An Optical Flow based Method for the Segmentation of Image Sequences

Espen Johansen Velsvik

Master of Science in Physics and Mathematics
Submission date: January 2017
Supervisor: Markus Grasmair, IMF
Co-supervisor: Sondre Andreas Engebråten, Forsvarets Forskningsinstitutt

Norwegian University of Science and Technology
Department of Mathematical Sciences

Abstract

The segmentation of motion in an image sequence is an important task in many computer vision applications. This thesis presents the theory and numerical algorithm for the detection of object boundaries of moving objects in an image sequence, using optical flow to estimate movement and active contours to locate flow boundaries. Three different methods for regularizing optical flow are presented and analyzed based on the application of this flow field in a segmentation framework. This active contours framework is formulated as the computation of geodesic curves in a Riemannian space defined by the gradients of the optical flow field. We will use a level set function to describe and evolve these contours, effectively incorporating structural information of the flow field in the level sets of this function. A Mumford-Shah segmentation is used to extract this information from the flow field. The boundary of this segmentation is represented by a cubic B-spline. This leads to a combined evolution of the level set function and the B-spline curve. The performance of the algorithm is validated on two real-world data sets, provided by the Norwegian Defence Research Establishment (FFI).

Samandrag

Rørslesegmentering i ein biletesekvens er ei viktig oppgåve i mange applikasjonar innan maskinsyn. Denne avhandlinga presenterer teorien og den numeriske framgangsmåten for å detektere omrisset av objekt i rørsle i ein biletesekvens. Dette vert gjort ved å bruke optisk flyt for å estimere rørsle og aktive konturar for å lokalisere flytgrenser. Vi vil samanlikne tre regulariseringsmåtar for flytestimering basert på bruken av dette flytfeltet i eit segmenteringsrammeverk. Segmenteringsoppgåva er formulert som ei lengdeminimering i eit Riemannsk rom, definert av gradientane til det optiske flytfeltet. For å beskrive utviklinga av dei aktive konturane, vil vi bruke ein nivåmengde-funksjon, noko som fører til at informasjon om strukturen til det optiske flytfeltet vert integrert i nivåmengdene til denne funksjonen. Denne strukturinformasjonen vert henta ut ved å utføre ein Mumford-Shah-segmentering av nivåmengde-funskjonen. Segmenteringsgrensene for denne prosessen vert beskrivne av ein kubisk B-spline. Dette fører til ei kombinert utvikling av nivåmengde-funksjonen og B-spline-kurven. Den numeriske algoritma er testa på to røynlege datasett, innsamla av Forsvarets Forskningsinstitutt (FFI).

Preface

This master thesis completes my studies at the Norwegian University of Science and Technology (NTNU) in the field of applied mathematics. The work was carried out during the spring and fall semesters of 2016 at the Department of Mathematical Science in Trondheim, Norway.

My sincere gratitude and thanks goes to my supervisor, prof. Markus Grasmair, for sharing his knowledge and taking the time for weekly discussions. His contribution to this thesis has been invaluable.

I also would like to thank the Norwegian Defence Research Establishment (FFI), for providing me with datasets for the thesis. In particular, thanks is due to my contacts, Sondre Andreas Engebråten and Trym Vegard Haavardsholm.

For her moral support, patience and continuous optimism, I thank my significant other, Jorid.

Espen Johansen Velsvik
Trondheim, January 2017

Contents

List of Figures

List of Abbreviations

AC	active contours
BCA	brightness constancy assumption
CCTV	closed-circuit television
DFM	Drone flying over man
FD	flow driven
GCA	gradient constancy assumption
HS	Horn and Schunck
ID	image driven
MS	Mumford-Shah
OF	optical flow
PDE	partial differential equation
TV	total variation
UAV	unmanned aerial vehicle
WMF	Walking man in forest
WMFL	Walking man in forest - Left
WMFR	Walking man in forest - Right

1 Introduction

The relative motion of objects is important in how we perceive the world. Humans are naturally good at registering movement and at inferring aspects from this movement. From the movement of a car, a person would be able to reason from what direction it came, and where it is headed. Motion is also an essential element of social interaction, and actions like waving, nodding and bowing all have different social interpretation. We are also able to distinguish between motion patterns, say for example when meeting another car while driving. Due to the car's movement, the scenery looks like it is moving, but one can still see the movement of the other car, which exhibits a different motion pattern than the background. This ability of distinguishing between moving objects is called motion segmentation.

The segmentation of motion is a key feature in many applications and is an important step in various computer vision tasks such as action recognition, movement tracking and motion scene analysis. An example of such an application is the use of motion segmentation in surveillance. Surveillance cameras can be stationary cameras like the CCTV cameras found in most public areas, or they could be moving cameras, for instance a camera mounted to a flying drone, also called an UAV. Motion segmentation is used to detect moving objects in the surveillance video, and also to track the movement of these objects.

1.1 Optical flow

The process of motion segmentation involves both the detection of apparent motion, and the localization and representation of the boundaries of this motion. One way of detecting motion in an image is to look at the movement of each point from one frame to the next, and try to map the position of each point in the first frame to a position in the next frame by some translation. For each point, this mapping results in a vector that yields the movement of this point. This vector field is called the optical flow.

Let $\Omega \in \mathbb{R}^2$ be the image plane, and let $f(x,t) \in \mathbb{R}$ be the brightness pattern or grayscale value of some image sequence at $x \in \Omega$ and some time $t \in [0,T]$. The optical flow $\boldsymbol{w}(x,t) \in \mathbb{R}^2$ yields the relative velocity at position $x \in \Omega$ and time $t \in [0,T]$. This vector field is found by looking at the change in brightness between consecutive frames in the image sequence. More precisely, one makes the assumption that a moving object has a constant brightness along its trajectory. This assumption was first introduced by Horn and Schunck, and it is most often referred to as *the brightness constancy assumption* (BCA). In the example of the moving car, this assumption says that the car does not change its appearance when it moves. Using the BCA one can derive an equation that must hold for the optical flow field. This equation is called *the optical flow equation* and will be derived in section 2.1.

The optical flow equation is known to be ill-posed, and does not admit a unique solution. Solving this equation requires the use of regularization in the form of an added smoothness constraint. This constraint, called *the spatial coherence assumption*, states that points on the same object can not move independently. One would like the vector field to be smooth within objects, with discontinuities only occurring along the edges of regions exhibiting different motion patterns. For the moving car, this assumption imposes the constraint that the different parts of the car have to move in approximately the same direction.

1.2 Motion segmentation

The optical flow field provides a velocity field at every point in the image plane Ω, but does not yield an explicit description of the moving regions in the image. This is the aim of the motion segmentation process. More accurately, one tries to divide the image into parts that describe moving objects. This problem amounts to correctly localizing the boundaries of the moving objects in the image scene, which will correspond to discontinuities in the optical flow field.

This thesis presents a geodesic *active contours* (AC) approach to segmentation. The theory of geodesic active contours is due to Caselles et al. [7], and is based on the classical method of *snakes*, first introduced by Kass et al. [15] in an image segmentation framework. The method aims to deform a closed curve C, a contour, to lock onto boundaries in the image f. The idea is to associate an energy with the closed curve, and use this energy to guide the search for image edges. These image edges can be identified by using a so called edge detector g, a function which intention is to stop the evolution of the curve at these edges. The *geodesic* framework reformulates this energy based active contours model as a search for a curve of minimal length in a Riemannian space whose metric is defined by the edge detector g. Now, the segmentation of motion is not so much concerned with finding image edges, but flow edges. Thus, we propose to use an edge detector for flow edges instead of image edges. This changes the metric of the Riemannian space from being defined by the content of one single image to being defined by the content of two consecutive images in an image sequence, and in particular the gradients of each component in the optical flow field.

For representing these active contours a level set function will be used. The level set method was developed in the late 1980s by Osher and Sethian [24], and has been a widely used representation for closed curves in image processing. The method assumes that the active contours can be described by the level sets of a function φ. This representation is convenient as it allows for intrinsic geometric properties of the contour to be easily determined. In addition, the function representation facilitates the use of simple methods for numerical approximation [28, p. 13]. As proposed by Fusch et al. [12], a Mumford-Shah (MS) type segmentation will be employed to make use of information from all level sets of φ and not just the level set describing the active contour. The curve describing the segmentation boundary of this process is parametrized by using a periodic cubic B-spline, leading to a combined evolution of the level set function and the spline curve.

1.3 Thesis outline

The framework in this paper can be divided into two parts, the optical flow estimation and the active contours segmentation. Chapter 2 presents an outline of these two frameworks in order to introduce concepts that will be relevant in the further analysis. Chapter 3 gives a detailed description of optical flow estimation by variational methods. The chapter starts out with a presentation of alternative assumptions for the optical flow equation along with remedies for increasing the robustness of the model in section 3.2. In section 3.3 we show three different methods of regularization, namely the classical method of Horn and Schunck (HS) [13], the anisotropic image driven (ID) regularization of Nagel and Enkelmann [22] and the flow driven (FD) regularization of Shulman and Herve [29].

The active contours segmentation is presented in chapter 4. Section 4.1 describes the classical method of snakes, which was the first model for energy based active contours. The section continues by showing an alternative way of expressing the snake model by formulating the active contours segmentation as a problem of finding geodesics in a Riemannian space. The details of the level set method is shown in section 4.2 along with the evolution of the level sets of φ given by the gradient flow. Chapter 5 presents a modified Mumford-Shah functional to segment the level set function along with the details regarding the B-spline formulation of the segmentation boundary.

In chapter 6 we show some of the numerical details and implemented features. In particular, we present a splitting algorithm used to handle self-intersections in the evolving curve. The results are shown in chapters 7 and 8. Chapter 7 starts by comparing the results from the three different regularization methods for optical flow. Then follows a demonstration of the combined algorithm for segmenting flow fields. Real-world examples are presented in chapter 8, showing the performance of the segmentation algorithm for data sets provided by the *Norwegian Defence Research Establishment*.

