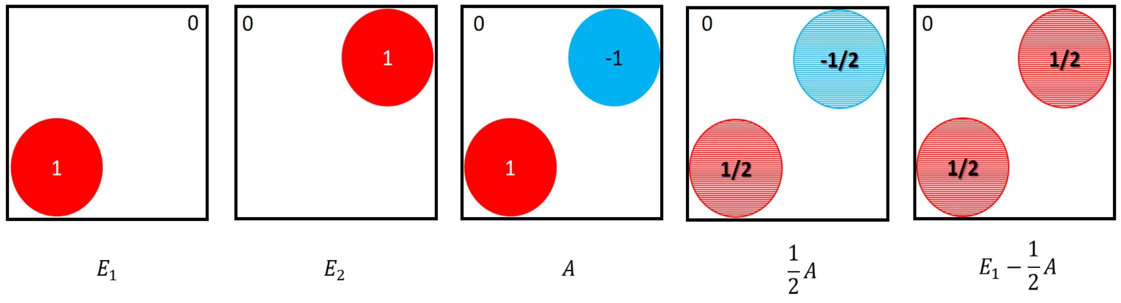

Figure 4.3. A sketch of an attempt to approximate the middle slice between two images using half of the time step size in HS.

Let us start out with an image E_1 that has the value 1 inside the red circle and 0 everywhere else. The second image, E_2, contains the same disc at a shifted location. Define $A = E_x u + E_y v$ (see the previous figure). By taking a portion of A, in this case half, we get the 4th image. And by finally computing $E_1 - \frac{1}{2}A$ we get the last figure. That is, if we want to use the reconstruction term to give us a nice smooth shift from Image 1 to Image 2 it will fail just like our linear interpolation did. By taking the middle term we get an average of the two images, and by taking different time increments for Δt (not equal to 0 or 1) we would always have a "shadow" either of Image 1 or Image 2 within our data frame.

4.2 The Explicit Pixel Movement Framework

After solving the inverse problem of optical flow, the pixel movement has been codified into the functions u and v. To take advantage of this information we can think of how to evolve each pixel in time with the direction (u, v). Supposing that we have a pixel located in position (x_0, y_0), we would like to evolve that pixel along velocity vector field (u, v) with time $t \in [0, 1]$. I.e. we want to find the new coordinates (x, y) that (x_0, y_0) evolves to in a given time t. For a small value of t, and initial conditions

$$x(0) = x_0, \tag{4.1}$$

$$y(0) = y_0, \tag{4.2}$$

we can approximate the coordinates of the new pixel location at

$$x = x_0 + t \cdot u, \tag{4.3}$$

$$y = y_0 + t \cdot v. \tag{4.4}$$

With this fairly simple setup we can actually track pixel movement in time. This is a first order Euler method for solving an initial value problem for a system of ordinary differential equations (ODE's). The general formulation of the problem could first be cast as an autonomous system

$$\frac{dx}{dt} = u(x, y), \tag{4.5}$$

$$\frac{dy}{dt} = v(x, y), \tag{4.6}$$

and ultimately as a non-autonomous system

$$\frac{dx}{dt} = u(x, y, t), \tag{4.7}$$

$$\frac{dy}{dt} = v(x, y, t), \tag{4.8}$$

In the following demonstration we take the phantom located at the origin and evolve it in time with the constant flow of $u = 1$ and $v = 1$. We do this for 20 iterations, with a time step of .5.

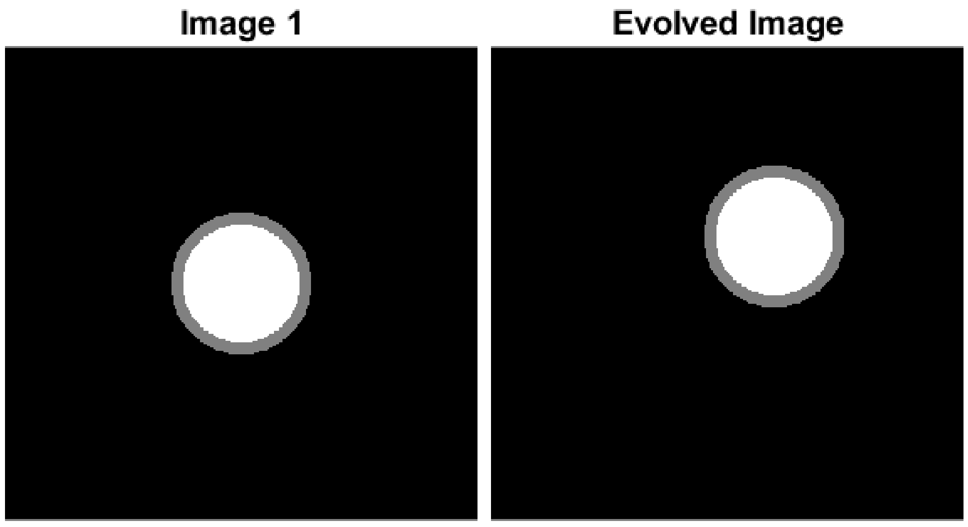

Figure 4.4. (Left) Original phantom. (Right) Phantom moved with $u = 1, v = 1$. Result of 20 iterations with step size .5.

The underlying flow field can be visualized as such

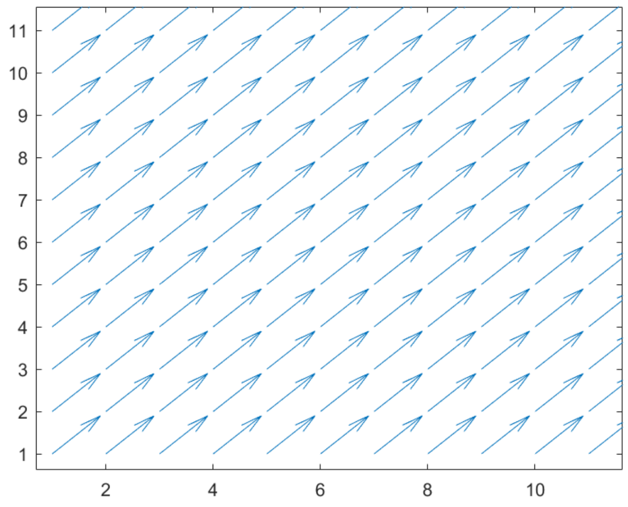

Figure 4.5. Zoomed-in flow field.

We can think of the motion as taking every pixel present in the scene and "walking" diagonally, in unison, towards the top right part of the image. This motion is smooth and continuous so all pixels are moved in the same manner.

We need not be restricted to linear flow fields. For example, if we would like to evolve our images in a nonlinear way, say $u = \cos(x)$ and $v = \sin(y)$, then after 10 iterations, with step size .1 we have the following figure.

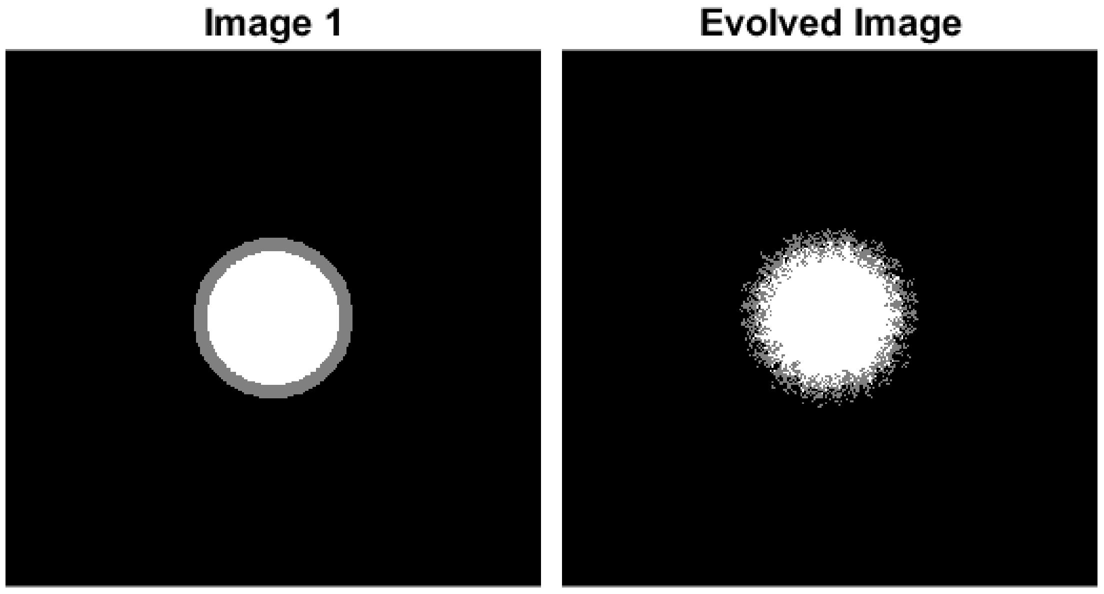

Figure 4.6. Using a non-linear flow field.

Let us now go back to the task of generating images of intermediate slices between two given images. In the upcoming experiments we use two synthetic images (of the same type as in our previous figures) to generate the (u, v) pair via our Horn-Schunck algorithm between the two images. We then feed this information into our motion algorithm and show the results.

Just to give a bit more information, we put the terms $\delta x = \delta y = .5$ and $\Delta t = 1$ (these are needed for the computation of E_x, E_y and E_t). We then set the time step to be equal to .5, which would mean that Image 1 should move (evolve) in the direction of Image 2 and would (hypothetically) arrive in 2 iterations. However, after 2 iterations we have the following

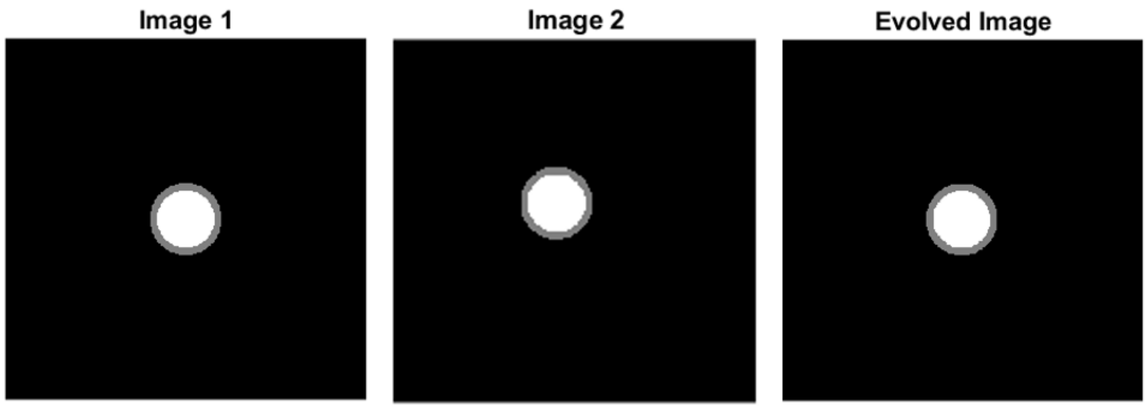

Figure 4.7. Using the flow field generated by HS after 2 iterations, with step size equal to .5.

We are trying to evolve the phantom, centered at $(0, 0)$, on the left into another phantom with the same structure, but moved in the north-western direction, centered at $(-20, 20)$ (see next figure). After the 2 iterations very little movement has happened overall. If we allow the algorithm to run more iterations (for example 10 steps instead of 2), we arrive at the following images.

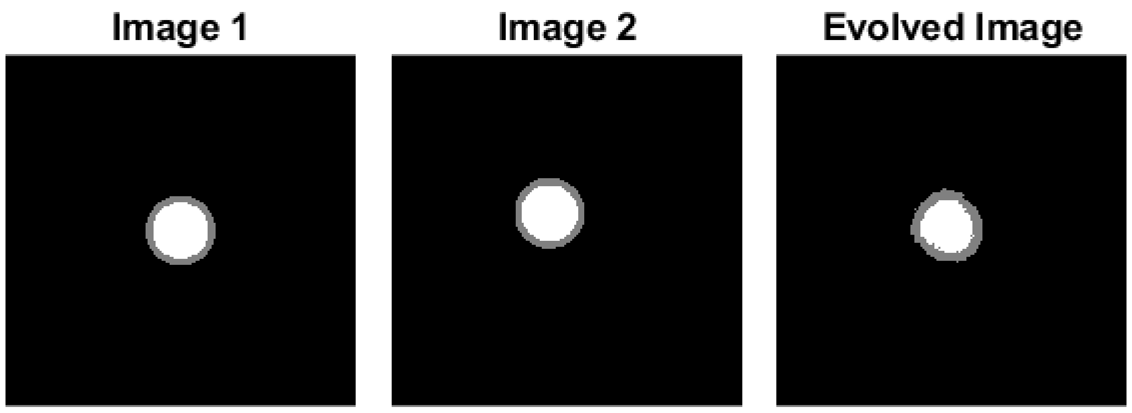

Figure 4.8. (Left) Image 1. (Middle) Image 2. (Right) Evolved Image 1 using the flow field generated from Image 1 and Image 2.

The movement that has occurred is inaccurate, as seen in the Evolved Image in Figure (4.8). Below we present also the quiver plot of the velocity vectors.

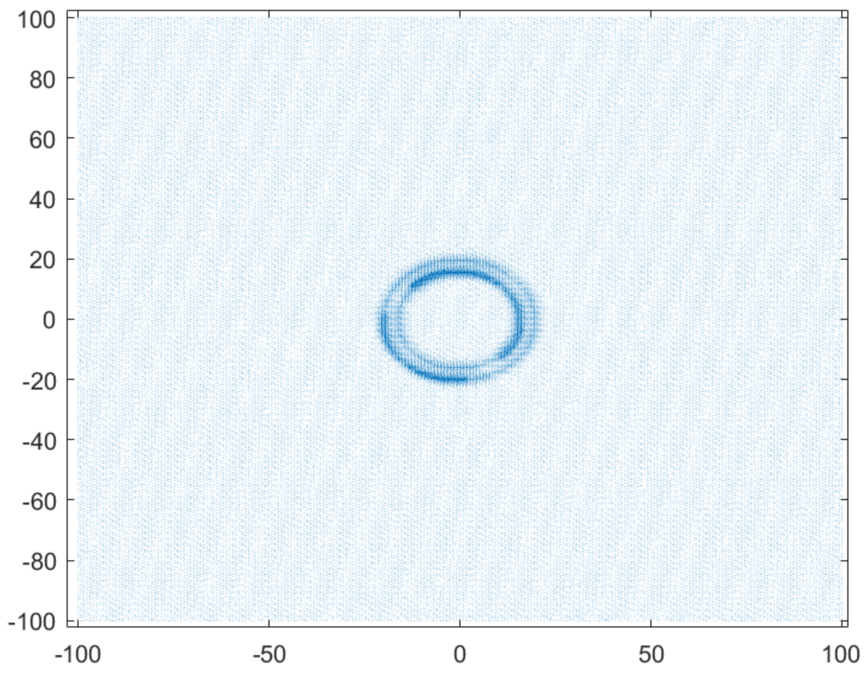

Figure 4.9. Flow field generated from Image 1 and Image 2.

We can see movement happening within the scene, but it is not uniform. Some portions of the image are being moved correctly (in the NW direction), but some pixels are not being moved at all.

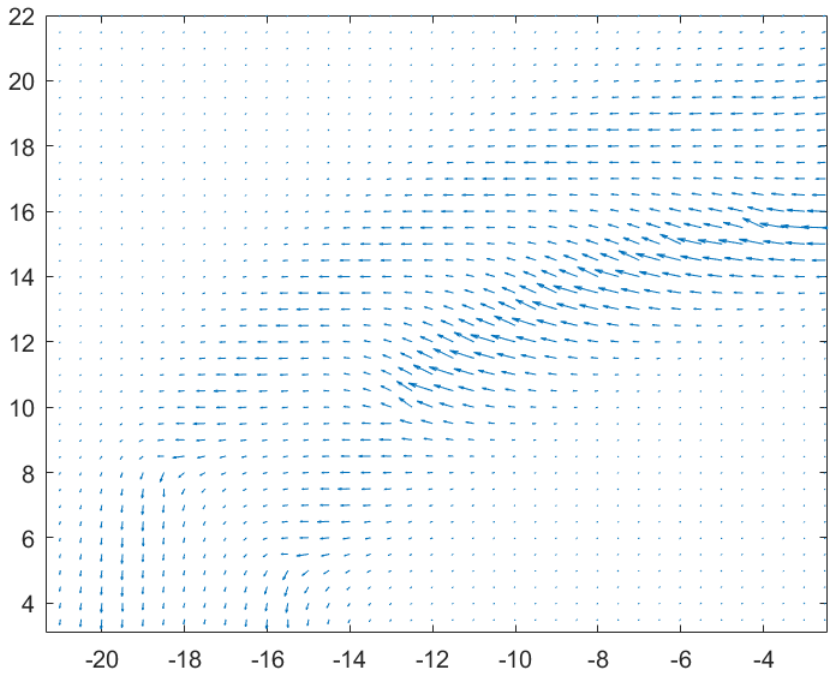

Figure 4.10. Zoom-in of the top-left portion of the phantom.

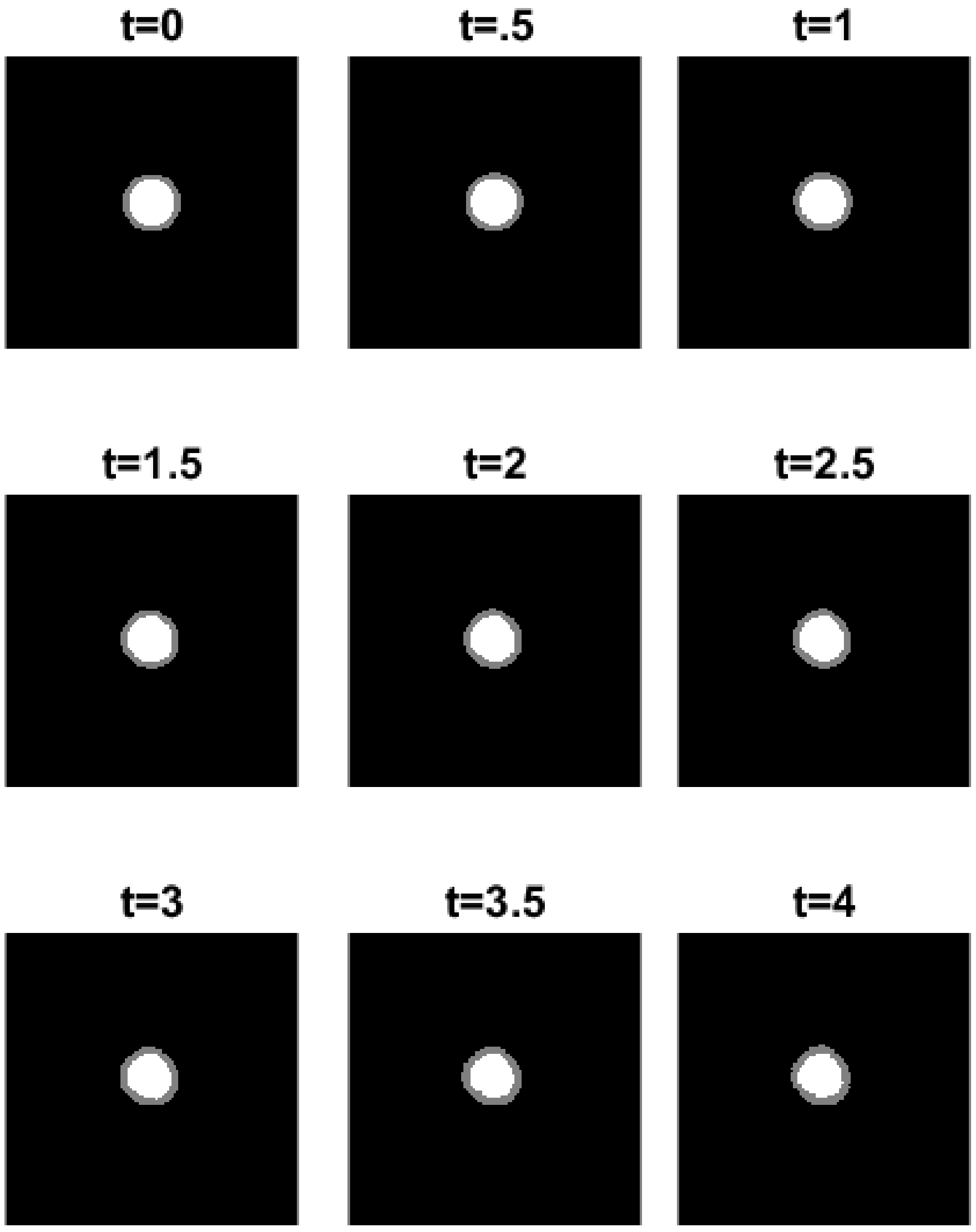

Figure 4.11. Evolved Image with the flow field prescribed via Image 1 and Image 2. (Top-Left) Starting Image. (Top-Middle) Evolved Image 1 step. (Top-Right) Evolved Image 2 steps. (Middle-Left) Evolved Image 3 steps. Etc..

In other words, the field generated by the HS-method is yielding a non-uniform motion. To correct this we will update the (u, v) after every iteration of our algorithm. In essence we start with the (u, v) given by the original Image 1 and Image 2. After one step of pixel movement we recalculate our (u, v) in between Image 2 and our Evolved Image, as seen below. Again, we are taking Image 1 and evolving it with our flow field, so the images in the next figure are resultant steps after each iterate. After our updates we arrive at the following set of images.

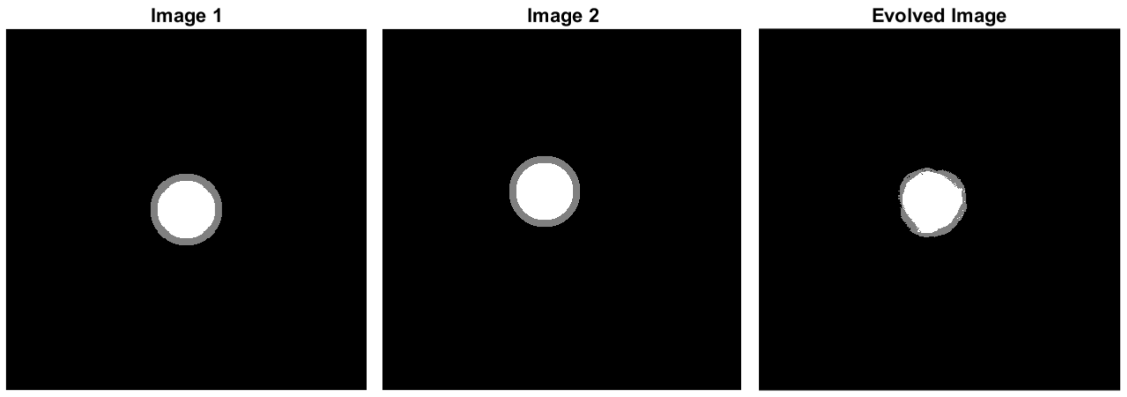

Figure 4.12. Evolved Image with the flow field generated via Image 1 and evolved Image 1 after 10 steps.

While still imperfect, we can see through the relative error

$$\text{Relative Error} = \frac{\|\tilde{E} - E_2\|}{\|E_2\|}$$

that our motion is yielding better and better results after each iterate.

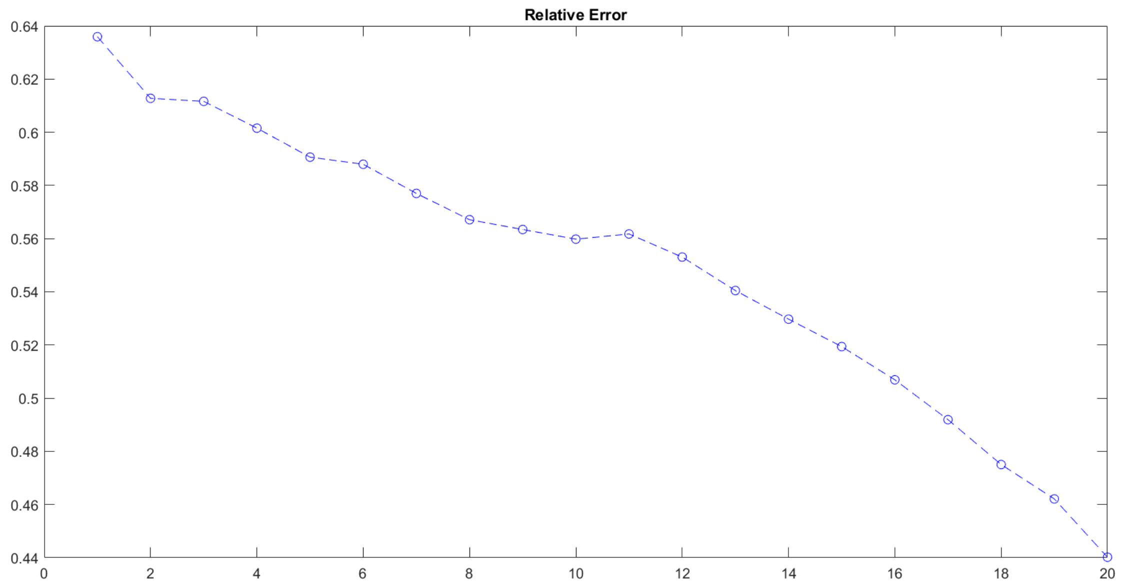

Figure 4.13. Relative error between Image 2 and the Evolved Image.

We can see that updating our (u, v) after each iterate causes the image to evolve towards Image 2.

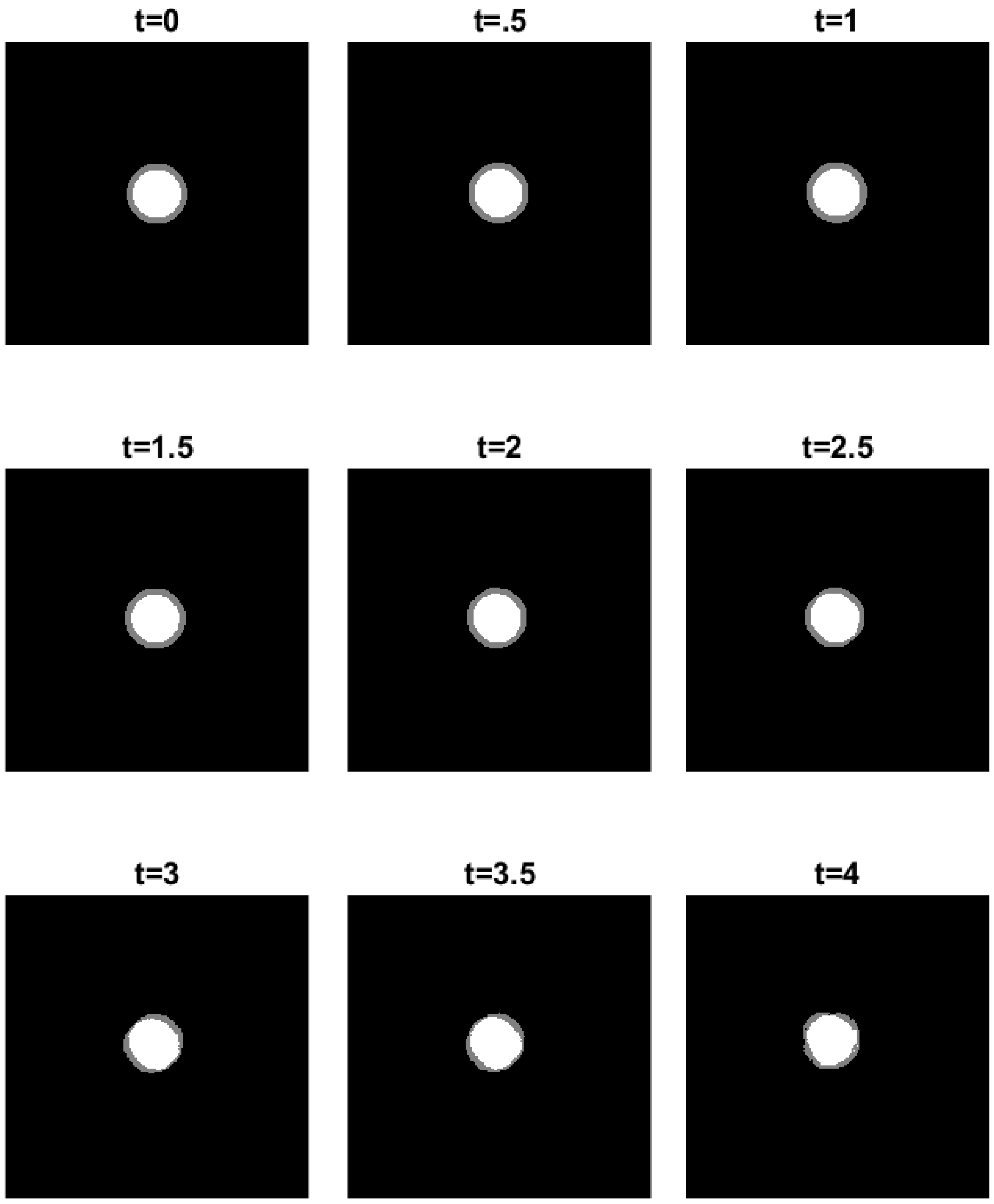

Figure 4.14. Evolved Image with updated (u, v).

Because the flow field that we are generating comes with errors and is non-uniform, we can fix this by applying a Gaussian filter to the field "smoothening" it after each generation of (u, v). As a result, the pixels will be oriented along the most pronounced directions. We employ the imgaussfilt() command in MATLAB to the components (u, v) after they are computed. One may think of this function as the discritized convolution between u and a Gaussian kernal with a prescribed variance (v is also "smoothened" in the same fashion). More details can be found in "help imgaussfilt" in MATLAB for more discussion.