2 Segmentation of optical flow

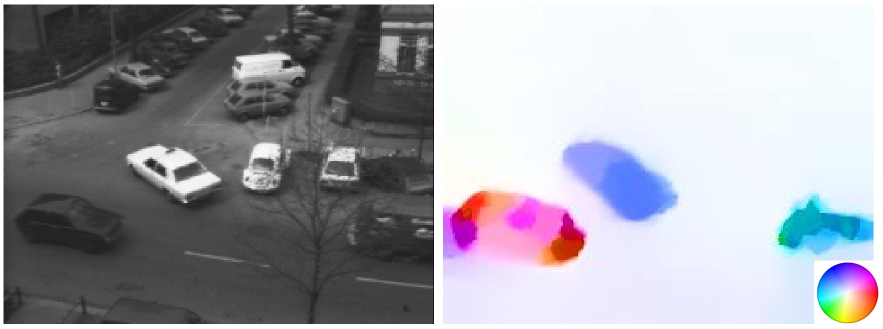
We will start our treatment of optical flow segmentation by providing the reader with an outline of the main topics in this thesis, namely optical flow estimation and segmentation.

2.1 The optical flow equation

The optical flow (OF), or optic flow, is a flow field that describes apparent motion in an image scene. This apparent motion may be the result of actual motion between the observer and some objects. An example of such a flow field is shown in figure 2.1, with the flow field of three moving cars. Alternatively the apparent motion may be induced by illumination changes in the image. Conversely, motion need not result in an optical flow. Consider rotating a uniform sphere around its center. If the surface of the sphere is uniform, both in color and shape, this rotation is not observable.

These examples illustrate that there need not be a relation between the optical flow and the motion of objects. To be able to associate the optical flow with motion, one must first assume that motion always corresponds to a visible optical flow. Additionally, one needs to assume that there are no illumination changes in the image scene. This assumption is artificial and is often violated in real life. Nevertheless, we will assume this in able to make the connection between optical flow and the motion of objects.

To estimate the optical flow from an image sequence we need to make some assumptions about how objects appear when they move through an image scene. The optical flow pioneers Horn and Schunck introduced what



(a) Some image scene (b) Example of estimated optical flow

Figure 2.1: The left image shows one image in the Hamburg taxi sequence. The right image shows an example of some optical flow estimation. The colorwheel indicates the direction of the movement.

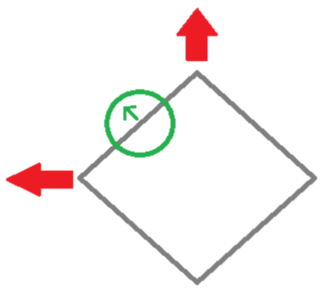


Figure 2.2: Illustration of the aperture problem. Moving the gray rectangle in the direction of any of the two red arrows both give the movement shown by the green arrow if seen through the green circled aperture.

is known as *the brightness constancy assumption*. This assumption says that objects moving in an image scene will have a constant brightness pattern along their trajectory. Let $f(\boldsymbol{x},t)$ be the brightness, or image intensity, at position $\boldsymbol{x} \in \Omega$ and time t, where Ω is a rectangular image domain. Consider an object moving along the trajectory $\boldsymbol{r}(t)$ such that

$$\frac{d\boldsymbol{r}}{dt} = \boldsymbol{w}$$

along the trajectory of this motion, where $\boldsymbol{w} = (u,v)$ is called the optical flow. By assuming a constant brightness along this trajectory, we are essentially imposing the constraint

$$\frac{d}{dt} f(\boldsymbol{r},t) = 0, \tag{2.1}$$

which, by using the chain rule of differentiation, gives

$$\nabla f^T \boldsymbol{w} + f_t = 0. \tag{2.2}$$

We call this *the optical flow equation*. Since the flow consists of two components, this equation is not sufficient to determine the flow, but only the component of the flow in the direction of the gradient, or what is known as the normal flow. This is called *the aperture problem* and it is illustrated in figure 2.2. One way of solving this problem is to introduce an additional constraint. This constraint is called *the spatial coherence assumption*, and it says that points within objects can not move independently. This assumption is explained in detail in chapter 3.

For now we denote

$$\Upsilon_{BCA}(\boldsymbol{w}) = \nabla f^T \boldsymbol{w} + f_t,$$

and let $M(\boldsymbol{w}) \geq 0$ be a term so that the solution to

$$M(\boldsymbol{w}) = 0$$

is also the solution to

$$\Upsilon_{BCA}(\boldsymbol{w}) = 0.$$

Also, we let $V(\nabla u, \nabla v) \geq 0$ be a function that penalizes the constraint from the spatial coherence assumption, so that the solution to

$$V(\nabla u, \nabla v) = 0,$$

gives a flow field satisfying this assumption. One way to look at $M(\boldsymbol{w})$ and $V(\nabla u, \nabla v)$ is to consider them as functions measuring the error in the BCA and the smoothness constrain. As Horn and Schunck [13] argued, one can not expect these to be identically zero simultaneously. We define the total error as the global energy function

$$E_{OF}(u,v) = \int_{\Omega} M(u,v) + \frac{1}{\xi^2} V(\nabla u, \nabla v)\, dx^1\, dx^2, \tag{2.3}$$

using a parameter $\xi > 0$ to determine the strength of the regularization. To simultaneously minimize $M(u,v)$ and $V(\nabla u, \nabla v)$ one wants to find the minimum of the energy functional $E_{OF}(u,v)$, solving the optimization problem

$$\underset{u,v}{\text{minimize}} \quad E_{OF}(u,v) \quad . \tag{2.4}$$

This can be done by using results from variational calculus, and the details are shown in chapter 3. Now we turn our focus to the segmentation of a flow field that solves (2.4).

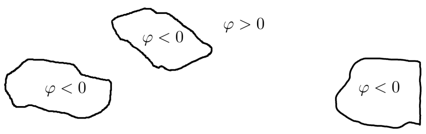
2.2 Segmentation

By segmentation we mean the task of partitioning an image into regions with some common characteristic. Given an image sequence, the objective is to divide the image into regions with movement and regions without movement. The segmented regions can be described by either a set of closed curves, or the pixels enclosed by these curves, where the former representation is used in the proposed framework. This thesis will present an energy based approach to segmentation, where the solution to the partitioning problem is assumed to be the minimum of some energy functional. That is, we try to find closed curves $C(t) : [0,1] \to \mathbb{R}^2$ that solve the optimization problem

$$\underset{C}{\text{minimize}} \quad E_{AC}(C) \tag{2.5}$$

where E_{AC} is some appropriate energy functional. The best known of these energy based segmentation approaches is called *snakes*, and was developed by Kass et al. [15]. It is an *active contours* model, which means that the curves, called snakes due to their slithering behaviour, lock onto nearby

(a) Some image scene



(b) Segmented moving objects.

Figure 2.3: The left image shows one image in the Hamburg taxi sequence. The right image shows a segmentation of the image into 4 regions, where 3 regions describe moving objects.

edges. Chapter 4 gives an outline of this active contours model. For our energy based framework, we will use an energy functional of the form

$$E_{AC}(C) = 2\int_0^1 |C'(t)| g(C(t))\,dt, \tag{2.6}$$

as proposed by Caselles et al. [7], where g is some flow edge detector function that is monotonically decreasing and such that

$$g(\boldsymbol{x}) \to 0 \text{ as } \nabla u(\boldsymbol{x}) \to \infty \text{ or } \nabla v(\boldsymbol{x}) \to \infty. \tag{2.7}$$

The role of the flow edge detector is to draw the curve C towards the flow edges, leading to the behaviour described for the active contours model.

Let now $\mathcal{I}$ and $\mathcal{O}$ denote the interior and exterior of the curve C respectively. To describe the curve C, we will use a level set function $\varphi : \mathbb{R}^2 \to \mathbb{R}$ so that

$$\varphi(\boldsymbol{x},\tau) < 0 \qquad \text{for } \boldsymbol{x} \in \mathcal{I} \tag{2.8}$$

$$\varphi(\boldsymbol{x},\tau) > 0 \qquad \text{for } \boldsymbol{x} \in \mathcal{O} \tag{2.9}$$

$$\varphi(\boldsymbol{x},\tau) = 0 \qquad \text{for } \boldsymbol{x} \text{ on } C. \tag{2.10}$$

This level set encoding is illustrated in figure 2.3 for hand drawn segmentation boundaries. Our aim is to evolve this level set function from some starting point φ_0, guided by the energy functional (2.6), so that the zero level set describes the edges of the flow $\boldsymbol{w}$ that solves problem (2.4).

In the optical flow based segmentation process we first compute a flow field $\boldsymbol{w}$ by solving (2.4), and then solve (2.5) to segment the flow field. Since the role of the flow edge detector is to detect flow edges, it is important for the segmentation process that the optical flow estimation returns a flow field with *edges* that *g* can actually *detect*. Thus, the performance of the segmentation algorithm will depend heavily on the quality of the flow field. Also, since the goal of the segmentation is to describe the boundaries of the moving objects, the estimated optical flow should give a good description of these objects. This means that the flow boundaries must be well localized.

3 Motion detection by optical flow

This chapter presents the theory behind the method used for optical flow estimation. Most of the theory presented is due to the pioneers Horn and Schunck [13], which made significant contributions to the field of variational optical flow. The first part of the chapter will give a derivation of the variational formulation of the optical flow problem, starting at the optical flow equation.

3.1 Variational optical flow

Let $f_0(\boldsymbol{x},t)$ be some image where $\boldsymbol{x}=(x^1,x^2)\in\Omega$ for some rectangular image plane Ω. The theory presented here will require a smooth image, and so a pre-processing step is required in able to guarantee this. This pre-processing step consists of convolving the image with a Gaussian filter, and so we let

$$f(x,t)=(K_\sigma * f_0), \tag{3.1}$$

where K_σ denotes a spatial Gaussian filter with standard deviation σ, so that $f\in C^\infty$. The goal of the variational approach to optical flow is to find the flow field $\boldsymbol{w}=(u,v)$ that best approximates the solution to the following set of equations

$$M(\boldsymbol{w})=0 \tag{3.2}$$

$$V(\boldsymbol{w})=0, \tag{3.3}$$

where $M(\boldsymbol{w})$ is called the data term and $V(\nabla u,\nabla v)$ is called the smoothness term. In the upcoming sections we will give a derivation of these two terms. To find the flow field that solves both (3.2) and (3.3) we want to minimize the energy functional

$$E(\boldsymbol{w})=\int_\Omega M(\boldsymbol{w})+\frac{1}{\xi^2}V(\nabla u,\nabla v)\,dx, \tag{3.4}$$

where ξ is some regularization parameter. That is, we are looking for solutions to the optimization problem

$$\underset{\boldsymbol{w}}{\text{minimize}} \quad E(\boldsymbol{w}) \quad .$$

This can be achieved by rewriting the optimization problem above as a PDE, and then solving this PDE to find the flow field. Let F be a function with continuous first partial derivatives. From calculus of variations we have that if $\boldsymbol{w}$ minimizes a functional

$$J(\boldsymbol{w})=\iint\limits_\Omega F(x^1,x^2,\boldsymbol{w},\boldsymbol{w}_{x^1},\boldsymbol{w}_{x^2})\,dx^1\,dx^2,$$

we have

$$\begin{aligned} F_w - \frac{d}{dx^1} F_{w_{x^1}} - \frac{d}{dx^2} F_{w_{x^2}} &= 0 \quad \text{in } \Omega, \\ F_{w_{x^1}} &= 0 \quad \text{on } \Gamma_E \text{ and } \Gamma_W, \\ F_{w_{x^2}} &= 0 \quad \text{on } \Gamma_N \text{ and } \Gamma_S, \end{aligned}$$

where Γ_E, Γ_W, Γ_N and Γ_S are the east, west, north and south boundaries of our domain respectively. This is called the Euler-Lagrange equation of variational calculus (see appendix A). From this result it is easy to see that the following must hold for (3.4):

$$\begin{aligned} \partial_w M - \frac{1}{\xi^2}\left(\frac{d}{dx^1}\partial_{w_{x^1}} V + \frac{d}{dx^2}\partial_{w_{x^2}} V\right) &= 0 \quad \text{in } \Omega, \\ \partial_{w_{x^1}} V &= 0 \quad \text{on } \Gamma_E \text{ and } \Gamma_W, \\ \partial_{w_{x^2}} V &= 0 \quad \text{on } \Gamma_N \text{ and } \Gamma_S. \end{aligned} \tag{3.5}$$

Given a data term and a smoothness term, this PDE can be solved to find the optical flow $\boldsymbol{w}$.

3.2 The data term

The derivation of the data term (3.2) is based on two assumptions, namely *the brightness constancy assumption* (BCA) and *the gradient constancy assumption* (GCA), where the former was used by Horn and Schunck [13], and the latter was introduced by Brox et al. [5]. These assumptions will lead to constraints in the form of equations

$$\Upsilon_{BCA}(\boldsymbol{w}) = 0, \tag{3.6}$$

$$\Upsilon_{GCA}(\boldsymbol{w}) = 0, \tag{3.7}$$

that can be combined to form one data term $M(\boldsymbol{w})$ by some method of penalization. The following subsections presents these assumptions.

3.2.1 The brightness constancy assumption

The most fundamental assumption for the construction of the data term (3.2) is the *the brightness constancy assumption* introduced in section 2.1. Remember that

$$\Upsilon_{BCA}(\boldsymbol{w}) = \nabla f^T \boldsymbol{w} + f_t,$$

and that this assumption resulted in the constraint

$$\Upsilon_{BCA}(\boldsymbol{w}) = 0.$$

As mentioned in section 2.1, the system is ill-posed, and trying to solve (3.6) results is an under-determined system (the flow has two components). However, it does allow us to compute the normal component of the flow in the direction of the gradient ∇f, *the normal flow*. To obtain a fully determined system we need to impose some smoothness constraint. This constraint comes from *the spatial coherence assumption*, which is described in detail in section 3.3. These two constraints result in a system with a unique solution, but to make the model more robust against additive illumination changes in the image scene, Brox et al. [5] suggested to include an additional constraint on the gradients of the image along the trajectory. This constraint comes from *the gradient constancy assumption*.

3.2.2 The gradient constancy assumption

In the model so far we have assumed that the illumination is the same for the whole scene, but this assumption is very likely to be violated. Thus, to make the model more robust against additive illumination changes in the image scene Brox et al. [5] proposed to include a constraint regarding the gradients of the brightness. The assumption is called *the gradient constancy assumption* (GCA), and it says that gradients remain constant under their displacement, that is

$$\frac{d}{dt}\nabla f(\boldsymbol{r}(t), t) = 0, \tag{3.8}$$

which gives

$$\nabla f_{x^1}^T \boldsymbol{w} + f_{x^1 t} = 0 \qquad \text{and} \qquad \nabla f_{x^2}^T \boldsymbol{w} + f_{x^1 t} = 0. \tag{3.9}$$

We will combine these two into one term $\Upsilon_{GCA}(\boldsymbol{w})$ by defining

$$\Upsilon_{x^1}(\boldsymbol{w}) = \nabla f_{x^1}^T \boldsymbol{w} + f_{x^1 t} \qquad \text{and} \qquad \Upsilon_{x^2}(\boldsymbol{w}) = \nabla f_{x^2}^T \boldsymbol{w} + f_{x^2 t}, \tag{3.10}$$

and letting

$$\Upsilon_{GCA}(\boldsymbol{w})^2 = \Upsilon_{x^1}(\boldsymbol{w})^2 + \Upsilon_{x^2}(\boldsymbol{w})^2. \tag{3.11}$$

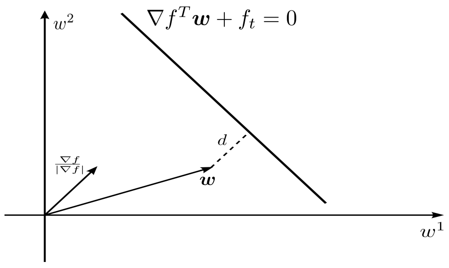
The solution to

$$\Upsilon_{GCA}(\boldsymbol{w}) = 0,$$

is given by the flow field $\boldsymbol{w}$ that satisfies (3.9).

3.2.3 Robust penalization of the data term

We now want to construct a data term which aims to penalize high values of the constraint terms Υ_{BCA} and Υ_{GCA}, but first we make some enhancements to make the model more robust.



$\nabla f^T \boldsymbol{w} + f_t = 0$
w^2
w^1
$\frac{\nabla f}{|\nabla f|}$
d
$\boldsymbol{w}$

Figure 3.1: Geometric interpretation of the line of reasoning for normalization.

Zimmer et al. [35] argued that what one actually wants to minimize is the distance between the flow vector $\boldsymbol{w}$ and the nearest solution to equation (3.6) coming from the BCA. This distance is denoted as d, and it is the shortest distance from $\boldsymbol{w}$ and the line defined by $\nabla f^T \boldsymbol{w} + f_t = 0$. This is shown in figure 3.1. The distance d is given by

$$d = \frac{\nabla f^T \boldsymbol{w} + f_t}{|\nabla f|},$$

if $|\nabla f| \neq 0$, which gives

$$\Upsilon_{BCA}(\boldsymbol{w}) = |\nabla f| d. \tag{3.12}$$

In [35] the authors reported that normalizing the data term can be beneficial, and suggested that one should ideally use d^2 to penalize the constraint coming from the BCA, which is weighted by the square of the image gradient in the expression for Υ^2_{BCA}, as seen in (3.12). We define the normalized constraint as

$$\bar{\Upsilon}_{BCA} = \theta_{BCA}(\nabla f^T \boldsymbol{w} + f_t) \approx d, \tag{3.13}$$

where the normalisation factor θ_{BCA} is defined as

$$\theta_{BCA} = \frac{1}{\sqrt{|\nabla f|^2 + \zeta^2}}.$$

The regularization parameter $\zeta > 0$ avoids division by zero and simultaneously reduces the effect of small gradients. This normalization prevents an undesirable overweighting of the BCA-constraint in areas where the image gradient is large [35].

The geometric reasoning above can also be applied to the GCA, and thus we define the normalized GCA-constraints as

$$\bar{\Upsilon}_{x^1}(\boldsymbol{w}) = \theta_{x^1}(\nabla f_{x^1}^T \boldsymbol{w} + f_{x^1 t}) \quad \text{and} \quad \bar{\Upsilon}_{x^2}(\boldsymbol{w}) = \theta_{x^2}(\nabla f_{x^2}^T \boldsymbol{w} + f_{x^2 t}), \tag{3.14}$$

where

$$\theta_{x^1} = \frac{1}{\sqrt{|\nabla f_{x^1}|^2+\zeta^2}} \qquad \text{and} \qquad \theta_{x^2} = \frac{1}{\sqrt{|\nabla f_{x^2}|^2+\zeta^2}}.$$

Then we define

$$\bar{\Upsilon}_{GCA}(\boldsymbol{w})^2 = \bar{\Upsilon}_{x^1}(\boldsymbol{w})^2 + \bar{\Upsilon}_{x^2}(\boldsymbol{w})^2, \tag{3.15}$$

and write the normalized version of (3.7) as

$$\bar{\Upsilon}_{GCA}(\boldsymbol{w}) = 0. \tag{3.16}$$

We now want to create a data term $M(\boldsymbol{w})$ that penalizes high values of $\bar{\Upsilon}_{BCA}$ and $\bar{\Upsilon}_{GCA}$, starting with a joint penalization of the two terms. Denote

$$\bar{\Upsilon}(\boldsymbol{w})^2 = \bar{\Upsilon}_{BCA}(\boldsymbol{w})^2 + \gamma_{OF}\bar{\Upsilon}_{GCA}(\boldsymbol{w})^2, \tag{3.17}$$

where $\gamma_{OF} > 0$ controls the contribution of the GCA part of the data term. Let $\Psi(h^2)$ be a function that penalizes high values of h^2. By setting the data term to be

$$M(\boldsymbol{w}) = \Psi\left(\bar{\Upsilon}(\boldsymbol{w})^2\right), \tag{3.18}$$

we jointly penalize the BCA and the GCA. The contribution to the Euler-Lagrange system (3.5) is

$$\partial_{\boldsymbol{w}} M = 2\Psi'\left(\bar{\Upsilon}^2\right)\left(\bar{\Upsilon}_{BCA}\partial_{\boldsymbol{w}}\bar{\Upsilon}_{BCA} + \gamma_{OF}\bar{\Upsilon}_{GCA}\partial_{\boldsymbol{w}}\bar{\Upsilon}_{GCA}\right),$$

where

$$\partial_{\boldsymbol{w}}\bar{\Upsilon}_{BCA} = \frac{\nabla f}{\sqrt{|\nabla f|+\zeta^2}}, \tag{3.19}$$

$$\partial_{\boldsymbol{w}}\bar{\Upsilon}_{GCA} = \frac{1}{\bar{\Upsilon}_{GCA}}\left(\bar{\Upsilon}_{x^1}\frac{\nabla f_{x^1}}{\sqrt{|\nabla f_{x^1}|+\zeta^2}} + \bar{\Upsilon}_{x^2}\frac{\nabla f_{x^2}}{\sqrt{|\nabla f_{x^2}|+\zeta^2}}\right). \tag{3.20}$$