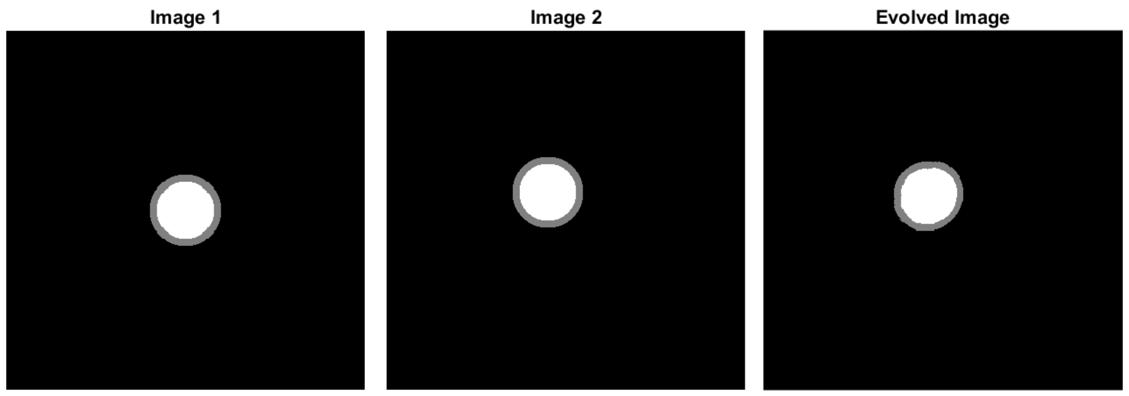

Figure 4.15. Updated (u, v) along with a Gaussian filter on (u, v).

We can see that by correcting (u, v) after each step and passing a Gaussian filter over the results we arrive at an Evolved Image that more closely resembles our target Image 2.

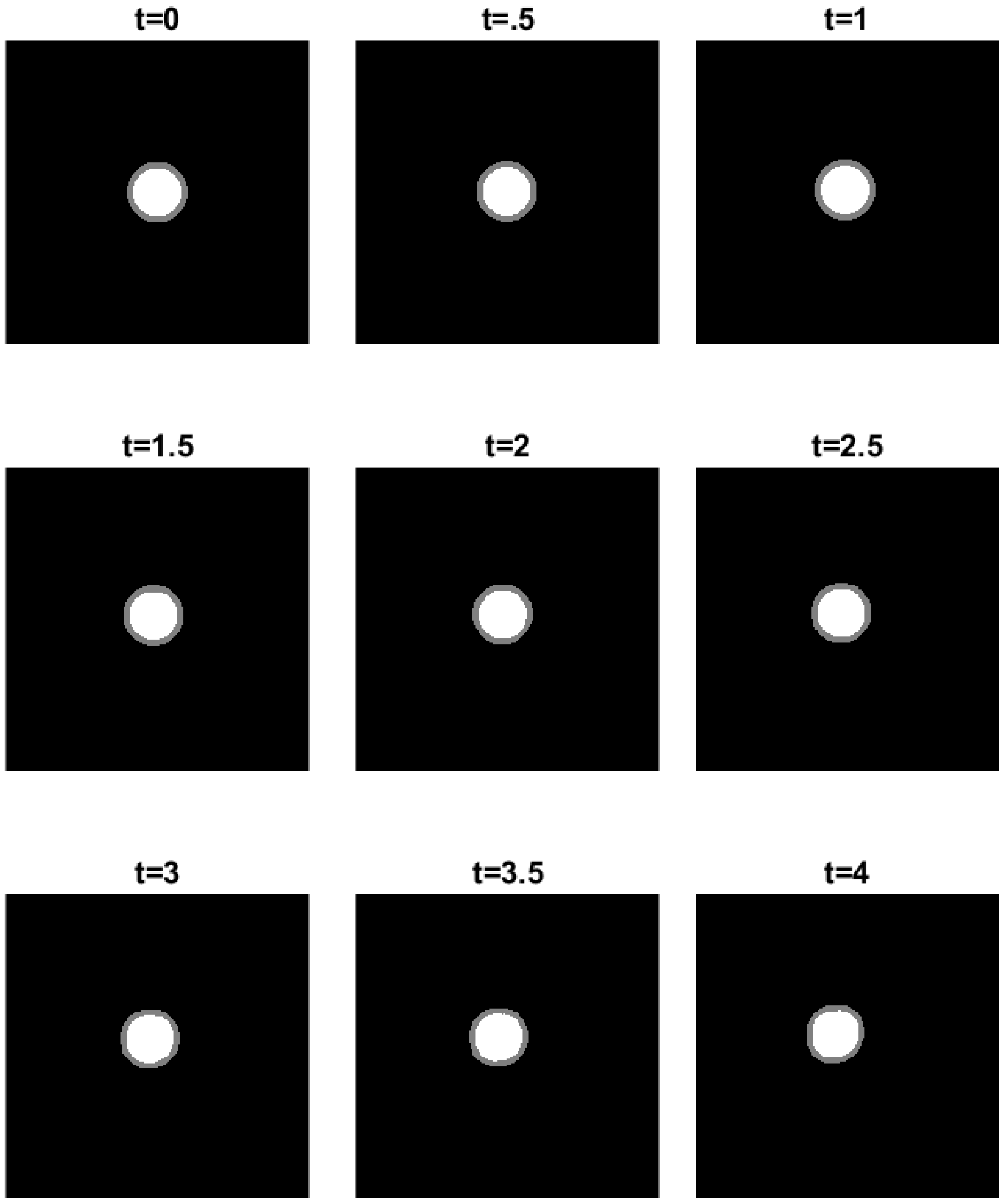

Figure 4.16. Updated (u, v) along with a Gaussian filter on (u, v).

As we iterate through our algorithm, the relative error between the Evolved Image and Image 2 decreases.

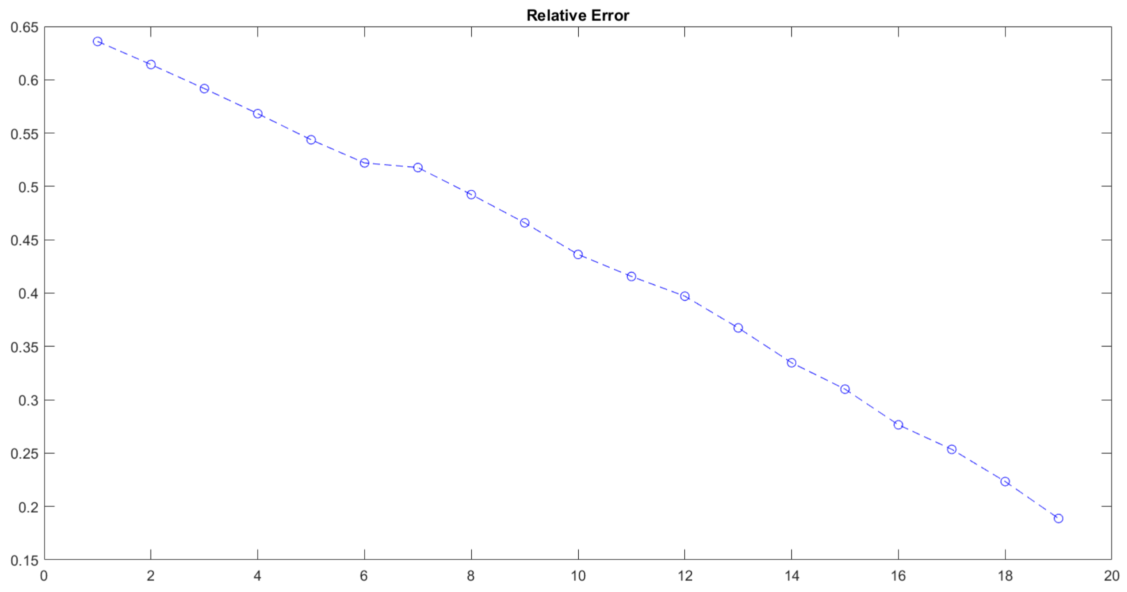

Figure 4.17. The relative error of the original image with the new image.

By smoothening our flow field we are able to locally correct inaccuracies in the flow field and we arrive at a type of motion more aligned with our original goal. The amount with which one should smoothed these fields is largely ad hoc. It should be enough to orient pixels in a common direction, but not so severe as to delete crucial path information from the (u, v).

4.3 Explicit Movement in MRI

In this section we apply the method described in Section 4.2 to actual MR images obtained in clinical setting. Using the explicit movement method we can take a pair of our MRIs and generate a method to move pixels from a starting image

(Image 1) to another image (Image 2). We will begin with the following image pair and explain the method as we go along. For the algorithm in its entirety please see the end of Appendix A.

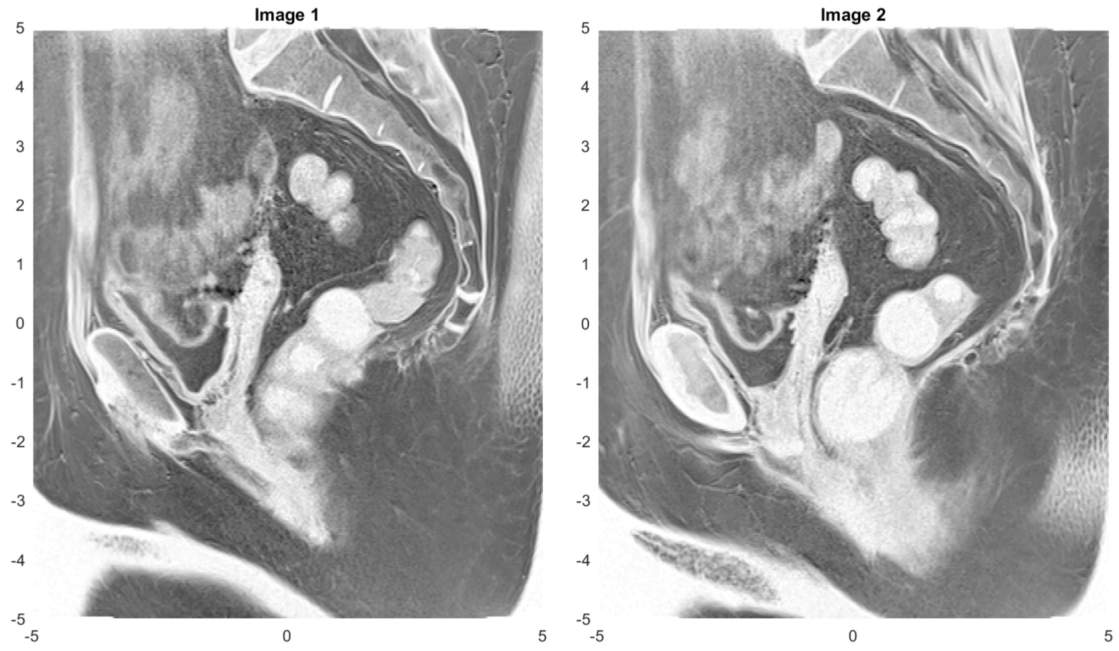

Figure 4.18. Original image pair.

These two images are two successive slices of an MRI sequence. We can see that while they have some similarities, there are key differences between them. The shadowy region in the bottom right corner of Image 2 is not present in Image 1. The bone in the middle left portion of Image 1 becomes larger in the second image.

Using the explicit pixel movement technique covered in the previous section, we can actually attempt to deform Image 1 into Image 2. But the main deviation from our previous methods of changing the flow field (u, v) discussed in Sections 3.1 and 3.2 (global and local change in the fields (u, v)), is that here we have movement of the image scene in between each iteration. In Section 3.1 we would have Image 1 and then add to it a portion of the $\Delta t \cdot (E_x \cdot u + E_y \cdot v)$ term until we arrived at the second image. Here we apply the ODE framework of forward propagation and use the advection term to help us update the flow field after each iteration of the movement. This key difference is better viewed in real-time via a movie created by the data terms. One may need to implement our code in the Appendix to fully appreciate the differences.

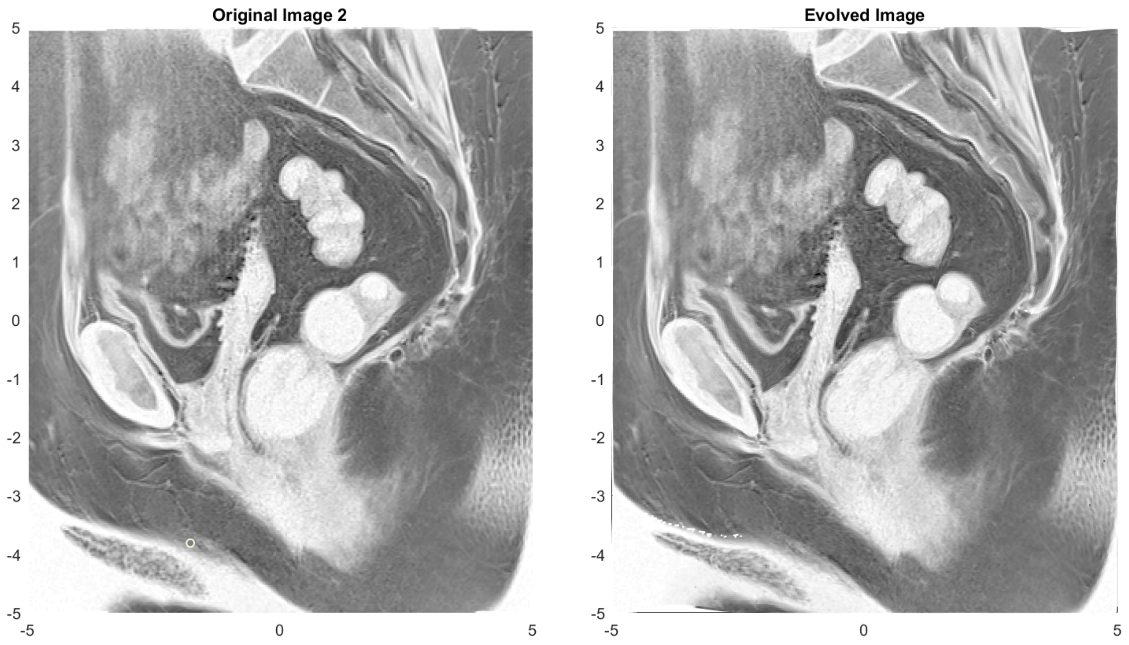

Figure 4.19. Image 2 and Evolved Image.

As one can see, we arrive at an image on the right that is very close to the target Image 2. There are some differences though. In the middle of the image a part of the intestine is not quite where it should be. We also have a few speckles towards the bottom left part of the Evolved Image that should not be there. Also directly in the center of the evolved image we don't have the correct shape.

In the following figure we can see the intermediate images that are being generated after each iteration.