Thus the contribution to the system is seen to be

$$\partial_{\boldsymbol{w}} M = 2\Psi'\left(\bar{\Upsilon}^2\right)\left(\bar{\Upsilon}_{BCA}\overline{\nabla f} + \gamma_{OF}\left(\bar{\Upsilon}_{x^1}\overline{\nabla f_{x^1}} + \bar{\Upsilon}_{x^2}\overline{\nabla f_{x^2}}\right)\right), \tag{3.21}$$

where we simplify the notation by defining

$$\overline{\nabla f} = \frac{\nabla f}{\sqrt{|\nabla f|+\zeta^2}}, \qquad \overline{\nabla f_{x^1}} = \frac{\nabla f_{x^1}}{\sqrt{|\nabla f_{x^1}|+\zeta^2}}, \qquad \overline{\nabla f_{x^2}} = \frac{\nabla f_{x^2}}{\sqrt{|\nabla f_{x^2}|+\zeta^2}}. \tag{3.22}$$

When the brightness constancy assumption was first introduced, Horn and Schunck used a quadratic penalization of $\bar{\Upsilon}_{BCA}$,

$$\Psi(h^2) = h^2,$$

which is equivalent to a least-squares minimization. This penalization reduces (3.21) to

$$\partial_w M = 2\bar{\Upsilon}_{BCA}\overline{\nabla f} + 2\gamma_{OF}\left(\bar{\Upsilon}_{x^1}\overline{\nabla f_{x^1}} + \bar{\Upsilon}_{x^2}\overline{\nabla f_{x^2}}\right),$$

which is the same as

$$\partial_w M = 2\frac{\nabla f^T w + f_t}{|\nabla f|^2 + \zeta^2} + 2\gamma_{OF}\left(\frac{\nabla f_{x^1}^T w + f_t}{|\nabla f_{x^1}|^2 + \zeta^2} + \frac{\nabla f_{x^2}^T w + f_t}{|\nabla f_{x^2}|^2 + \zeta^2}\right). \tag{3.23}$$

Other methods of penalization have been proposed. Black and Anandan [4] proposed several subquadratic penalizer functions, arguing that a subquadratic penalizer would improve the robustness in the presence of outliers. Still, due to its simplicity and satisfactory results, we will only use quadratic penalizations of the data term in this thesis.

By using two assumptions, the BCA and the GCA, we have derived two constraints for our optical flow model, and combined them in one constraint term called the data term. In the earliest framework for variational optical flow, proposed by Horn and Schunck, the data term only included the BCA. As we noted in section 2.1 the BCA does not admit a unique solution. To obtain a unique solution Horn and Schunck suggested imposing a smoothness constraint on the flow.

3.3 The smoothness term

As a reminder we state the energy functional for optical flow given in section 3.1,

$$E(w) = \int_\Omega M(w) + \frac{1}{\xi^2}V(\nabla u, \nabla v)\,dx,$$

where $M(w)$ denotes the data term derived in the previous section, and $V(\nabla u, \nabla v)$ is a smoothness term to be derived in this section. As noted previously, the optical flow equation does not admit a unique solution using only the BCA. To get uniqueness of solutions a common approach is to incorporate a smoothness constraint in the model, an idea introduced by Horn and Schunck [13]. This smoothness constraint, also called *the spatial coherence assumption* [4], says that points can not move independently in the brightness pattern. There has to be some smoothness in the flow vector for points belonging to the same object. In other words, points on the same object move with approximately the same velocity. A natural way of obtaining a smoother solution would be to minimize some term depending on the sizes of the gradients ∇u and ∇v. The penalization of these terms will be either quadratic or subquadratic, and it will be instructive to write the

Euler-Lagrange system in the form

$$\begin{aligned} \partial_u M(\boldsymbol{w}) - \frac{1}{\xi^2}\operatorname{div}(\Theta_u \nabla u) &= 0, \\ \partial_v M(\boldsymbol{w}) - \frac{1}{\xi^2}\operatorname{div}(\Theta_v \nabla v) &= 0, \end{aligned} \tag{3.24}$$

where the data term contribution is given by equation (3.21). The matrices $\Theta_u = \Theta_u(x^1, x^2, \nabla u, \nabla v)$ and $\Theta_v = \Theta_v(x^1, x^2, \nabla u, \nabla v)$ will be called the diffusion matrices, as they control the direction and strength of the diffusion process; their eigenvectors and eigenvalues determine the direction and strength respectively. We will start by looking at the simplest smoothness term, namely the isotropic smoothing of Horn and Schunck.

3.3.1 Isotropic smoothing

The smoothness term used by Horn and Schunck is

$$V(\nabla u, \nabla u) = |\nabla u|^2 + |\nabla v|^2.$$

This is a homogeneous regularizer which means that it applies an equal amount of diffusion in all directions. In the framework of (3.24), this is equivalent to the diffusion matrices Θ_u and Θ_v being the identity matrix. Using this function as a flow regularizer gives

$$\begin{aligned} \partial_{\boldsymbol{w}_{x^1}} V &= 2\boldsymbol{w}_{x^1}, \\ \partial_{\boldsymbol{w}_{x^2}} V &= 2\boldsymbol{w}_{x^2}. \end{aligned}$$

In their original paper Horn and Schunck used a quadratic penalization of the data term without normalization and with no gradient constancy constraint ($\gamma_{OF} = 0$). This results in the system

$$\begin{aligned} (f_{x^1}u + f_{x^2}v + f_t)f_{x^1} - \frac{1}{\xi^2}\left(\frac{d}{dx^1}u_{x^1} + \frac{d}{dx^2}u_{x^2}\right) &= 0 \quad \text{in } \Omega, \\ (f_{x^1}u + f_{x^2}v + f_t)f_{x^2} - \frac{1}{\xi^2}\left(\frac{d}{dx^1}v_{x^1} + \frac{d}{dx^2}v_{x^2}\right) &= 0 \quad \text{in } \Omega, \\ \boldsymbol{w}_{x^1} &= 0 \quad \text{on } \Gamma_E \text{ and } \Gamma_W, \\ \boldsymbol{w}_{x^2} &= 0 \quad \text{on } \Gamma_N \text{ and } \Gamma_S, \end{aligned} \tag{3.25}$$

which can be seen as a system of coupled linear elliptic equations with Neumann boundary conditions:

$$\begin{aligned} -\frac{1}{\xi^2}\Delta u + f_{x^1}^2 u &= -(F(v) + f_t f_{x^1}), \\ -\frac{1}{\xi^2}\Delta v + f_{x^2}^2 v &= -(F(u) + f_t f_{x^2}), \end{aligned}$$

where $F(q) = f_{x^1} f_{x^2} q$.

This smoothness term smooths the flow in all directions, which is not always desirable as this may blur out important flow edges. We want to keep the flow field discontinuous at the optical flow boundaries. These flow boundaries are not known a priori, but one might expect them to coincide with the image edges. This is the reasoning behind the anisotropic smoothness term of Nagel and Enkelmann [22].

3.3.2 Anisotropic image driven smoothing

As one of the assumptions to the optical flow model is that different objects have different brightness patterns, one would expect that flow boundaries are contained in image edges. Thus, an amendment to the issue of blurry flow edges is to construct a smoothness term which takes the gradients of the image into account, and smooths the flow field *along* image edges instead of *across* them. Such methods are called image driven regularization methods. To that end, consider the 2×2 *structure matrix*

$$S_\rho = K_\rho * \left[\nabla f \nabla f^T\right], \tag{3.26}$$

where K_ρ is a spatial Gaussian with standard deviation ρ, and denote its eigenvectors as $\boldsymbol{s}_1$ and $\boldsymbol{s}_2$. These eigenvectors point *across* and *along* image structures respectively [35]. For $\rho = 0$ these vectors correspond to the unit vectors

$$\boldsymbol{s}_1^0 = \frac{1}{|\nabla f|}\begin{bmatrix} f_{x^1} \\ f_{x^2} \end{bmatrix}, \qquad \boldsymbol{s}_2^0 = \frac{1}{|\nabla f|}\begin{bmatrix} -f_{x^2} \\ f_{x^1} \end{bmatrix}. \tag{3.27}$$

The anisotropic regularizer of Nagel and Enkelmann [22] performs smoothing in the direction given by $\boldsymbol{s}_2$, that is, along image structures, and prevents smoothing across image structures. Given a vector $\boldsymbol{z} \in \mathbb{R}^2$ with an orthogonal vector $\boldsymbol{z}^\perp$ of same length, let $P(\boldsymbol{z})$ be a 2×2 regularized projection matrix defined as

$$P(\boldsymbol{z}) = \frac{1}{|\boldsymbol{z}|^2 + 2\kappa^2}(\boldsymbol{z}^\perp(\boldsymbol{z}^\perp)^T + \kappa^2 I),$$

where $\kappa > 0$ is a regularization parameter. For $\kappa = 0$ (and $|\boldsymbol{z}|$) the matrix multiplication $P(\boldsymbol{z})\boldsymbol{q}$ projects the vector $\boldsymbol{q}$ in the direction given by $\boldsymbol{z}^\perp$. To prevent numerical issues in the cases where $|\boldsymbol{z}| \approx 0$, we will use a small regularization. The smoothness term of Nagel and Enkelmann is given as

$$\begin{aligned} V(\nabla u, \nabla v) &= \nabla u^T P(\nabla f)\nabla u + \nabla v^T P(\nabla f)\nabla v \\ &= \frac{(f_{x^1}^2 + \kappa^2)(u_{x^1})^2 - 2 f_{x^1} f_{x^2} u_{x^1} u_{x^2} + (f_{x^2}^2 + \kappa^2) u_{x^2}}{|\nabla f|^2 + 2\kappa^2} \\ &\qquad + \frac{(f_{x^1}^2 + \kappa^2)(v_{x^1})^2 - 2 f_{x^1} f_{x^2} v_{x^1} v_{x^2} + (f_{x^2}^2 + \kappa^2) v_{x^2}}{|\nabla f|^2 + 2\kappa^2}. \end{aligned}$$

Now we define the following directional derivatives

$$u_{s_1} = s_1^T \nabla u = \frac{f_{x^1} u_{x^1} + f_{x^2} u_{x^2}}{|\nabla f|}, \qquad v_{s_1} = s_1^T \nabla v = \frac{f_{x^1} v_{x^1} + f_{x^2} v_{x^2}}{|\nabla f|}, \tag{3.28}$$

$$u_{s_2} = s_2^T \nabla u = \frac{-f_{x^2} u_{x^1} + f_{x^1} u_{x^2}}{|\nabla f|}, \qquad v_{s_2} = s_2^T \nabla v = \frac{-f_{x^2} v_{x^1} + f_{x^1} v_{x^2}}{|\nabla f|}. \tag{3.29}$$

Using this notation we can write the smoothness term of Nagel and Enkelmann as

$$\begin{aligned} V(\nabla u, \nabla v) &= \frac{|\nabla f| u_{s_2}^2 + \kappa^2 \left(u_{s_1}^2 + u_{s_2}^2 \right)}{|\nabla f|^2 + 2\kappa^2} + \frac{|\nabla f| v_{s_2}^2 + \kappa^2 \left(v_{s_1}^2 + v_{s_2}^2 \right)}{|\nabla f|^2 + 2\kappa^2} \\ &= \frac{\kappa^2}{|\nabla f|^2 + 2\kappa^2} \left(u_{s_1}^2 + v_{s_1}^2 \right) + \frac{|\nabla f|^2 + \kappa^2}{|\nabla f|^2 + 2\kappa^2} \left(u_{s_2}^2 + v_{s_2}^2 \right). \end{aligned}$$

If κ is small, setting $\Theta_u = \Theta_v = P$ in (3.24) steers the diffusion so that flow vectors are smoothed along image edges and not across them. This leads to the following Euler-Lagrange system:

$$\begin{aligned} \frac{\partial M}{\partial u} - \frac{1}{\xi^2} \operatorname{div}(P \nabla u) &= 0, \\ \frac{\partial M}{\partial v} - \frac{1}{\xi^2} \operatorname{div}(P \nabla v) &= 0, \end{aligned}$$

or equivalently

$$\frac{\partial M}{\partial u} - \frac{2}{\xi^2} \left(\frac{d}{dx^1} \frac{(f_{x^2}^2 + \kappa^2) u_{x^1} - f_{x^1} f_{x^2} u_{x^2}}{|\nabla f|^2 + 2\kappa^2} + \frac{d}{dx^2} \frac{-f_{x^1} f_{x^2} u_{x^1} + (f_{x^1}^2 + \kappa^2) u_{x^2}}{|\nabla f|^2 + 2\kappa^2} \right) = 0,$$

$$\frac{\partial M}{\partial v} - \frac{2}{\xi^2} \left(\frac{d}{dx^1} \frac{(f_{x^2}^2 + \kappa^2) v_{x^1} - f_{x^1} f_{x^2} v_{x^2}}{|\nabla f|^2 + 2\kappa^2} + \frac{d}{dx^2} \frac{-f_{x^1} f_{x^2} v_{x^1} + (f_{x1}^2 + \kappa^2) v_{x^2}}{|\nabla f|^2 + 2\kappa^2} \right) = 0.$$

As this diffusion matrix uses structural information of the image to steer the diffusion, the resulting flow field may induce flow discontinuities at image edges that does not coincide with flow edges. An amendment to this oversegmentation is to make the diffusion matrix depend on the flow edges, and not the image edges; a flow driven approach.

3.3.3 Isotropic flow driven smoothing

As mentioned in the previous subsection, a drawback of the image driven approach to regularization is that there is often a great deal of oversegmentation, since image boundaries are not necessarily flow boundaries. The solution is to decrease the smoothing at flow boundaries, which leads to a so called flow driven approach. An obvious problem here is that the flow boundaries are not known a priori. The flow boundaries are located in regions where the values of the flow derivatives are high, and so the aim is to reduce smoothing in these regions. This can be done by the use of a subquadratic penalization of the flow gradients. Shulman and Herve [29] suggested using a subquadratic penalizer instead of a quadratic one. They argued that a quadratic penalizer assumes a Gaussian distribution of the flow gradients which would penalize large gradients, assumed to correspond to flow boundaries, too much. The flow driven smoothness term can be written as

$$V(\nabla u, \nabla v) = \psi_V\left(|\nabla u|^2 + |\nabla v|^2\right),$$

where $\psi_V(h^2)$ is some subquadratic penalizing function performing a nonlinear isotropic diffusion, reducing the diffusion at flow boundaries by means of the decreasing diffusivity ψ'. The contribution to (3.5) is

$$\operatorname{div}\left(\left[\partial_{u_{x1}} V, \partial_{u_{x2}} V\right]\right).$$

Computing the individual components, we get

$$\begin{aligned}
\partial_{u_{x1}} V &= 2\psi_V'\left(|\nabla u|^2 + |\nabla v|^2\right) u_{x^1} \\
\partial_{u_{x2}} V &= 2\psi_V'\left(|\nabla u|^2 + |\nabla v|^2\right) u_{x^2},
\end{aligned}$$

and similarly for $(\partial_{v_{x1}} V, \partial_{v_{x2}} V)$. Thus the diffusion matrices of (3.24) is given as

$$\Theta_u = \Theta_v = 2\psi_V'\left(|\nabla u|^2 + |\nabla v|^2\right) I,$$

where I is the identity matrix. The diffusion matrix is now seen to be a function of the flow gradients, and the strength of the diffusion will depend on the sizes of the gradients.

As a convex penalizer, Cohen [10] suggested the following total variation (TV) regularizer:

$$\psi_V(h^2) = \sqrt{h^2 + \epsilon_{OF}^2}, \tag{3.30}$$

with $\epsilon_{OF} > 0$ being a small regularization parameter. The TV regularizer essentially minimizes the L^1-norm of the gradient of the flow, which has shown to give good results for image denoising [25]. In addition to being more robust to outliers compared to the minimization of the L^2-norm, the total variation allows for sharp edges and discontinuities in the flow [10]. The diffusivity is

$$\psi_V'(h^2) = \frac{1}{2\sqrt{h^2 + \epsilon_{OF}^2}}.$$

This penalizer function results in the Euler-Lagrange system

$$\begin{aligned}
\partial_u M - \frac{1}{\xi^2}\left(\frac{\partial}{\partial x^1}\left[\frac{u_{x^1}}{\sqrt{|\nabla u|^2+|\nabla v|^2+\epsilon_{OF}^2}}\right]+\frac{\partial}{\partial x^2}\left[\frac{u_{x^2}}{\sqrt{|\nabla u|^2+|\nabla v|^2+\epsilon_{OF}^2}}\right]\right) &= 0,\\
\partial_v M - \frac{1}{\xi^2}\left(\frac{\partial}{\partial x^1}\left[\frac{v_{x^1}}{\sqrt{|\nabla u|^2+|\nabla v|^2+\epsilon_{OF}^2}}\right]+\frac{\partial}{\partial x^2}\left[\frac{v_{x^2}}{\sqrt{|\nabla u|^2+|\nabla v|^2+\epsilon_{OF}^2}}\right]\right) &= 0.
\end{aligned} \tag{3.31}$$

Unlike the previous linear systems, which could be solved directly, this system is nonlinear and must be solved by some iterative method. We propose to do this by ussing the method of lagged diffusivity.

The method of lagged diffusivity

From equation (3.31), the lagged diffusivity fixed point iteration can be defined as

$$\begin{aligned}
\partial_{u^{k+1}} M - \frac{1}{\xi^2}\left(\frac{\partial}{\partial x}\left[\frac{u^{k+1}_{x^1}}{\sqrt{|\nabla u^k|^2+|\nabla v^k|^2+\epsilon_{OF}^2}}\right]+\frac{\partial}{\partial x^2}\left[\frac{u^{k+1}_{x^2}}{\sqrt{|\nabla u^k|^2+|\nabla v^k|^2+\epsilon_{OF}^2}}\right]\right) &= 0,\\
\partial_{v^{k+1}} M - \frac{1}{\xi^2}\left(\frac{\partial}{\partial x}\left[\frac{v^{k+1}_{x^1}}{\sqrt{|\nabla u^k|^2+|\nabla v^k|^2+\epsilon_{OF}^2}}\right]+\frac{\partial}{\partial x^2}\left[\frac{v^{k+1}_{x^2}}{\sqrt{|\nabla u^k|^2+|\nabla v^k|^2+\epsilon_{OF}^2}}\right]\right) &= 0,
\end{aligned}$$

where the flow components u^{k+1} and v^{k+1} are obtained by using the diffusivity from the previous iteration. If we let the data term be penalized quadratically, we get the system

$$\begin{aligned}
&\bar{\Upsilon}_{BCA}\left(w^{k+1}\right)\overline{f_{x^1}} + \gamma_{OF}\left(\bar{\Upsilon}_{x^1}\left(w^{k+1}\right)\overline{f_{x^1x^1}} + \bar{\Upsilon}_{x^2}\left(w^{k+1}\right)\overline{f_{x^1x^2}}\right)\\
&\quad - \frac{1}{2\xi^2}\left(\frac{\partial}{\partial x}\left[\frac{u^{k+1}_{x^1}}{\sqrt{|\nabla u^k|^2+|\nabla v^k|^2+\epsilon_{OF}^2}}\right]+\frac{\partial}{\partial x^2}\left[\frac{u^{k+1}_{x^2}}{\sqrt{|\nabla u^k|^2+|\nabla v^k|^2+\epsilon_{OF}^2}}\right]\right) = 0,
\end{aligned}$$