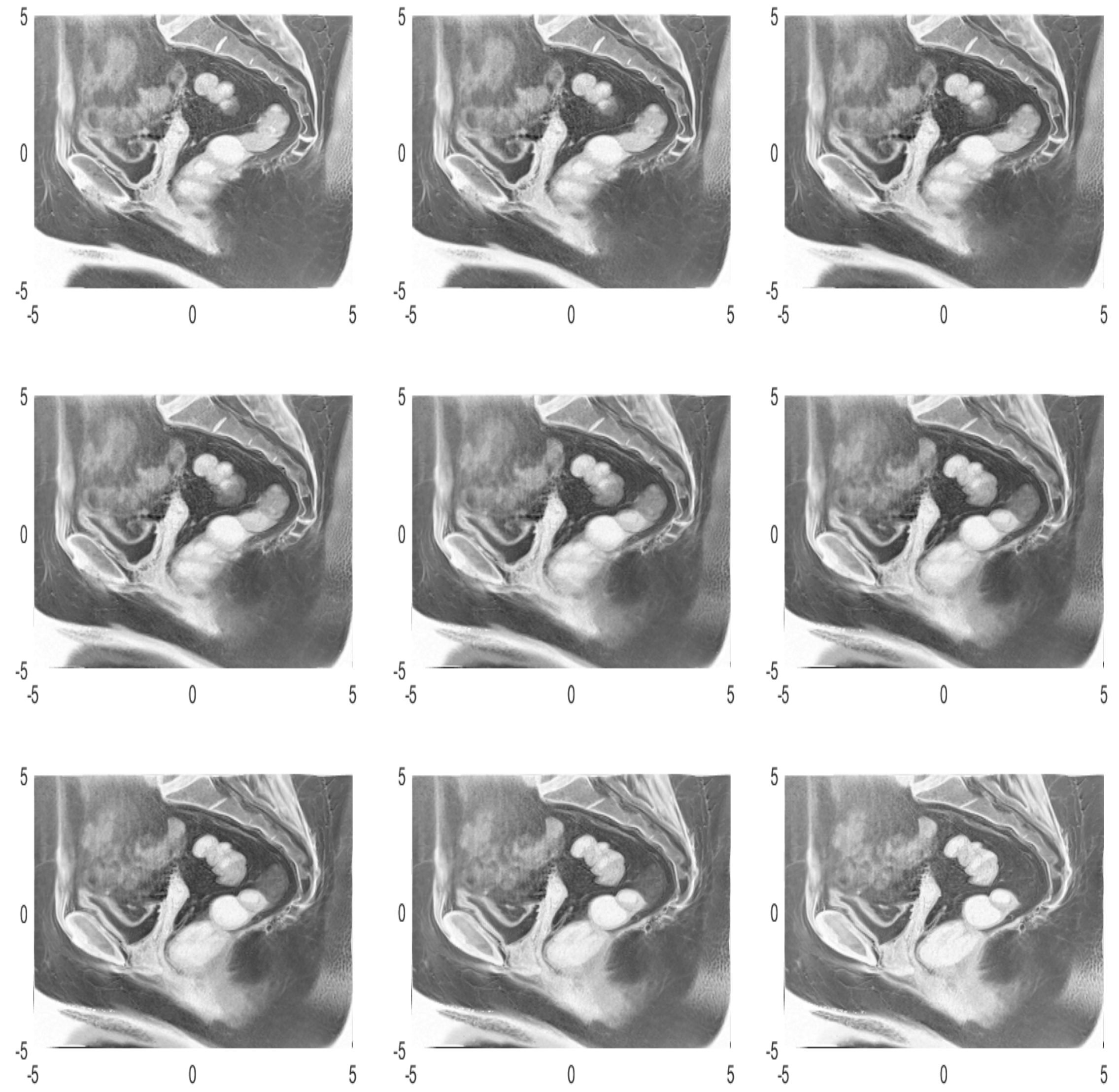

Figure 4.20. The sequence of generated images.

4.4 Methods of Forward Propagation

By now we have a few different methods in our toolbox to create images. We have done the groundwork with Horn and Schunck's framework and have shown classes of images, where reconstruction of $\tilde{E}$ is possible, and others where extra information is needed (E.g. addition of a noise term). But after completing the HS algorithm we have been using different techniques to fill in the "in-between" data from our original data. We wanted to know how one can meaningfully generate this intermediate data that captures the movement of our image. That is, how can one deform Image 1 into Image 2 via these processes that we have stated so far?

We now discuss the pros and cons of each of the methods we have used so far.

Approach 1 Propagating via HS

Utilizing an iterative method for generating (u, v) that satisfies the advection equation, we varied our (u, v) to get approximate data in the intermediate layers (see Figure 2.4). We showed that, in the MRI case, no extra information was needed and the image generated via

$$\tilde{E_2} = E_1 - \Delta t \cdot (E_x \cdot u + E_y \cdot v)$$

is almost an exact Image 2. But in the case of the simple phantoms, we needed to add a noise term before our pseudo-Image 2 and actual Image 2 would agree. This method is simple to implement, but has major flaws when it comes to intermediate approximations, since their evolution is "additive". By that we mean that we are taking Image 1 and adding successively larger portions of $-\Delta t \cdot (E_x \cdot u + E_y \cdot v)$ through each iterate. This has the net effect of taking parts of Image 1 that should move to become Image 2, and "dimming" their contribution in the intermediary. While at the same time taking regions of Image 2 that are not present in Image 1 and slowly adding them into the scene.

This may be acceptable as a first step and we can use this methods and variants to generate new data from our existing two images. But the way we evolve through the intermediate steps leaves something to be desired. We want the regions of our first image to continuously move and/or deform into Image 2, and with this method boundaries are not moving. But instead they are dimming and brightening through the iterates.

Approach 2 The ODE approach

The second approach was to start from a system of ODEs

$$\frac{dx}{dt} = u$$
$$\frac{dy}{dt} = v$$

with the initial data $(x(0), y(0)) = (x_0, y_0)$ and evolve these images via a prescribed flow field (u, v) (See Figure 2.18 and Appendix B). We saw here that in the case of simple phantoms we had to recalculate (u, v) after each iteration and initialize our starting guess for $(\overline{u}, \overline{v})$ differently to make sure that the motion that was seen in the intermediate steps was correct. We also had to apply a Gaussian filter to (u, v) to correct for inaccuracies of our flow field. This is needed because the (u, v) that we get from HS will fail to be accurate around the boundary of a purely synthetic image. Sharp corners on the boundary imply a discontinuity of our flow field and thus those pixels in the boundary need to be adjusted.

Approach 3 The Hybrid Approach

By taking the ODE approach, with iterative updates, and the flow field calculator from HS we are able to get actual motion within our intermediate steps (See Figure 2.27). But this relies on continual updates within our flow field and more parameters to tune. That leads to a major difficulty for the hybrid case, namely the need to tune a lot of parameters.

One needs to make choices for λ, the Gaussian filter, the proportions of previous data A and the advection term to "mix" (See Appendix A) in the pixel update stage. We take a portion p of the previous iterate, and a portion $q = 1 - p$ of the advection term and mix them together in the image update stage

$$A := p \cdot A + q \cdot \left(A^1 - t \cdot (E_x^0 \cdot u^0 + E_y^0 \cdot v^0)\right),$$

where E_x^0, E_y^0, U^0, V^0 are the E_x, E_y, u, v terms that are satisfied for the original image pair, and A^1 is the original Image 1. Different proportions of these terms leads to different images. In Figure 2.26 we take $p = q = 1/2$, but this is another thing to keep in mind.

We also explore changing our coordinate updates. When using the ODE approach we used

$$x = x_0 + t \cdot u$$

$$y = y_0 + t \cdot v$$

This is a first order explicit Euler method, where we discard the higher order terms. This works great as a proof of concept model, however we wanted to change this framework as well to see how this is affecting our intermediate images. So we also used a nonstandard finite difference scheme, specifically a non-standard explicit Euler method, of the form

$$x = x_0 + \frac{(1 - e^{-\gamma \cdot t}) \cdot u}{\gamma},$$
$$y = y_0 + \frac{(1 - e^{-\gamma \cdot t}) \cdot v}{\gamma},$$

where γ is a parameter you choose. The choice of a secondary method was inspired by work from Dimitrov and Kojouharov [7]. It involves no further computation than the one done above, so the speed is not affected.

The following images are the result of running the standard Euler's method and the non-standard Euler's method side by side. We choose $p = .90$ and $q = .10$ for the mixing term and $\gamma = .5$ for the parameter in the NSE. Some things that stand out is that for initial values of the iterates both methods seem to be giving similar results. We are running 20 iterations and for about 15 of them they are similar. Not until iterations after 15 do they start to drastically differ.

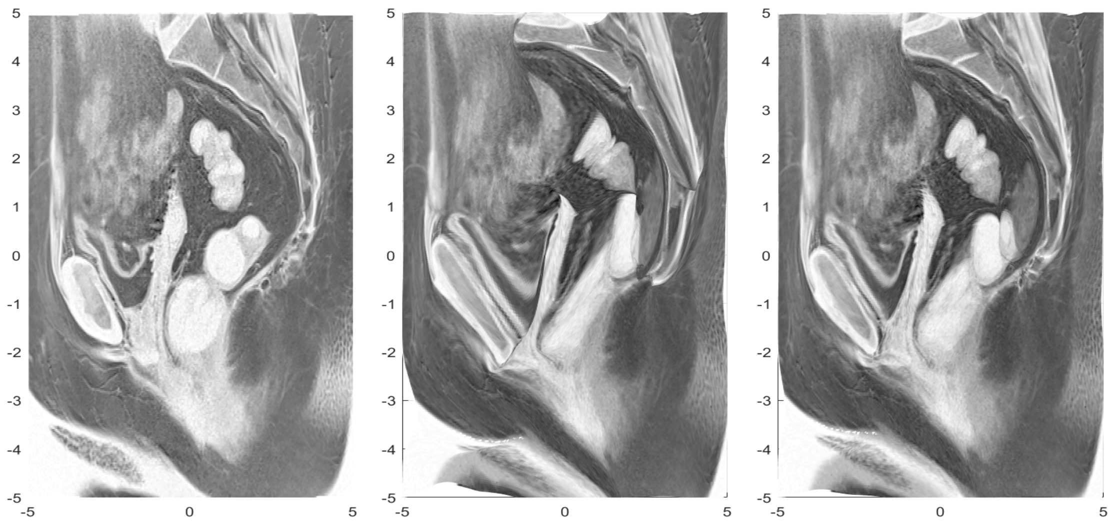

Figure 4.21. (Left) Actual Image 2. (Middle) Final iteration using Euler's method. (Right) Final iteration using Non-standard Euler $\gamma = .5$.

Again, since we are limited by the medium, we take samples of each of the methods as they evolve and show iterates $\{1, 5, 15, 20\}$, to give the reader a sense of what the movie of these looks like.

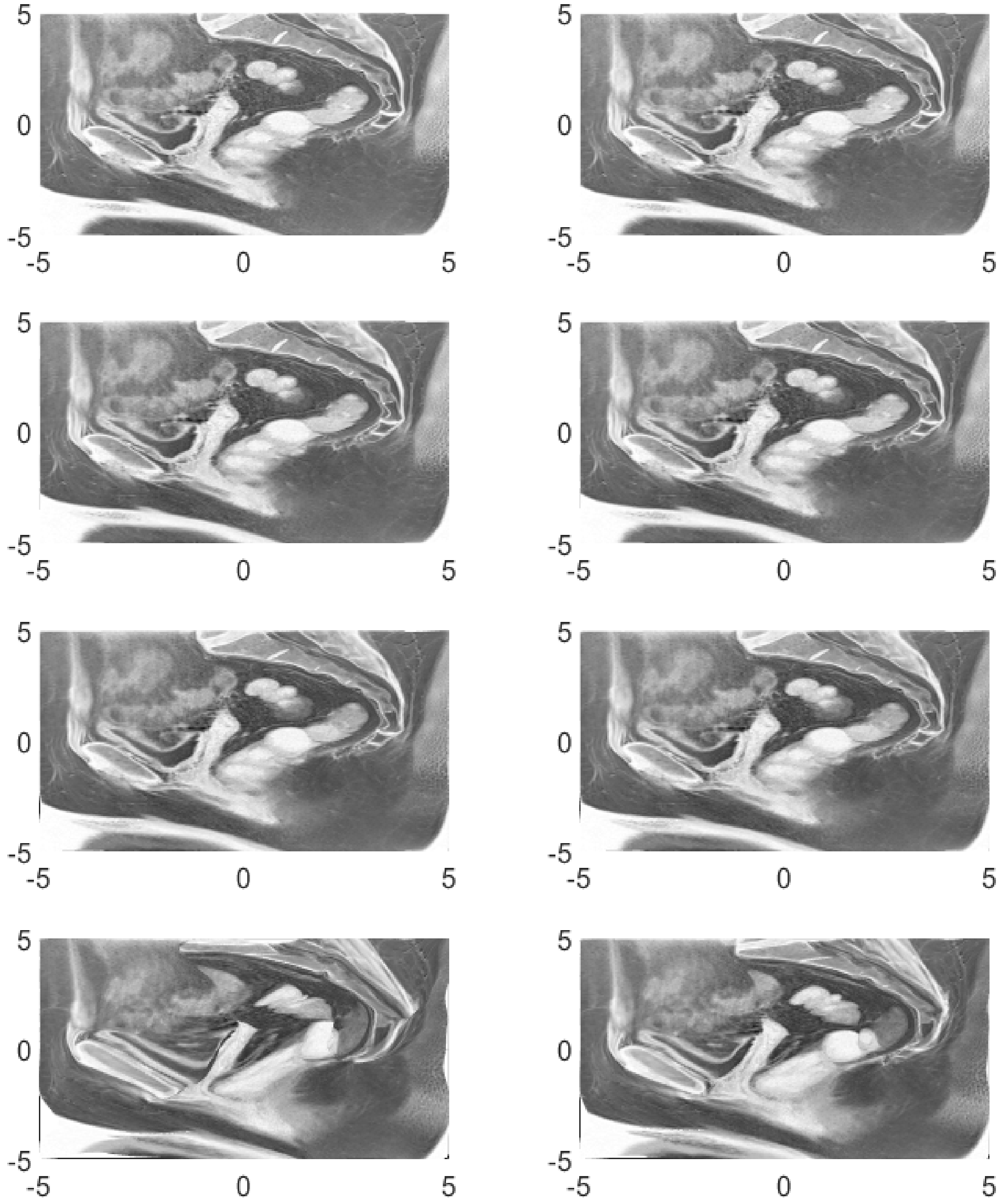

Figure 4.22. (Left column) Samples of evolution method using Euler. (Right column) Samples of evolution method using Non-standard Euler $\gamma = .5$.

4.5 Area Preserving Transformations

During our discussions on flow propagation an interesting fact seemed to present itself, namely the fact that the brightness consistency constraint implies area preservation. To expand on that for a moment, the brightness consistency constraint (BCC) says, in plain English, that all of the pixels that are present in one moment of time can be found "close by" in another moment of time. It is sort of a closed system, where we allow pixels to change in a small region, but we do not let pixels instantly "pop into existence" or "delete" themselves from the image.

This implies that the corresponding transformations should be area preserving. In the case of affine transformations, the area preserving ones are limited to the following four and their compositions:

- shifts/translations
- rotations
- squeezing
- shear

Almost all of these transformations are linear except for one. Shifting an image is not linear because it would take the origin to another location. The other three are linear. Area preserving affine transformations are called **special affine transformations**.

The interesting thing about linear transformations in the plane is that they correspond to matrix multiplication. Forms of the matrices for the aforementioned linear transformations are well known, for example

$$B_{\text{Rot}} = \begin{bmatrix} \cos t & -\sin t \\ \sin t & \cos t \end{bmatrix} \qquad B_{\text{Dial}} = \begin{bmatrix} t & 0 \\ 0 & \frac{1}{t} \end{bmatrix} \qquad B_{\text{Shear}} = \begin{bmatrix} 1 & t \\ 0 & 1 \end{bmatrix}, \tag{4.9}$$

where the matrices B_{Rot}, B_{Dial}, and B_{Shear} correspond to rotation, dilation and shearing respectively. Here t is a parameter that determines the "size" of the transformation. For example in the rotation matrix B_{Rot}, t is the angular amount of rotation counterclockwise of a point.

In our setup, we would like to accomplish the same by mapping the vector $(x(0), y(0))$ to a vector $(x(t), y(t))$ not through a vector matrix multiplication, but by solving an initial value problem. That is, given a point located at $(x(0), y(0))$ we want to find the $(x(t), y(t))$ satisfying the following system

$$\begin{bmatrix} x'(t) \\ y'(t) \end{bmatrix} = A \begin{bmatrix} x(t) \\ y(t) \end{bmatrix}. \tag{4.10}$$

Remark 4. Notice, that the matrix A in formula (4.10) is not the same as the matrix of the corresponding linear transformation. Here we assume that A is a matrix of constant coefficients.

This is a classical initial value problem that is found in any textbook about ODEs. If A is such that it has two linearly independent eigenvectors $\bar{v}^{(1)}$ and $\bar{v}^{(2)}$ and the eigenvalues λ_1, λ_2 are real, then we have the following solution

$$\begin{bmatrix} x(t) \\ y(t) \end{bmatrix} = c_1 e^{\lambda_1 t} \begin{bmatrix} v_1^{(1)} \\ v_2^{(1)} \end{bmatrix} + c_2 e^{\lambda_2 t} \begin{bmatrix} v_1^{(2)} \\ v_2^{(2)} \end{bmatrix}. \tag{4.11}$$

This coupled with our initial condition that at time $t = 0$ our pixel is located at $(x(0), y(0))$, implies that we have

$$\begin{bmatrix} x(0) \\ y(0) \end{bmatrix} = c_1 \bar{v}^{(1)} + c_2 \bar{v}^{(2)} = \underbrace{\begin{bmatrix} v_1^{(1)} & v_1^{(2)} \\ v_2^{(1)} & v_2^{(2)} \end{bmatrix}}_{V} \underbrace{\begin{bmatrix} c_1 \\ c_2 \end{bmatrix}}_{\bar{c}}$$

V^{-1} exists because of the linear independence of $\bar{v}^{(1)}$ and $\bar{v}^{(2)}$, hence we can write the following

$$\bar{c} = V^{-1} \begin{bmatrix} x(0) \\ y(0) \end{bmatrix}.$$

We may also write our solution (4.15) as

$$\begin{bmatrix} x(t) \\ y(t) \end{bmatrix} = V \begin{bmatrix} c_1 e^{\lambda_1 t} \\ c_2 e^{\lambda_2 t} \end{bmatrix} = V \underbrace{\begin{bmatrix} e^{\lambda_1 t} & 0 \\ 0 & e^{\lambda_2 t} \end{bmatrix}}_{\Lambda} \underbrace{\begin{bmatrix} c_1 \\ c_2 \end{bmatrix}}_{\bar{c}} = V\Lambda V^{-1} \begin{bmatrix} x(0) \\ y(0) \end{bmatrix}.$$

Let us make the substitution that

$$B = V\Lambda V^{-1}. \tag{4.12}$$

The representation (4.12) is a composition of transformations usually called the **change of basis formula**. Utilizing this new notation we have

$$\begin{bmatrix} x(t) \\ y(t) \end{bmatrix} = B \begin{bmatrix} x(0) \\ y(0) \end{bmatrix}. \tag{4.13}$$

Which geometrically implies application of a scaling transformation along new basis vectors coinciding with the eigenvectors of matrix A.Thus

$$\det(B) = \det(V\Lambda V^{-1}) = \det(V) \cdot \det(\Lambda) \cdot \det(V)^{-1} = e^{(\lambda_1+\lambda_2)t}. \tag{4.14}$$

For area preservation (i.e. for scaling to be a squeezing transformation) we would want $|\det(B)| = 1$. Then, we want

$$|e^{(\lambda_1+\lambda_2)t}| = 1$$

which means either

$$t = 0 \qquad \text{or} \qquad \lambda_1 = -\lambda_2$$

At the moment $t = 0$ we naturally have area preservation, because no movement has occurred. And since we want the general result we must have $\lambda_1 = -\lambda_2$.

Example 1 A system that corresponds to the "squeezing" of the plane along the x-axis and y-axis can be written as follows

$$\begin{bmatrix} x'(t) \\ y'(t) \end{bmatrix} = \begin{bmatrix} x(t) \\ -y(t) \end{bmatrix}. \tag{4.15}$$

We can rewrite this system as follows

$$\begin{bmatrix} x'(t) \\ y'(t) \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \cdot \begin{bmatrix} x(t) \\ y(t) \end{bmatrix}.$$

One can see that $|\det(A)| = 1$, the eigenvalues $\lambda_1 = 1$ and $\lambda_2 = -1$ are real and the eigenvectors $\bar{v}^1 = [1, 0]^T$ and $\bar{v}^2 = [0, 1]^T$ are linearly independent. And finally, the solution of this system with initial condition at time $t = 0$ at pixel $(x(0), y(0))$ is

$$\begin{bmatrix} x(t) \\ y(t) \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} e^t & 0 \\ 0 & e^{-t} \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}^{-1} \begin{bmatrix} x(0) \\ y(0) \end{bmatrix}.$$

Hence

$$\begin{bmatrix} x(t) \\ y(t) \end{bmatrix} = \begin{bmatrix} e^t & 0 \\ 0 & e^{-t} \end{bmatrix} \begin{bmatrix} x(0) \\ y(0) \end{bmatrix}.$$

For an idea of what this motion looks like please see Figure B.6 in Appendix B.

Example 2 We can try the following system

$$\begin{bmatrix} x'(t) \\ y'(t) \end{bmatrix} = \begin{bmatrix} x(t) + y(t) \\ 3x(t) - y(t) \end{bmatrix}, \tag{4.16}$$

which again can be written as

$$\begin{bmatrix} x'(t) \\ y'(t) \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 3 & -1 \end{bmatrix} \cdot \begin{bmatrix} x(t) \\ y(t) \end{bmatrix}.$$

Here we have $\lambda_1 = 2, \lambda_2 = -2$ and the eigenvectors are $\bar{v}^1 = [1, 1]^T$ and $\bar{v}^2 = [-1, 3]^T$. This makes our solution take the form

$$\begin{bmatrix} x(t) \\ y(t) \end{bmatrix} = \begin{bmatrix} 1 & -1 \\ 1 & 3 \end{bmatrix} \cdot \begin{bmatrix} e^{2t} & 0 \\ 0 & e^{-2t} \end{bmatrix} \cdot \begin{bmatrix} 1 & -1 \\ 1 & 3 \end{bmatrix}^{-1} \begin{bmatrix} x(0) \\ y(0) \end{bmatrix}$$

For an idea of what this motion looks like please see Figure B.7 in Appendix B.

Example 3 For a case where we do not have real eigenvalues or real eigenvectors we could do the same sort of analysis with the rotation system given by

$$\begin{bmatrix} x'(t) \\ y'(t) \end{bmatrix} = \begin{bmatrix} -y(t) \\ x(t) \end{bmatrix} \tag{4.17}$$

which can be written as

$$\begin{bmatrix} x'(t) \\ y'(t) \end{bmatrix} = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} x(t) \\ y(t) \end{bmatrix}$$

our eigenvalues are now $\lambda_1 = i, \lambda_2 = -i$ and our eigenvectors are $\bar{v}^1 = [i, 1]^T = [0, 1]^T + i[1, 0]^T$ and $\bar{v}^2 = [-i, 1]^T = [0, 1]^T + i[-1, 0]^T$. Then for our solution we have

$$\begin{bmatrix} x(t) \\ y(t) \end{bmatrix} = c_1 \left(\cos t \begin{bmatrix} 0 \\ 1 \end{bmatrix} - \sin t \begin{bmatrix} 1 \\ 0 \end{bmatrix} \right) + c_2 \left(\sin t \begin{bmatrix} 0 \\ 1 \end{bmatrix} + \cos t \begin{bmatrix} 1 \\ 0 \end{bmatrix} \right)$$

Which can be written as

$$\begin{bmatrix} x(t) \\ y(t) \end{bmatrix} = \begin{bmatrix} -c_1 \sin t + c_2 \cos t \\ c_1 \cos t + c_2 \sin t \end{bmatrix} = \begin{bmatrix} -\sin t & \cos t \\ \cos t & \sin t \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \end{bmatrix}.$$

We can determine our initial conditions via

$$\begin{bmatrix} x_0 \\ y_0 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \end{bmatrix}$$

where

$$\begin{bmatrix} c_1 \\ c_2 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} x_0 \\ y_0 \end{bmatrix}.$$

Now we arrive at

$$\begin{bmatrix} x(t) \\ y(t) \end{bmatrix} = \begin{bmatrix} -\sin t & \cos t \\ \cos t & \sin t \end{bmatrix} \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} x_0 \\ y_0 \end{bmatrix} = \begin{bmatrix} \cos t & -\sin t \\ \sin t & \cos t \end{bmatrix} \begin{bmatrix} x_0 \\ y_0 \end{bmatrix}.$$

For an idea of what this motion looks like please see Figure B.1 in Appendix B.

Example 4 For our last example we have the system

$$\begin{bmatrix} x'(t) \\ y'(t) \end{bmatrix} = \begin{bmatrix} y(t) \\ 0 \end{bmatrix} \tag{4.18}$$

which corresponds to shear motion. We can instead write this as

$$\begin{bmatrix} x'(t) \\ y'(t) \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} x(t) \\ y(t) \end{bmatrix}.$$

This system has the eigenvalue $\lambda = 0$ with algebraic multiplicity 2, and eigenvector $\bar{v}^{(1)} = [1, 0]^T$. We then go searching for a **generalized eigenvector** $\bar{v}^{(1)} = [\nu_1, \nu_2]^T$ through the following process

$$(A - \lambda I)\bar{v}^{(2)} = \bar{v}^{(1)}$$

yields

$$\left(\begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} - 0 \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \right) \begin{bmatrix} \nu_1 \\ \nu_2 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$
$$\begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} \nu_1 \\ \nu_2 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}.$$

This means that $0\nu_1 + 1\nu_2 = 1$, so we arrive at $\nu_2 = 1$ and ν_1 is "free". We choose $\nu_1 = 0$, thus our generalized eigenvector is $\bar{v}^{(2)} = [0, 1]^T$. It is well known that solutions of this type will take the form

$$\begin{aligned} \begin{bmatrix} x(t) \\ y(t) \end{bmatrix} &= c_1 e^{at} \bar{v}^{(1)} + c_2 e^{at} \left(t\bar{v}^{(1)} + \bar{v}^{(2)} \right) \\ &= c_1 \begin{bmatrix} 1 \\ 0 \end{bmatrix} + c_2 \left(t \begin{bmatrix} 1 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} \right). \end{aligned}$$

We then determine the coefficients via

$$\begin{bmatrix} x(0) \\ y(0) \end{bmatrix} = \begin{bmatrix} c_1 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ c_2 \end{bmatrix} = \begin{bmatrix} c_1 \\ c_2 \end{bmatrix}.$$

Thus our solution becomes

$$\begin{aligned}
\begin{bmatrix} x(t) \\ y(t) \end{bmatrix} &= x(0) \begin{bmatrix} 1 \\ 0 \end{bmatrix} + y(0) \left(t \begin{bmatrix} 1 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} \right) \\
&= \begin{bmatrix} y(0)t + x(0) \\ y(0) \end{bmatrix} \\
&= \begin{bmatrix} 1 & t \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x(0) \\ y(0) \end{bmatrix}.
\end{aligned}$$

For an idea of what this motion looks like please see Figure B.7 in Appendix B.

APPENDIX A

Codes

We have assembled the codes that were used in the creation of this thesis. The first code that we have is the one that is needed to compute the terms

$$\frac{\partial E}{\partial x}, \frac{\partial E}{\partial y}, \frac{\partial E}{\partial t}.$$

These codes are needed in the creation of (u, v) via the Horn-Schunck method.

```
function [Ex,Ey,Et] = Compute_ExEyEt(A,B,delX,delY,delT)
[n,m] = size(A);      % input an image(matrix) of size n x m
Ex = zeros(n,m);      % initialize the derivatives
Ey = zeros(n,m);      % as zero matrices
Et = zeros(n,m);
for i = 2:n-1
    for j = 2:m-1
        Ex(i,j) = (A(i+1,j) - A(i,j))/delX;
        Ey(i,j) = (A(i,j+1) - A(i,j))/delY;
        Et(i,j) = (B(i,j)   - A(i,j))/delT;
    end
end
```

The diligent reader will notice that we do not compute the partial derivatives around the boundary points. This is because we have no macro-data about our image beyond what is given on the grid. So we assume that along the boundary

$$\frac{\partial E}{\partial x} = \frac{\partial E}{\partial y} = 0.$$

The things that must be furnished from the user are A, the first image, B, the second image, and the spatial and temporal differences.

Next these components are fed into an iterative method to compute (u, v).

```
function [U,V] = UVflow(Ex,Ey,Et,lambda,Kend)
[m,n] = size(Ex); % intake the dimensions
U = zeros(n,m);   % initialize (u,v) as zero matrices
V = zeros(n,m);
for k = 1:Kend
    for i = 2:m-1
        for j = 2:n-1
            Uavg = (U(i,j) + U(i+1,j) + U(i-1,j) +...
                                 U(i,j+1) + U(i,j-1))/5;
            Vavg = (V(i,j) + V(i+1,j) + V(i-1,j) +...
                                 V(i,j+1) + V(i,j-1))/5;
            P = Ex(i,j)*Uavg + Ey(i,j)*Vavg + Et(i,j);
            D = 5*lambda^2 + (Ex(i,j))^2 + (Ey(i,j))^2;
            U(i,j) = Uavg - (Ex(i,j)*P)/D;
            V(i,j) = Vavg - (Ey(i,j)*P)/D;
        end
    end
end
```

To generate the images from Appendix B we now show a completed algorithm from end-to-end. The major steps of the following code (written in MatLab) involve

- Define original image domain Ω
- Generate an image on that domain
- Specify the following parameters
  - the time interval

  - the values for Δx, Δy
  - the (u, v) that you would like to use

- Finally generate (or sample) the resulting images

Our implementation here should be fairly straightforward to anyone with an intermediate understanding of MatLab. But we have tried our best to comment each piece of the code to make implementation easier for the reader.

```
% Domain definition
a = -3;
b = 3;

% Number of sub-intervals
m = 100;

% Generating mesh
x = linspace(a,b,m+1);
y = linspace(a,b,m+1);
[x,y] = meshgrid(X,Y);
[alpha,beta] = size(x);

% Set the number of iterations
maxiter = 10;

% Prescribe the flow field
U = -y;
V = x;
```

```
A  = zeros(m+1,m+1);
A0 = zeros(m+1,m+1);

% Generate the image to be moved
for i = 1:m+1
    for j = 1:m+1
        if i < j
            A0(i,j) = .1;
        end
     if (x(i,j))^2+ + (y(i,j))^2 < 1^2
        A0(i,j) = 1;
        end
     if (x(i,j))^2 + (y(i,j))^2 < .5^2
        A0(i,j) = .5;
        end
    end
end
```

```
figure(2)
surf(x,y,A0,'Edgecolor','none')
view(0,90)
colormap(flipud(gray))
pause(0.01)

% Time step
dt = 0.05;

% Initial pixel grid
x0 = x;
y0 = y;

figure(2)
for iter = 1:maxiter
    for i = 1:m+1
        for j = 1:m+1
        % Moving old pixels
        % to new pixel location
x(i,j) = x0(i,j) + dt* U(i,j);
y(i,j) = y0(i,j) + dt* V(i,j);
      % Mapping intensities
      A(i,j) = A0(i,j);
        end
    end
    % Reassigning new pixels
    x0 = x;
    y0 = y;
    % Save the coordinates
    MovieY(:,:,iter) = y;
    MovieX(:,:,iter) = x;
    MovieA(:,:,iter) = A;
        % View the results
    surf(x,y,A,'Edgecolor','none')
    title(['iter = ',num2str(iter)])
    axis([-2,2 -2 2])
    view(0,90)
    colormap(flipud(gray))
    pause(0.1)

    % Reinitializing the
    A = zeros(m+1,m+1);
end
for i = 1:maxiter-1
subplot(3,3,i)
surf(MovieX(:,:,i),...
      MovieY(:,:,i),...
           MovieA(:,:,i),...
           'Edgecolor','none')
view(0,90)
axis([-2,2 -2 2])
end
```

We now furnish the code for the pixel movement and (u, v) update method. In this code we intake an MRI (this is read-in as a DICOM file which is then converted to a data type double for manipulation), then generate the pair (u, v) in between the two input images. This information is held for later use. We then evolve in time.