$$\begin{aligned}
&\bar{\Upsilon}_{BCA}\left(w^{k+1}\right)\overline{f_{x^2}} + \gamma_{OF}\left(\bar{\Upsilon}_{x^1}\left(w^{k+1}\right)\overline{f_{x^1x^2}} + \bar{\Upsilon}_{x^2}\left(w^{k+1}\right)\overline{f_{x^2x^2}}\right)\\
&\quad - \frac{1}{2\xi^2}\left(\frac{\partial}{\partial x}\left[\frac{v^{k+1}_{x^1}}{\sqrt{|\nabla u^k|^2+|\nabla v^k|^2+\epsilon_{OF}^2}}\right]+\frac{\partial}{\partial x^2}\left[\frac{v^{k+1}_{x^2}}{\sqrt{|\nabla u^k|^2+|\nabla v^k|^2+\epsilon_{OF}^2}}\right]\right) = 0,
\end{aligned}$$

where the bar notation is used to indicate the normalization of the data term as seen in equation (3.22). The convergence of the system above has been shown in [34] provided

$$\psi_V'(|\nabla u^k|^2 + |\nabla v^k|^2) = \frac{1}{2\sqrt{|\nabla u^k|^2 + |\nabla v^k|^2 + \epsilon_{OF}^2}}$$

is bounded. The rate of convergence is linear, with a convergence constant depending on the upper and lower bound of $\psi_V'(|\nabla u^k|^2 + |\nabla v^k|^2)$, and in particular the value of ϵ_{OF}; as ϵ_{OF} decreases, the rate of convergence decreases.

3.4 Optical flow summary

We have now obtained the data term and the smoothness term. The data term consists of the *brightness constancy assumption* and the *gradient constancy assumption*. These are penalized jointly using a quadratic penalization.

We also have three different methods of regularizing optical flow. The original isotropic smoothing of Horn and Schunck smooths the flow field an equal amount in all directions. The system is given as

$$\begin{aligned} \partial_u M - \frac{1}{\xi^2}\left(\frac{d}{dx^1}u_{x^1} + \frac{d}{dx^2}u_{x^2}\right) &= 0, \\ \partial_v M - \frac{1}{\xi^2}\left(\frac{d}{dx^1}v_{x^1} + \frac{d}{dx^2}v_{x^2}\right) &= 0. \end{aligned}$$

The anisotropic smoothing of Nagel and Enkelmann, also called image driven smoothing, looks at the image edges and tries to smooth the field along image edges, and not across them. The system can be written as

$$\begin{aligned} \partial_u M - \frac{2}{\xi^2}\left(\frac{d}{dx^1}\frac{(f_{x^2}^2 + \kappa^2)u_{x^1} - f_{x^1}f_{x^2}u_{x^2}}{|\nabla f|^2 + 2\kappa^2} + \frac{d}{dx^2}\frac{-f_{x^1}f_{x^2}u_{x^1} + (f_{x^1}^2 + \kappa^2)u_{x^2}}{|\nabla f|^2 + 2\kappa^2}\right) &= 0, \\ \partial_v M - \frac{2}{\xi^2}\left(\frac{d}{dx^1}\frac{(f_{x^2}^2 + \kappa^2)v_{x^1} - f_{x^1}f_{x^2}v_{x^2}}{|\nabla f|^2 + 2\kappa^2} + \frac{d}{dx^2}\frac{-f_{x^1}f_{x^2}v_{x^1} + (f_{x^1}^2 + \kappa^2)v_{x^2}}{|\nabla f|^2 + 2\kappa^2}\right) &= 0. \end{aligned}$$

The flow driven regularization aims to reduce the smoothing at flow boundaries. The method performs an isotropic smoothing, but controls the strength of the smoothing so that points close to the flow boundaries are not smoothed out as much as other points. The flow driven regularization results in the system

$$\begin{aligned} \partial_u M - \frac{1}{\xi^2}\left(\frac{\partial}{\partial x^1}\left[\frac{u_{x^1}}{\sqrt{|\nabla u|^2 + |\nabla v|^2 + \epsilon_{OF}^2}}\right] + \frac{\partial}{\partial x^2}\left[\frac{u_{x^2}}{\sqrt{|\nabla u|^2 + |\nabla v|^2 + \epsilon_{OF}^2}}\right]\right) &= 0, \\ \partial_v M - \frac{1}{\xi^2}\left(\frac{\partial}{\partial x^1}\left[\frac{v_{x^1}}{\sqrt{|\nabla u|^2 + |\nabla v|^2 + \epsilon_{OF}^2}}\right] + \frac{\partial}{\partial x^2}\left[\frac{v_{x^2}}{\sqrt{|\nabla u|^2 + |\nabla v|^2 + \epsilon_{OF}^2}}\right]\right) &= 0. \end{aligned}$$

When these regularization methods were proposed, none of them used the GCA. Thus, to compare these methods, chapter 7 presents the results without using the GCA ($\gamma_{OF} = 0$). In some image scenes, however, using the GCA can lead to more favourable results. Hence, some of the segmentation results in chapter 8 are computed from flow fields using the GCA. This will be clarified when necessary.

4 Active contours

Having found an optical flow field by using the methods of the previous chapter, we are now concerned with detecting the contours of this flow field. Traditionally, the methods presented in this chapter have been aimed at finding the contours of an image. Hence, we will start by presenting the methods in an image analysis framework. Section 4.1 begins by introducing the snakes model [15] for detecting image contours before turning to the model of geodesic active contours. Section 4.1.2 applies these methods of image contours to the problem of finding flow contours. Section 4.2 describes the level set method, which is a way of representing the contours found by the active contours model.

4.1 Energy based active contours

This section presents the classical energy based snakes approach. Let $f : \Omega \to \mathbb{R}$ be a given image. The goal of the snakes approach is to detect image boundaries of f, minimizing

$$E_1(C) = \alpha \int_0^1 |C'(t)|^2 \, dt + \beta \int_0^1 |C''(t)|^2 \, dt - \lambda \int_0^1 |\nabla f(C(t))| \, dt$$

over all parametrized closed curves $C(t) : [0,1] \to \mathbb{R}^2$, where α, β and γ are real positive constants. We will additionally assume that the closed curve C is a Jordan curve parametrized in such a way that the curve is traced clockwise. In this energy functional the first two terms serve to penalize first and second order derivatives of the curve, and thus they control the smoothness of the detected contours. The last term serves to attract the curve towards the image features with high gradients. Caselles et al. [7] derived a relation between the evolution of this energy based snakes contour and a geometric curve evolution, using $\beta = 0$. The authors of [7] argued that the final active contour is sufficiently smooth, which makes the second term of the energy functional unneeded. Assuming $\beta = 0$ the curve energy is reduced to

$$E_1(C) = \alpha \int_0^1 |C'(t)|^2 \, dt - \lambda \int_0^1 |\nabla f(C(t))| \, dt.$$

With the above energy depending only on two parameters we can without loss of generality set $\alpha = 1$ and obtain

$$E_1(C) = \int_0^1 |C'(t)|^2 \, dt - \lambda \int_0^1 |\nabla f(C(t))| \, dt. \tag{4.1}$$

In minimizing this functional we are essentially trying to globally maximize $|\nabla f|$, while simultaneously minimizing the curvature. The maxima of $|\nabla f|$ correspond to the image edges of f. For that purpose, Caselles et al. [7] introduced an edge detector function $g : [0,\infty) \to \mathbb{R}^+$ that is strictly decreasing and satisfies

$$g(r) \to 0 \text{ as } r \to \infty. \tag{4.2}$$

The ideas is to minimize $g(|\nabla f|)^2$ instead of minimizing $-|\nabla f|$. Replacing $-|\nabla f|$ with $g(|\nabla f|)^2$ in (4.1) gives

$$\begin{aligned} E_1(C) &= \int_0^1 |C'(t)|^2 \, dt + \lambda \int_0^1 g(|\nabla f(C(t))|)^2 \, dt & (4.3) \\ &= E_{\text{int}}(C) + \lambda E_{\text{ext}}(C), & (4.4) \end{aligned}$$

where

$$E_{\text{int}}(C) = \int_0^1 |C'(t)|^2 \, dt$$

is the internal energy of the curve C and

$$E_{\text{ext}}(C) = \int_0^1 g(|\nabla f(C(t))|)^2 \, dt$$

is the external energy.

4.1.1 The Geodesic model

The energy functional given in (4.1) is not intrinsic, as it depends on the parametrization of C. This is seen by parametrizing the curve C with a new variable q given by $t = \phi(q)$, for some function $\phi : [0,1] \to [0,1]$. Inserting this into (4.3) gives

$$E_1(C) = \int_0^1 |C'(q)|^2 \frac{1}{\phi'(q)} \, dq + \lambda \int_0^1 g(|\nabla f(C(q))|)^2 \phi'(q) \, dq,$$

which clearly depends on the function ϕ. Thus, as proposed in [7], we define a new functional

$$E_2(C) = \int_0^1 |C'(t)| g(|\nabla f(C(t))|)\, dt, \tag{4.5}$$

which gives

$$E_2(C) = \int_0^1 |C'(q)| g(|\nabla f(C(q))|)\, dq,$$

and is seen to be intrinsic. In [2, chapter 4,p. 177-181] the authors formulated a definition of equivalence for these two problems, and used this definition to argue that the problems $\inf_C E_1(C)$ and $\inf_C E_2(C)$ are equivalent. Comparing $E_2(C)$ with the expression for the Euclidean length of the curve C

$$L = \int_0^1 |C'(t)|\, dt,$$

we see that our new energy functional is just a weighted length,

$$L_R(C) = E_2(C) = \int_0^1 \sqrt{C'^T R C'}\, dt = \int_0^1 ||C'||_R\, dt, \tag{4.6}$$

where we have defined the norm

$$||\boldsymbol{p}||_R^2 = \boldsymbol{p}^T R \boldsymbol{p}.$$

The positive definite matrix R is given as

$$R(t) = \lambda g(|\nabla f(C(t)|) I.$$

The problem of detecting image boundaries can now be formulated as the minimization of a length in a curved space defined by the new metric:

$$\underset{C}{\text{minimize}} \quad \int_0^1 ||C'||_R\, dt. \tag{4.7}$$

This curve of minimal length is called a geodesic curve.