Within each step we update our grid and use the advection equation to update pixels at each iteration. We also modify our parameter λ to better reflect the type of motion that is begin seen in between each iterate. As a reminder to the reader, a $\lambda = 0$ means abrupt motion and a larger value of λ means that our motion is smoother.

We are also smoothing our flow field, this is to make sure that regions move in each time step with similar motion.

```
% Define a grid to work on
a = -5;
b = 5;


% Generate points
[x,y] = meshgrid(X,Y);
[alpha,beta] = size(x);


% Decide how many iterations
maxiter = 20;


% Image slice 1
A1 = dicomread('IM_0008');
% Image slice 2
A2 = dicomread('IM_0009');

% Convert data types
A1 = im2double(A1);
A2 = im2double(A2);
A = zeros(alpha,beta);

figure
subplot(1,2,1)
surf(x,y,flip(A1),...
        'Edgecolor','none')
view(0,90)
colormap(flipud(gray))
subplot(1,2,2)
```

```
surf(x,y,flip(A2),...
        'Edgecolor','none')
view(0,90)
colormap(flipud(gray))

% Image pair data
delx =  .6875;
dely =  .6875;
delt =     4.4;

% Compute Spatial derivatives
[Ex,Ey,Et] = ...
   Compute_ExEyEt(A1,A2,...
             delx,dely,delt);

%Compute U,V
[U,V] = ...
    UVflow(Ex,Ey,Et,10^(-5),15);

% Orientation and scaling
% U = flip(U)/10;
% V = flip(V)/10;

% Make copies of original data
U0 = U;
V0 = V;
Ex0 = Ex;
Ey0 = Ey;
Et0 = Et;
% Initial pixel grid
x0 = x;
y0 = y;
time = 0;


figure
for iter = 1:maxiter
%     A = ones(alpha,beta);
    for i = 1:m+1
        for j = 1:m+1
            % Moving the old
            % pixel location

    x(i,j) = x0(i,j) + time*U(i,j);
    y(i,j) = y0(i,j) + time*V(i,j);

            % Mapping the intensities
   A(i,j) = .5*A(i,j) + .5*A1(i,j) ...
          - .5*time*(Ex0(i,j)*U0(i,j)...
          + Ey0(i,j)*V0(i,j));
        end
    end

    % Reassigning new pixels
    x0 = x;
    y0 = y;

        % Display images
    surf(x,y,flip(A),...
    'Edgecolor',...
           'none')
    view(0,90)
    colormap(flipud(gray))
    axis([a,b a b])
    pause(0.2)

        % Update time
    time = time + .22;

    % Recompute spatial data
```

```
    [Ex,Ey,Et] = Compute_ExEyEt(A,A2,...
           delx,dely,delt-.22*time);

       % Recompute U,V
       [U,V] = UVflow(Ex,Ey,Et,...
             10^(-2.2),5);

       % Smooth U,V
       U = imgaussfilt(U,10);
       V = imgaussfilt(V,10);
       % Orientation
       U = flip(U);
       V = flip(V);
end
```

The last code that we will give is the one for the Euler vs Non-standard Euler method that produce Figures 2.28 and 2.29.

```
a = -5;
b = 5;
m = 319;

[x,y] = meshgrid(X,Y);
[alpha,beta] = size(x);
maxiter = 20;
```

```
A1 = dicomread('IM_0008');
% image slice 1
A2 = dicomread('IM_0009');
% image slice 2
A1 = im2double(A1);
% convert to type double
A2 = im2double(A2);
% convert to type double
```

```
AE    = zeros(alpha,beta);
ANs   = zeros(alpha,beta);
AE1   = zeros(alpha,beta);
ANs1 = zeros(alpha,beta);

% gamma = max(max(A1));


 delx =  .6875;
% x - spacing in the image
 dely =  .6875;
% y - spacing in the image
 delt =     4.4;
% t - spacing in the image
 [Ex,Ey,Et] = ...
     Compute_ExEyEt(...
     A1,A2,delx,dely,delt);
 [U,V] = ...
     UVflow(Ex,Ey,Et,10^(-5),15);

U0 = U;
V0 = V;
Ex0 = Ex;
Ey0 = Ey;
Et0 = Et;
Ex1 = Ex;
Ey1 = Ey;
Ex2 = Ex;
```

```
Ey2 = Ey;


U1 = U;
U2 = U;
V1 = V;
V2 = V;


% Initial pixel grid
x0 = x;
y0 = y;

x0E = x;
y0E = y;

x0Ns = x;
y0Ns = y;

q = .5;
time = 0;
P  = .9;
Q = 1 - P;

 figure
 pause(3)
 subplot(1,3,1)
 surf(x0E,y0E,...
flip(A2),'Edgecolor','none')
view(0,90)
colormap(flipud(gray))
axis([a,b a b])


for iter = 1:maxiter
     for i = 1:m+1
          for j = 1:m+1

 xE(i,j) = x0E(i,j) + time*U1(i,j);
 yE(i,j) = y0E(i,j) + time*V1(i,j);

 xNs(i,j) = x0Ns(i,j)+...
  (1-exp(-q*time))*U2(i,j)/q;
 yNs(i,j) = y0Ns(i,j)+...
  (1-exp(-q*time))*V2(i,j)/q;

AE(i,j)=  P*AE(i,j)+Q*A1(i,j)  -...
  Q*time*(Ex0(i,j)*U0(i,j)  +...
  Ey0(i,j)*V0(i,j));
ANs(i,j)= P*ANs(i,j)+Q*A1(i,j)-...
  Q*time*(Ex0(i,j)*U0(i,j)  +...
  Ey0(i,j)*V0(i,j));


          end
     end
X1Move(:,:,iter) = xE;
Y1Move(:,:,iter) = yE;
AEMove(:,:,iter) = AE;

X2Move(:,:,iter) = xNs;
Y2Move(:,:,iter) = yNs;
ANsMove(:,:,iter) = ANs;

 % Reassigning new pixel as
 % the old pixels
 x0E = xE;
 y0E = yE;

 x0Ns = xNs;
 y0Ns = yNs;

 subplot(1,3,2)
 surf(xE,yE,flip(AE),...
 'Edgecolor','none')
 view(0,90)
 colormap(flipud(gray))
 axis([a,b a b])

 subplot(1,3,3)
 surf(xNs,yNs,flip(ANs),...
 'Edgecolor','none')
 view(0,90)
```

```
colormap(flipud(gray))
axis([a,b a b])

pause(0.1)

time = time + .22;

[Ex1,Ey1,Et1] = ...
Compute_ExEyEt(AE,A2,...
delx,dely,delt-.22*iter);
[U1,V1] = UVflow(Ex1,Ey1,Et1...
,10^(-2.2),5);

[Ex2,Ey2,Et2] = ...
Compute_ExEyEt(ANs,A2,...
delx,dely,delt-.22*iter);
[U2,V2] = UVflow(Ex2,Ey2,Et2,...
10^(-2.2),5);

if iter < maxiter/2
    U1 = imgaussfilt(U1,10);
    V1 = imgaussfilt(V1,10);
    U2 = imgaussfilt(U2,10);
    V2 = imgaussfilt(V2,10);
else
    U1 = imgaussfilt(U1,15);
    V1 = imgaussfilt(V1,15);
    U2 = imgaussfilt(U2,15);
    V2 = imgaussfilt(V2,15);
end
U1 = flip(U1);
V1 = flip(V1);
U2 = flip(U2);
V2 = flip(V2);
end

figure

subplot(4,2,1)
surf(X1Move(:,:,1),...
Y1Move(:,:,1)...
,flip(AEMove(:,:,1)),...
'Edgecolor','none')
view(0,90)
colormap(flipud(gray))
axis([a,b a b])

subplot(4,2,2)
surf(X2Move(:,:,1),...
Y2Move(:,:,1),...
flip(ANsMove(:,:,1)),...
'Edgecolor','none')
view(0,90)
colormap(flipud(gray))
axis([a,b a b])
```

```
subplot(4,2,3)
surf(X1Move(:,:,5),...
Y1Move(:,:,5),...
flip(AEMove(:,:,5)),...
'Edgecolor','none')
view(0,90)
colormap(flipud(gray))
axis([a,b a b])

subplot(4,2,4)
surf(X2Move(:,:,5),...
Y2Move(:,:,5),...
flip(ANsMove(:,:,5)),...
'Edgecolor','none')
view(0,90)
colormap(flipud(gray))
axis([a,b a b])

subplot(4,2,5)
surf(X1Move(:,:,10),...
Y1Move(:,:,10),...
flip(AEMove(:,:,10)),...
'Edgecolor','none')
view(0,90)
colormap(flipud(gray))
axis([a,b a b])

subplot(4,2,6)
surf(X2Move(:,:,10),...
Y2Move(:,:,10),...
flip(ANsMove(:,:,10)),...
'Edgecolor','none')
view(0,90)
colormap(flipud(gray))
axis([a,b a b])

subplot(4,2,7)
surf(X1Move(:,:,20),...
Y1Move(:,:,20),...
flip(AEMove(:,:,20)),...
'Edgecolor','none')
view(0,90)
colormap(flipud(gray))
axis([a,b a b])

subplot(4,2,8)
surf(X2Move(:,:,20),...
Y2Move(:,:,20),...
flip(ANsMove(:,:,20)),...
'Edgecolor','none')
view(0,90)
colormap(flipud(gray))
axis([a,b a b])
```

APPENDIX B

Explicit Motion Examples

We have the following examples

$$u = -y$$

$$v = x;$$

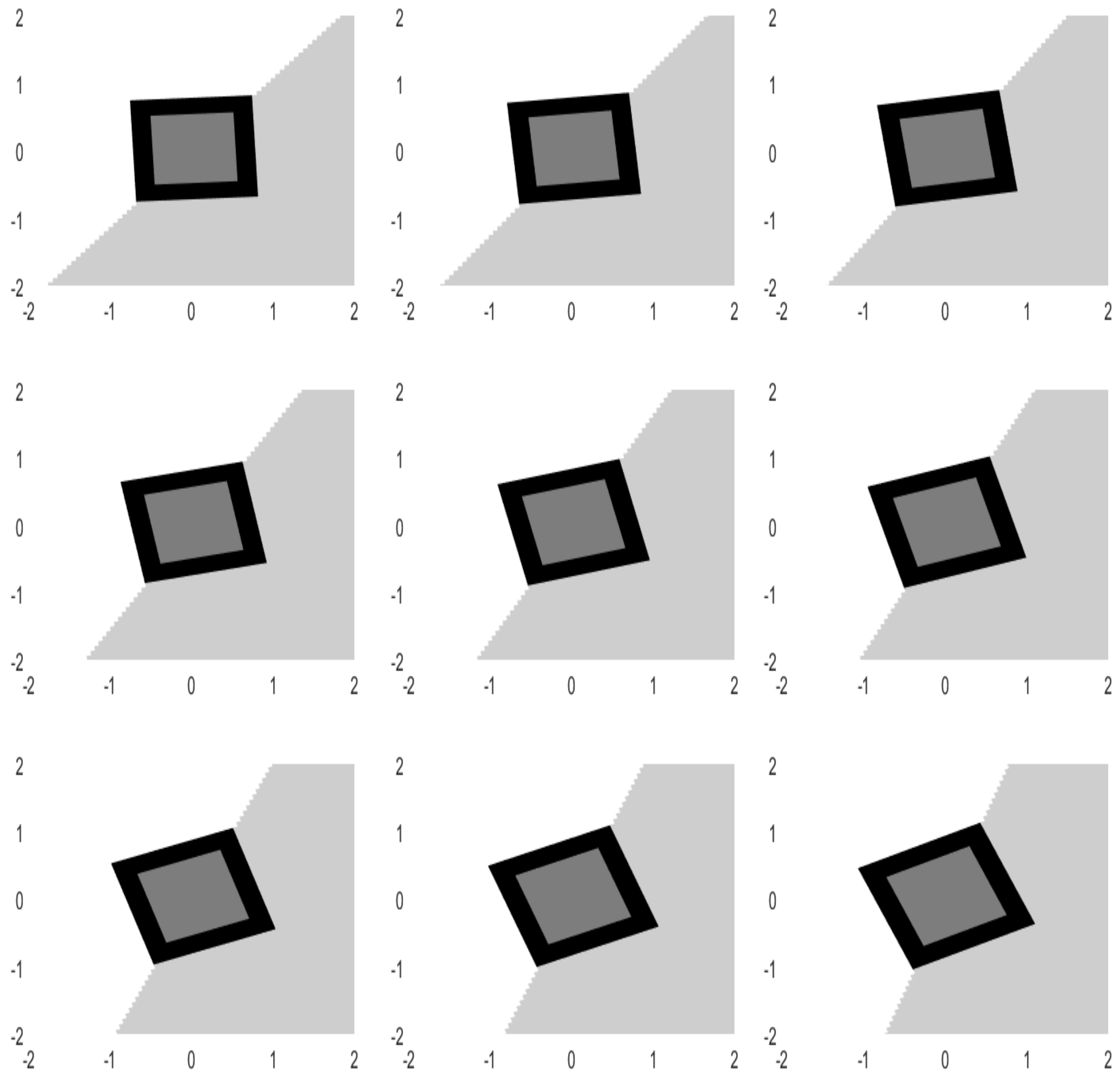

Figure B.1. Rotating an image.

$$u = 2$$

$$v = 0$$

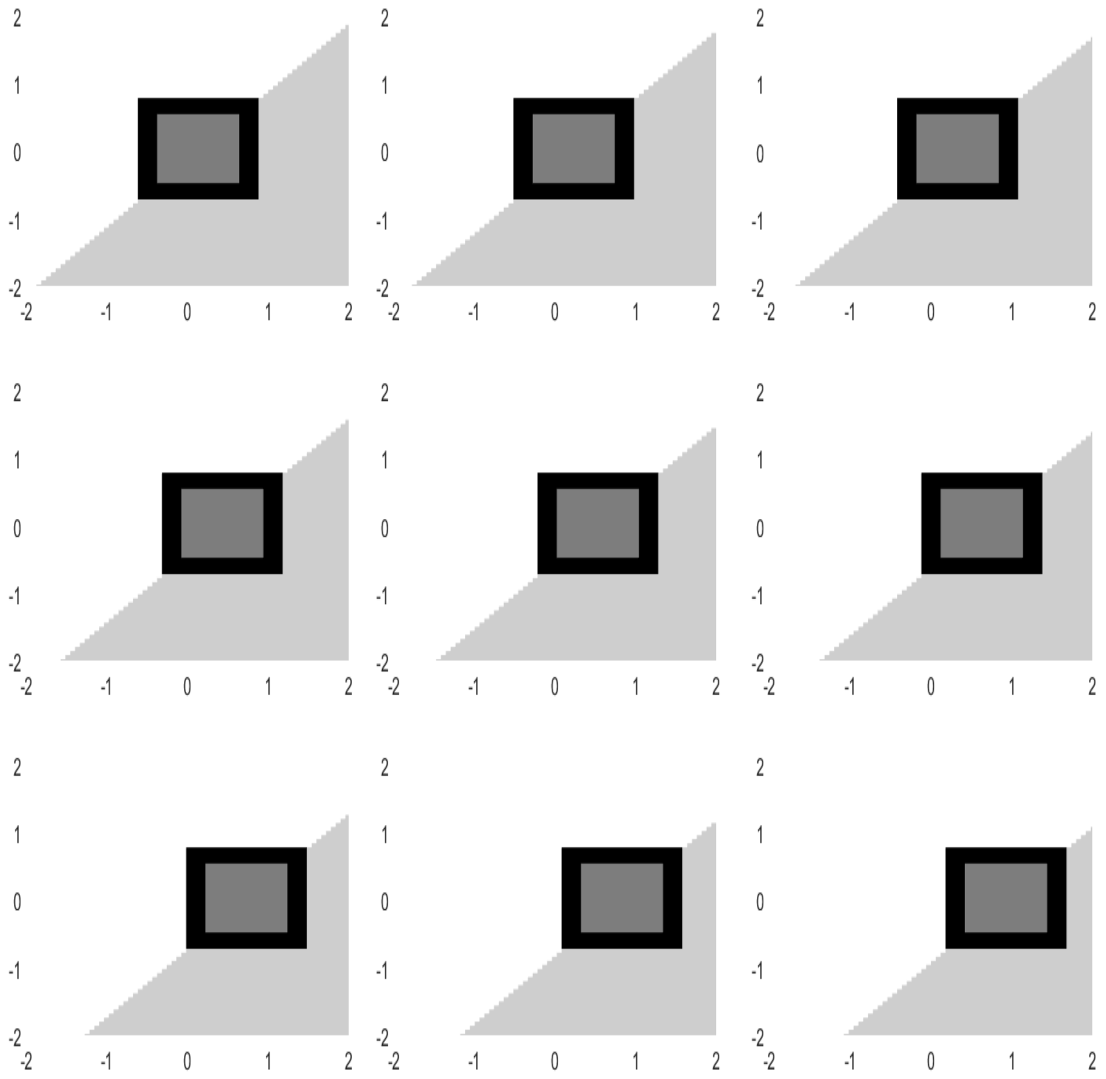

Figure B.2. Motion with constant field.

$$u = -.5 \cdot \sin(2 * y)$$

$$v = .5 \cdot \sin(2 * x)$$

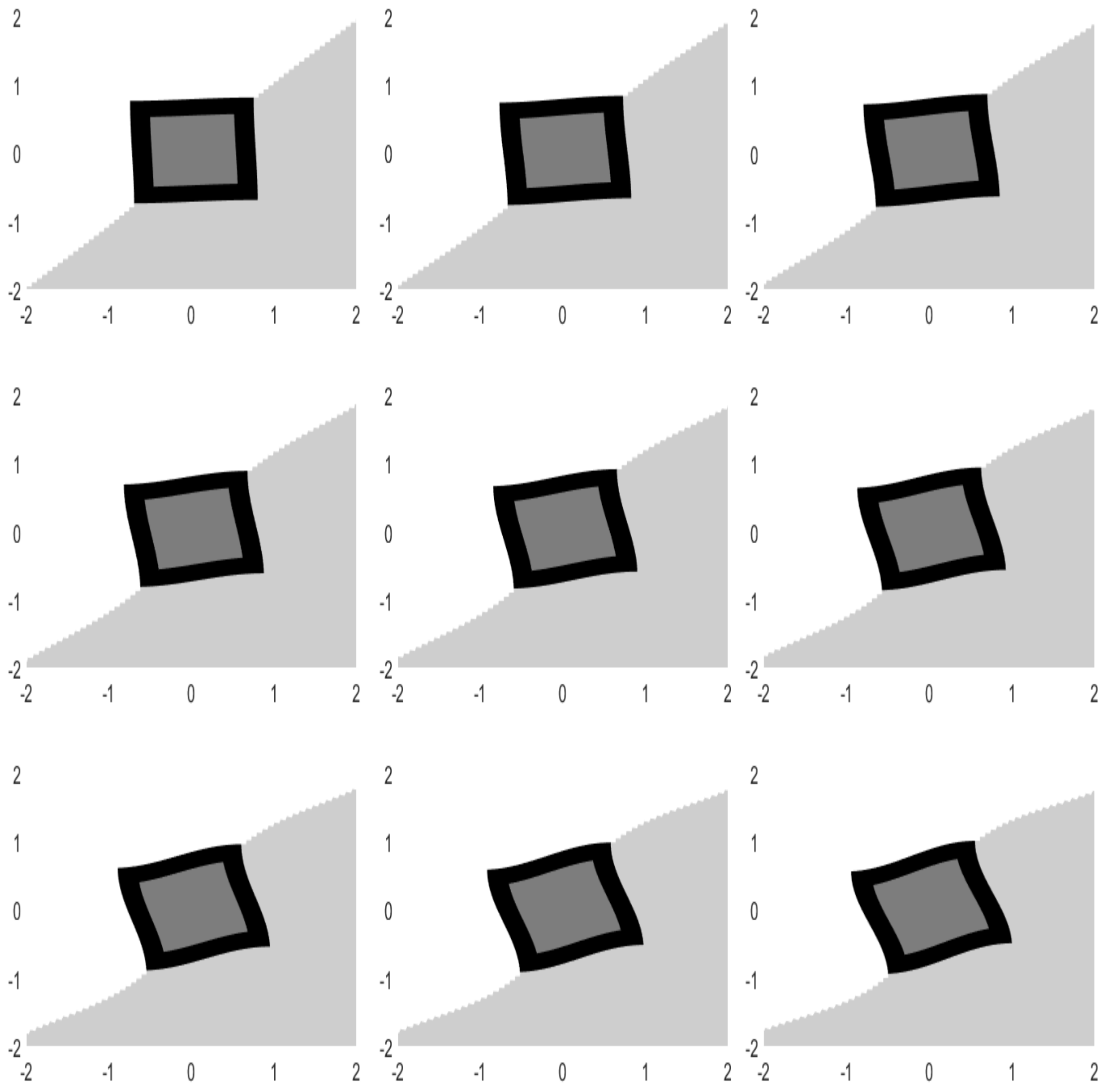

Figure B.3. Using the paired sine functions.

$$u = 1.5 \cdot x$$

$$v = 1.5 \cdot y$$

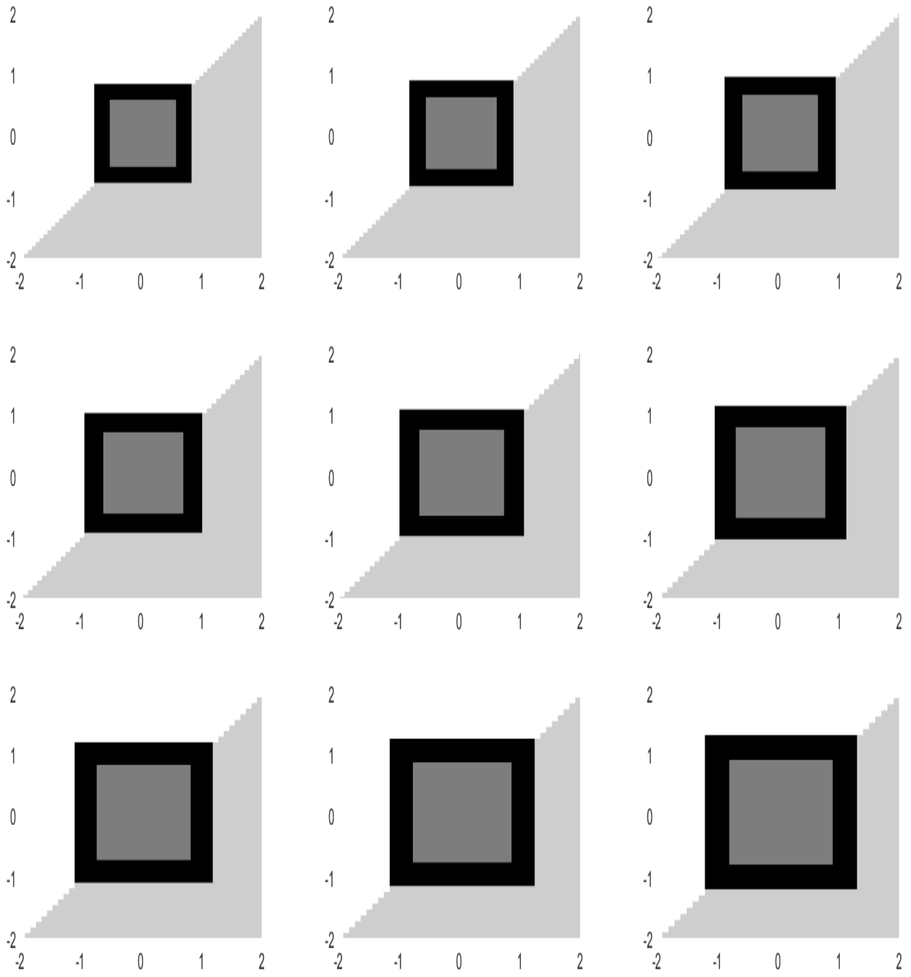

Figure B.4. Expansion of the image domain.

$$u = -.5 \cdot x$$

$$v = -.5 \cdot y$$

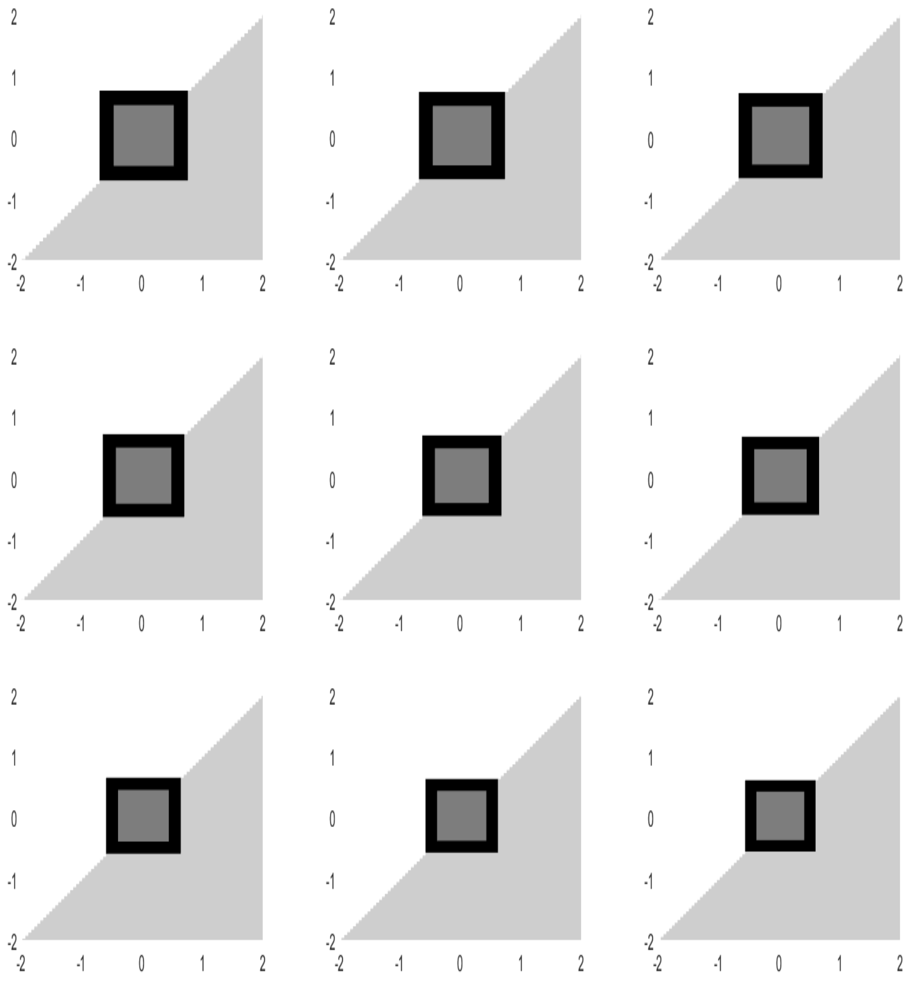

Figure B.5. Contraction of entire image domain.

$$u = x$$

$$v = -y$$

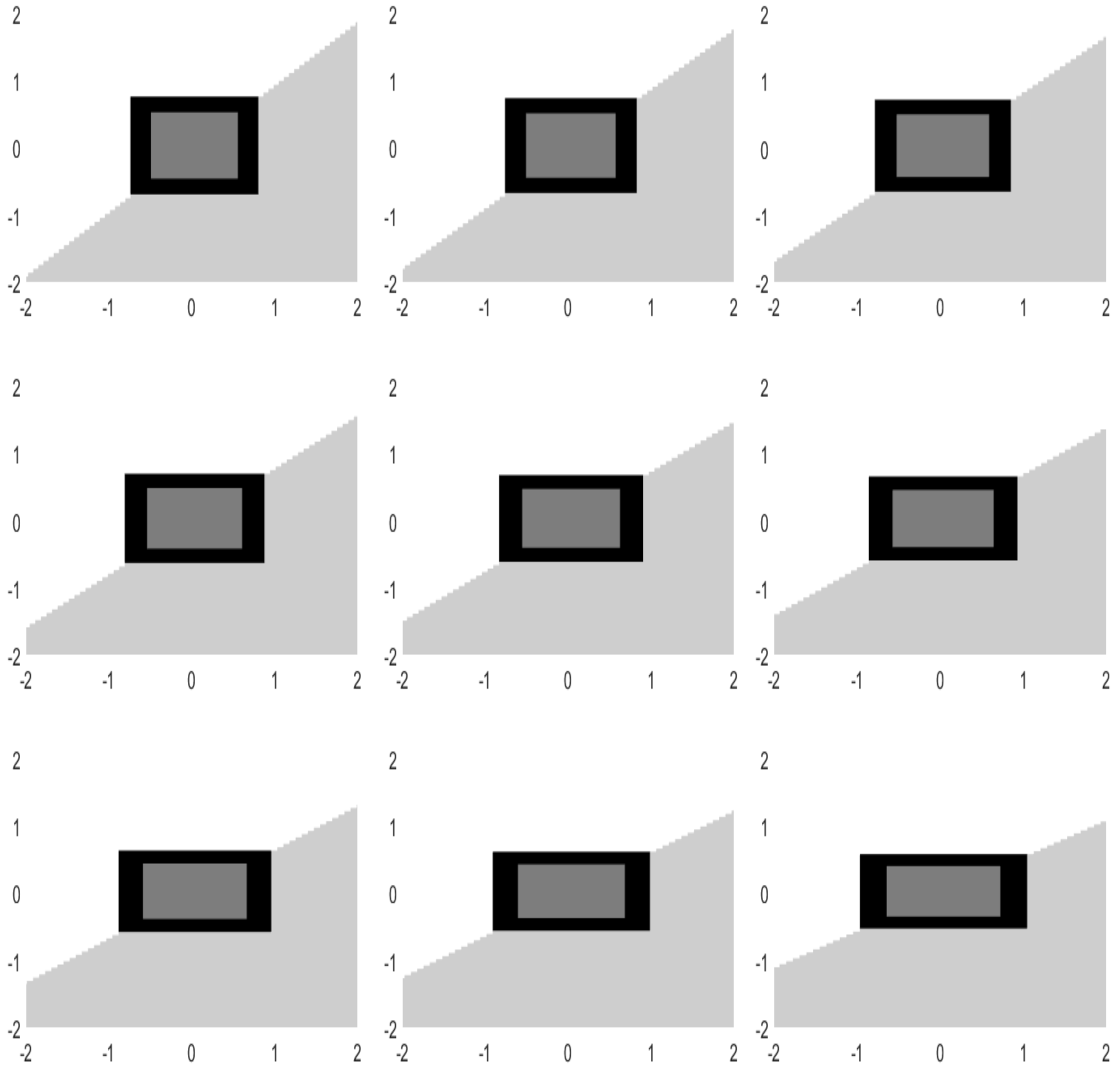

Figure B.6. Squeezing of the entire image.