4.1.2 Edge detector for optical flow boundaries

Let now the vector field $\boldsymbol{w}(x) = (u(x), v(x))$ denote the optical flow of some image sequence. Since our aim is to detect flow boundaries rather than image boundaries, we want to construct an edge detector that detects flow edges. The optical flow boundaries are given by the points in the vector field where $|\nabla u|$ and $|\nabla v|$ are large, thus instead of the edge detector given in the previous section we define an edge detector $g(r,q) : [0,\infty) \to \mathbb{R}^+$ that is strictly decreasing in both arguments and is such that

$$g(r,q) \to 0 \text{ as } r \to \infty \text{ or } q \to \infty, \tag{4.8}$$

analogous to assumption (4.2) for the image edge detector. Setting

$$R(t) = \lambda g\left(|\nabla u\left(C\left(t\right)\right)|, |\nabla v\left(C\left(t\right)\right)|\right) I, \tag{4.9}$$

and solving the minimization problem of equation (4.7) should draw the curve C towards flow boundaries.

4.1.3 Curve evolution

The steepest descent method will be used to solve the optimization problem (4.7). We start by finding an expression for the variation of

$$J(C) = \int_0^1 g\left(|\nabla u\left(C\left(t\right)\right)|, |\nabla v\left(C\left(t\right)\right)|\right) |C'\left(t\right)| \, dt,$$

assuming C is an immersed closed curve. To simplify the notation, the edge detector is written as a function of $\boldsymbol{x}$, so that $g(\boldsymbol{x}) = g\left(|\nabla u\left(\boldsymbol{x}\right)|, |\nabla v\left(\boldsymbol{x}\right)|\right)$, and

$$J(C) = \int_0^1 g(C(t))|C_t| \, dt, \tag{4.10}$$

where $C_t = C'(t)$. The first variation of $J(C(t))$ in direction $\eta(t)$ is given by

$$\delta J(C;\eta) = \left.\frac{d}{d\tau}\right|_{\tau=0} J(C(t) + \tau\eta(t)),$$

and so

$$\delta J(C;\eta) = \int_0^1 \left.\frac{d}{d\tau}\right|_{\tau=0} g(C + \tau\eta)|C_t + \tau\eta_t| \, dt,$$

where we omit specifying the dependence on the space parameter t for C and η. By using the product rule for differentiation we get

$$\begin{aligned}\delta J(C;\eta) = \int_0^1 |C_t + \tau\eta_t|_{\tau=0} \frac{d}{d\tau}\bigg|_{\tau=0} g(C+\tau\eta)\,dt \\ + \int_0^1 g(C+\tau\eta)_{\tau=0} \frac{d}{d\tau}\bigg|_{\tau=0} |C_t + \tau\eta_t|\,dt \\ = \int_0^1 (\nabla g(C)\cdot\eta)|C_t| + g(C)(\boldsymbol{T}\cdot\eta_t)\,dt, \quad (4.11)\end{aligned}$$

where $\boldsymbol{T}$ is the unit tangent vector to the curve C, defined as

$$\boldsymbol{T}(t) = \frac{C_t(t)}{|C_t(t)|}.$$

Using integration by parts we obtain

$$\begin{aligned}\int_0^1 g(C)\boldsymbol{T}\cdot\eta_t\,dt &= [g(C)\boldsymbol{T}\cdot\eta]_0^1 - \int_0^1 (g(C)\boldsymbol{T})_t\cdot\eta\,dt \\ &= -\int_0^1 (g(C)\boldsymbol{T})_t\cdot\eta\,dt \\ &= -\int_0^1 [\nabla g(C)\cdot C_t\boldsymbol{T} + g(C)\boldsymbol{T}_t]\cdot\eta\,dt \\ &= -\int_0^1 [\nabla g(C)\cdot C_t][\boldsymbol{T}\cdot\eta] + [g(C)\boldsymbol{T}_t\cdot\eta]\,dt,\end{aligned}$$

where we have used the assumption that $C(t)$ and $\eta(t)$ are closed curves so that $C(0) = C(1)$ and $\eta(0) = \eta(1)$. Inserting this in (4.11) leads to

$$\delta J(C;\eta) = \int_0^1 [\nabla g(C)\cdot\eta]|C_t| - [\nabla g(C)\cdot C_t][\boldsymbol{T}\cdot\eta] - [g(C)\boldsymbol{T}_t\cdot\eta]\,dt.$$

Now, let s denote the arc-length of $C(t)$, and note that $C_t = \boldsymbol{T}(t)|C_t|$, $\boldsymbol{T}_t = \boldsymbol{T}_s|C_t|$ and $ds = |C_t|dt$. By parametrizing the curve by arc-length the variation can be written as

$$\delta J(C;\eta) = \int_0^{L(C)} \nabla g(C(s)) \cdot \eta(s) \\ \qquad - [\nabla g(C(s)) \cdot \boldsymbol{T}(s)][\boldsymbol{T}(s) \cdot \eta(s)] - g(C(s))\boldsymbol{T}_s(s) \cdot \eta(s)\, ds.$$

Let the signed curvature be defined as

$$\kappa = \boldsymbol{T}_s \cdot \boldsymbol{N}, \tag{4.12}$$

where $\boldsymbol{N}$ is the unit inward normal assuming the curve $C(t)$ is traced clockwise for increasing values of t. Then, observe that

$$\boldsymbol{T}_s = \kappa \boldsymbol{N} \qquad \text{and} \qquad \nabla g(C) - (\nabla g(C) \cdot \boldsymbol{T})\boldsymbol{T} = (\nabla g(C) \cdot \boldsymbol{N})\boldsymbol{N}.$$

We have arrived at the following expression for the variation:

$$\delta J(C;\eta) = \int_0^{L(C)} [(\nabla g(C) \cdot \boldsymbol{N})\boldsymbol{N} - g(C)\kappa \boldsymbol{N}] \cdot \eta\, ds. \tag{4.13}$$

One might be tempted to conclude that we have found a derivative, and consequently a gradient, for the energy functional $J(C)$. But care must be taken, as there is no obvious way to define a tangent space on this space of curves (see, however, appendix C). We can at least define the formal gradient as

$$\nabla J(C) = (\nabla g(C) \cdot \boldsymbol{N})\boldsymbol{N} - g(C)\kappa \boldsymbol{N}. \tag{4.14}$$

Using the steepest descent method to evolve the curve we set

$$\begin{aligned} C_\tau &= g(\boldsymbol{w})\kappa \boldsymbol{N} - (\nabla g(\boldsymbol{w}) \cdot \boldsymbol{N})\boldsymbol{N} &\quad (4.15)\\ &= \omega \boldsymbol{N}, &\quad (4.16)\end{aligned}$$

where $g(\boldsymbol{w}) = g(|\nabla u(C(t))|, |\nabla v(C(t))|)$ and

$$\omega = g(\boldsymbol{w})\kappa - \nabla g(\boldsymbol{w}) \cdot \boldsymbol{N}. \tag{4.17}$$

4.2 Level set formulation

The previous section derived an evolution of a parametrized curve. In this section the representation of the curve C by a parametrization will be replaced by a level set representation. It is now assumed that the evolving curve $C(\tau)$ can be described by a level set of some function φ, called the level set function. Instead of using (4.15) to evolve the curve C we would like to find an expression for the evolution of the level set function, and let this expression guide the evolution of the curve. The following geometric derivation is presented in [7], and the method was first proposed by Osher and Sethian [24]. We will assume that the zero level set of φ describes the curve C. To this end, let the zero level set Γ of φ be a closed curve dividing $\mathbb{R}^2$ into two regions, the interior $\mathcal{I}$ and the exterior $\mathcal{O}$, so that

$$\varphi(\boldsymbol{x},\tau) < 0 \qquad \text{for } \boldsymbol{x} \in \mathcal{I}(\tau), \tag{4.18}$$

$$\varphi(\boldsymbol{x},\tau) > 0 \qquad \text{for } \boldsymbol{x} \in \mathcal{O}(\tau), \tag{4.19}$$

$$\varphi(\boldsymbol{x},\tau) = 0 \qquad \text{for } \boldsymbol{x} \in \Gamma(\tau). \tag{4.20}$$

The goal is to find an evolution of $\varphi(\tau)$ such that $C(\tau) = \Gamma(\tau)$, given

$$C_\tau = \omega \boldsymbol{N}.$$

Differentiating (4.20) with respect to τ gives

$$\Gamma_\tau \varphi + \varphi_\tau = 0.$$

From equations (4.18) – (4.20) it is seen that the gradient at Γ is pointing outwards, that is

$$\frac{\nabla\varphi}{|\nabla\varphi|} = -\boldsymbol{N}.$$

Setting $\Gamma_\tau = C_\tau = \omega\boldsymbol{N}$ gives

$$\varphi_\tau = \omega|\nabla\varphi|.$$

This PDE governs the time-dependent evolution of the level sets of φ, with ω giving the speed in the normal direction [28]. Inserting the expression for ω given in (4.17) results in

$$\varphi_\tau = (g(\boldsymbol{w})\kappa - \nabla g(\boldsymbol{w}) \cdot \boldsymbol{N})|\nabla\varphi| \tag{4.21}$$

$$= g(\boldsymbol{w})\kappa|\nabla\varphi| + \nabla g(\boldsymbol{w}) \cdot \nabla\varphi, \tag{4.22}$$

where the term $g(\boldsymbol{w})\kappa - \nabla g(\boldsymbol{w}) \cdot \boldsymbol{N}$ is evaluated at the level sets of φ. From [7] we have that the curvature is given by

$$\kappa = \operatorname{div}\left(\frac{\nabla\varphi}{|\nabla\varphi|}\right), \tag{4.23}$$

so that (4.21) can be rewritten as

$$\begin{aligned} \frac{\partial\varphi}{\partial\tau} &= g(\boldsymbol{w})\operatorname{div}\left(\frac{\nabla\varphi}{|\nabla\varphi|}\right)|\nabla\varphi| + \nabla g(\boldsymbol{w}) \cdot \nabla\varphi \\ &= |\nabla\varphi|\operatorname{div}\left(g(\boldsymbol{w})\frac{\nabla\varphi}{|\nabla\varphi|}\right). \end{aligned}$$

Caselles et al. [7] noted that the level set evolution can get stuck at unwanted local minima, and thus argued that adding a velocity term $\gamma g(\boldsymbol{w})|\nabla\varphi|$ could be beneficial. This term is called the balloon force, and the authors of [7] pointed out that adding this term will increase the speed of convergence. The curve evolution with the added motion term is given by

$$\frac{\partial\varphi}{\partial\tau} = |\nabla\varphi|\operatorname{div}\left(g(\boldsymbol{w})\frac{\nabla\varphi}{|\nabla\varphi|}\right) + \gamma g(\boldsymbol{w})|\nabla\varphi|,$$

which is equivalent to

$$\frac{\partial\varphi}{\partial\tau} = |\nabla\varphi| g(\boldsymbol{w})(\kappa + \gamma) + \nabla g(\boldsymbol{w}) \cdot \nabla\varphi, \tag{4.24}$$

with the curvature κ given in equation (4.23). The existence and uniqueness of solutions have been shown using the theory of viscosity solutions [7, 1, 11], assuming a sufficiently regular image (or flow field in this setting) and initial data φ^0. The authors of [7] considered an ideal edge where $g \to 0$. Assuming an initial level set function φ^0 enclosing the objects, they proved that all level sets of this function will converge to this ideal edge with respect to the Hausdorff distance.