$$u = x + y$$

$$v = 3x - y$$

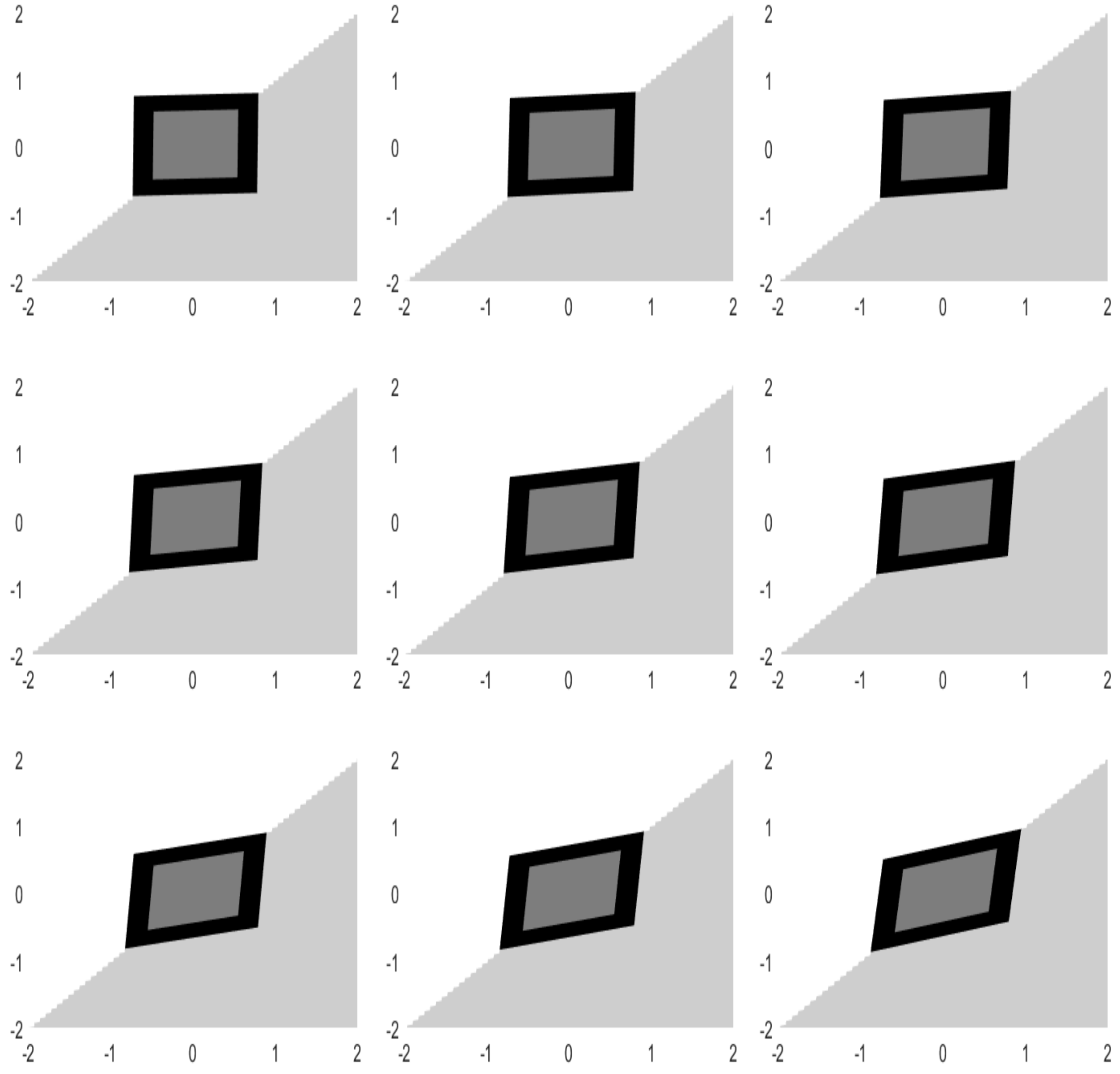

Figure B.7. Evolution of the image dictated by the system above.

$$u = y$$

$$v = 0$$

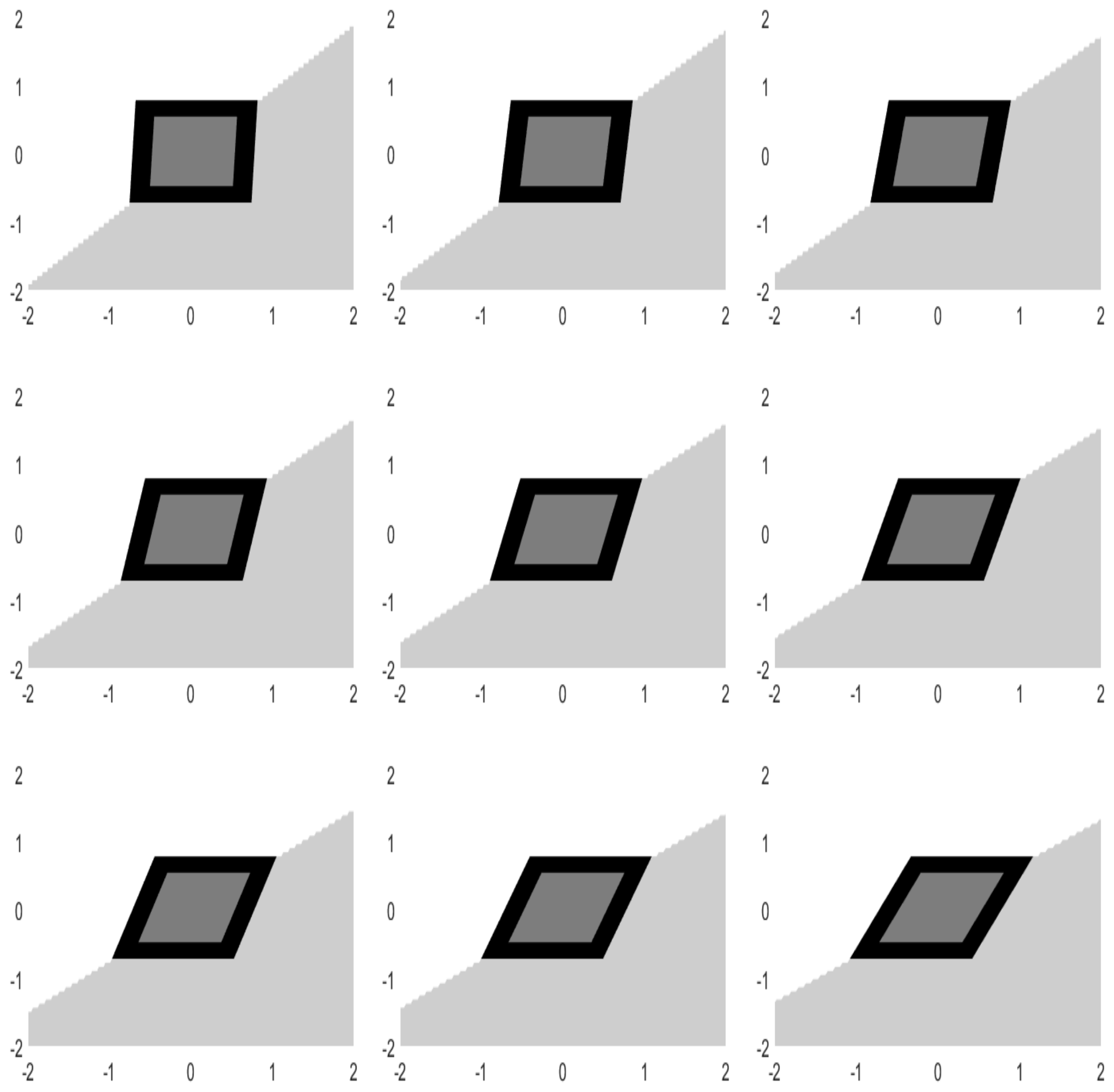
Figure B.8. Shear motion.

REFERENCES

[1] Aubert, G., Deriche, R., & Kornprobst, P., Computing Optical Flow Via Variational Techniques, *SIAM Journal of Applied Mathematics*, Vol. 60, (1999), pages 156-182.

[2] Bertero, M., Poggio, T., & Torre, V., Ill-Posed Problems In Early Vision, *Proceedings of the IEEE*, Volume 76, No. 8, (1988), pages 869-889, doi:10.1109/5.5962.

[3] Bour, P. & Cribelier, E., Crowd Behavior Analysis From Fixed and Moving Cameras, *In Alameda-Pineda & E. Ricci, Multimodal Behavior Analysis in the Wild*, Academic Press, (2019), pages 289-322.

[4] Bruhn, A. & Christoph, S., Lucas & Kanade Meets Horn & Schunck: Combining Local and Global Optic Flow Methods, *International Journal of Computer Vision,* Volume 61, (2005), pages 211-231.

[5] Bruhn, A. & Kohlberger, T., A Multigrid for Real-Time Motion Computation with Discontinuity-Preserving Variational Methods, *International Journal of Computer Vision*, Volume 70, (2006), 257-277, https://doi.org/10.1007/s11263-006-6616-7.

[6] Ciresan, D., Giusti, A., Gambardella, L., & Schmidhuber, J., Deep Neural Networks Segment Neuronal Membranes in Electron Microscopy Images, *NIPS'12: Proceedings of the 25th International Conference on Neural Information Processing Systems*, Volume II, (2012), pages 2843-2851.

[7] Dimitrov, D. & Kojouharov, H., Nonstandard Finite-Difference Schemes For General Two-Dimensional Autonomous Dynamical Systems, *Applied Mathematics Letters*, Volume 18, Issue 7, (2004), pages 769–774.

[8] Galic, S. & LonCaric, S., Spatio-Temporal Image Segmentation Using Optical Flow and Clustering Algorithm, *IWISPA 2000 Proceedings of the First International Workshop on Image and Signal Processing and Analysis in conjunction with 22nd International Conference on Information Technology Interfaces*, (2000), pages 63-68, doi: 10.1109/ISPA.2000.914892.

[9] Peddie, W., Helmholtz's Treatise on Physiological Optics (Trans. from 3rd German ed.), *Nature*, Volume 116, (1925), pages 88-89, https://doi.org/10.1038/116088a0 .

[10] Horn, B. & Schunck, B., Determining Optical Flow, *Artificial Intelligence*, Volume 17, Issues 1-3, (1981), pages 185-203.

[11] Jia, F., Liu, J., & Tai X., A Regularized Convolutional Neural Network for Semantic Image Segmentation, *Analysis and Applications*, (2020), pages 1-19, https://doi.org/10.1142/S0219530519410148.

[12] Kale, K., Pawar, S., & Dhulekar, P., Moving Object Tracking Using Optical Flow and Motion Vector Estimation, *IEEE - 2015 4th International Conference on Reliability, Infocom Technologies and Optimization (ICRITO) (Trends and Future Directions)*, (2015), pages 1-6.

[13] Khatri, G. & Carmel, M., Postoperative Imaging After Surgical Repair for Pelvic Floor Dysfunction, *RadioGraphics*, Volume 36, Issue 4, (2016), pages 1233-1256, https://doi.org/10.1148/rg.2016150215.

[14] Long, J., Shelhammer, E., & Darrell, T., Fully Convolutional Networks for Semantic Segmentation, *arXiv:1411.4038*, (2014).

[15] Lucas, B.D. & Kanade, T., An Iterative Image Registration Technique with an Application to Stereo Vision, *IJCAI'81: Proceedings of the 7th International Joint Conference on Artificial Intelligence*, Volume 2, (1981), pages 674-679.

[16] Mayer, N., Ilg, E., & Fischer, P., What Makes Good Synthetic Training Data for Learning Disparity and Optical Flow Estimation?, *International Journal of Computer Vision*, Volume 126, (2018), pages 942-960.

[17] Mukherjee, K. & Mukherjee A., Joint Optical Flow Motion Compensation and Video Compression Using Hybrid Vector Quantization, *IEEE - Proceedings DCC'99 Data Compression Conference (Cat. No. PR00096)*, (1999), pages 541.

[18] Onal, S., Automated Localization and Segmentation of Pelvic Floor Structures on MRI to Predict Pelvic Organ Prolapse, *PhD Thesis - University of South Florida: Scholar Commons*, Graduate Theses and Dissertations, (2014), https://scholarcommons.usf.edu/etd/5288.

[19] Poggop, T., Torre, V., & Koch, C., Computational Vision and Regularization Theory, *Nature*, Volume 317, (1985), pages 314- 319.

[20] Ronneberger, O., Fischer, P., Brox, T., U-Net: Convolutional Networks for Biomedical Image Segmentation, *arXiv:1505.04597*, (2015).

[21] Roy, S., Reconstruction of a Class of Fluid Flows by Variational Methods and Inversion of Integral Transforms in Tomography, *PhD Thesis - Bangalore: TATA Institute of Fundamental Research Center for Applicable Mathematics*, (2015).

[22] Shorten, C. & Khoshgoftaar, T., A Survey on Image Data Augmentation for Deep Learning, *Journal of Big Data*, Volume 6, (2019), page 60, https://doi.org/10.1186/s40537-019-0197-0.

[23] Tarnec, L., & Destrempes, F., A Proof of Convergence of the Horn and Schunck Optical Flow Algorithm in Arbitrary Dimension, *SIAM Journal of Imaging Science*, Volume 7, Issue 1, (2014), pages 277 - 293, doi: 10.1137/130904727.

[24] Vedaldi, A., Lenc, K. & Gupta, A., MatConvNet: Convolutional Neural Networks for Matlab, *MM '15: Proceedings of the 23rd ACM International Conference on Multimedia*, (2015), pages 689-692, doi.org/10.1145/2733373.2807412.

BIOGRAPHICAL STATEMENT

John Montalbo was born in San Antonio, Texas in 1986 to Juan and Renee Montalbo. He received a B.Sc. from the University of Texas Pan American in May 2014 and an M.Sc. degree from the University of Texas Rio Grande Valley in May of 2016, both in Mathematics. His main focus of research is mathematical imaging problems, specifically how we develop new ways to solve practical problems in imaging science. His masters thesis covered how the field of compressive sensing could be used in conjunction with radar imaging to solve undersampling and compression problems.