Our segmentation process is now driven by the gradient flow of a level set function. This gradient flow has two driving forces, namely the diffusion term $\operatorname{div}\left(g(\boldsymbol{w})\frac{\nabla\varphi}{|\nabla\varphi|}\right)$ and the balloon force term $\gamma g(\boldsymbol{w})|\nabla\varphi|$. The behaviour of the level set evolution depends on three parameters:

- The balloon force γ,
- The initial level set function φ^0,
- The edge detector function g, which also includes a sensitivity parameter η, for controlling how sensitive the edge detector is with respect to gradients.

Choosing appropriate values for these is not an easy task, and failing to do so might lead to unwanted behaviour. For instance, if the edge of the optical flow field is not strong enough or the sensitivity of the edge detector is chosen too low, the contour might cross the edge due to numerical inaccuracies. Also, since the balloon force results in an added velocity term, the zero level set will eventually cross the edges. If the level set evolves from the outer parts of the domain, the contour might cross through edges in the outer parts of the optical flow field before the whole object is completely segmented. This problem can be, to some extent, amended by choosing a steep initial function ϕ^0, however, a steep level set function is more sensitive to noise [12].

The behaviour discussed above is illustrated in figure 4.1, which shows an attempt to segment an optical flow field estimated from the Hamburg taxi sequence (the original image is shown in figure 2.1a). The zero level

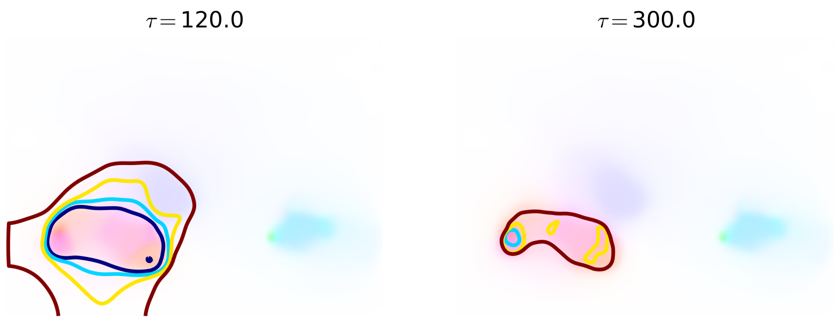


Figure 4.1: The problem of a vanishing zero level set. The dark blue line shows the zero level set, the light blue line shows the line $\varphi = 2$, the yellow line shows $\varphi = 4$ and the red line shows $\varphi = 6$.

set (dark blue line) is seen to enter the black car. As a result the active contour described as the zero level set is not able to segment the object correctly. Nonetheless, if the zero level set passes through the edge, other level sets of the function φ might give an accurate description of the wanted segmentation boundary. This is seen from the situation at $\tau = 300.0$ shown in figure 4.1. The level set $\varphi = 6$ (red line) gives a good description of the boundary of the optical flow field for the black car. Thus, in the case of vanishing zero level sets, one may want to consider more than one level set of φ in the estimation of the segmentation boundary. This is the motivation for the next approach, namely a Mumford-Shah type segmentation of the level set function.

5 A modified Mumford-Shah segmentation

The Mumford-Shah segmentation functional, proposed by Mumford and Shah [21] in 1989, establishes an energy that can be used to segment an image into subregions. The original functional is given as

$$E(f,C) = \alpha \int_{\Omega} (f_0 - f)^2 \, dx + \int_{\Omega \setminus C} |\nabla f|^2 \, dx + \beta L(C), \tag{5.1}$$

where f_0 is the image, C is some closed curve dividing the image domain Ω into two parts, $L(C)$ is the length of the curve and α and β are positive constants. In the Mumford-Shah functional the first term tries to approximate the image f_0, the second term penalizes high gradients and tries to prevent f from varying too much, while the third term penalizes long segmentation boundaries C. When minimizing this energy the resulting function f will be smooth inside C, with a sharp boundary at C. Versions of the functional have been widely used for the segmentation of images, also in combination with the active contours approach [9]. As proposed by Fuchs et al. [12], we will use a modified version of this functional for segmenting the active contour driven level set function.

5.1 Segmenting the level set function

In section 4.2 the problem of vanishing boundaries was explained and illustrated using an example from the Hamburg taxi sequence. But, as figure 4.1 shows, the level set function can still give an estimation for the segmentation boundaries through other level sets. Fuchs et al. [12] remarked that by using a Mumford-Shah type segmentation of the level set function φ, one can extract information from all level sets of φ instead of just one. This approach will be followed here, using a modified version of the Mumford-Shah functional,

$$I(C) = \alpha \int_{\text{int}(C)} (\overline{\varphi}_{int} - \varphi)^2 \, dx + \alpha \int_{\text{ext}(C)} (\overline{\varphi}_{ext} - \varphi)^2 \, dx + \beta L(C), \tag{5.2}$$

where C is some closed curve. We use int(C) and ext(C) to denote the part of Ω lying inside and outside the curve C respectively, and

$$\overline{\varphi}_{int} = \frac{1}{A_{int}} \int_{\text{int}(C)} \varphi \, dx \qquad \overline{\varphi}_{ext} = \frac{1}{A_{ext}} \int_{\text{ext}(C)} \varphi \, dx. \tag{5.3}$$

The areas A_{int} and A_{ext} are the areas of the regions int(C) and ext(C) respectively. Given a level set function φ and parameters $\alpha, \beta > 0$, the aim is to solve the optimization problem

$$\underset{C}{\text{minimize}} \quad I(C) \tag{5.4}$$

for closed curves $C : [0,1] \to \mathbb{R}^2$. The first two terms will try to find a closed curve that minimizes the variance of the level set function inside and outside the curve respectively, while the second term will aim to minimize the length of the curve. The problem, which is often referred to as the minimal partition problem, was also studied in [21]. In [20, p. 47] the existence of a minimizer was proved for a measurable bounded function φ, and an extensive treatment of the regularity of solutions can be found there. Alternatively, see [2, chapter 4, p. 154] for a shorter treatment.

The functional can be minimized by using a gradient descent method, and so we want to proceed by calculating a gradient to the functional. Theoretically this is challenging, as one needs some inner product on the space of all curves for a theoretical gradient to exist. However, one can calculate the variation and define a formal gradient. To compute the variation of (5.2) with respect to C, we start by assuming that the boundary $C = C(\tau)$ deforms with velocity $\frac{dx}{d\tau} = \boldsymbol{v}(\tau, x)$. For simplicity, it is also assumed that the curve is traced with unit speed, so that $|C'| = 1$. We follow the ideas of Aubert and Kornprobst [2, chapter 4, p. 158–161], and define the function

$$f(\tau) = \alpha \int\limits_{\text{int}(C(\tau))} (\overline{\varphi}_{int}(\tau) - \varphi)^2 \, dx + \alpha \int\limits_{\text{ext}(C(\tau))} (\overline{\varphi}_{ext}(\tau) - \varphi)^2 \, dx + \beta \int\limits_{C(\tau)} ds,$$

where the length is given as

$$L(C) = \int\limits_{C} ds.$$

When computing the derivative of the first two integrals, we will use Leibniz' integral rule for the derivative of a domain integral, more commonly known as Reynolds transport theorem. This says that if $l(\tau, x)$ is a regular function defined on a regular domain $w(\tau)$ of $\mathbb{R}^2$, and

$$g(\tau) = \int\limits_{w(\tau)} l(\tau, x) \, dx,$$

then

$$g'(\tau) = \int\limits_{w(\tau)} \frac{\partial l}{\partial \tau}(\tau, x) \, dx + \int\limits_{\partial w(\tau)} l(\tau, x) \boldsymbol{v} \cdot \boldsymbol{N} \, ds, \tag{5.5}$$

where $\partial w(\tau)$ is the boundary of $w(\tau)$ with outward normal $\boldsymbol{N}$, moving with velocity $\boldsymbol{v}$. For the derivative of the last integral, note that

$$\frac{d}{d\tau}\left(\int\limits_{C(\tau)} d\sigma\right) = \int\limits_{C(\tau)} \kappa \boldsymbol{v}\cdot\boldsymbol{N}\, ds, \tag{5.6}$$

where κ is the curvature of $C(\tau)$. To see this, consider (4.10) with the choice $g(C) = 1$. This choice results in $\nabla g = 0$, and thus, by using (4.13) it is seen that the variation simplifies to the expression above. The minus sign is due to $\boldsymbol{N}$ denoting an inward normal in (4.13) and an outward normal here. Using (5.5) and (5.6) we get

$$\begin{aligned} f'(\tau) = \alpha \Bigg(& 2\int\limits_{\text{int}(C(\tau))} (\overline{\varphi}_{int}(\tau)-\varphi)\,\frac{\partial\overline{\varphi}_{int}}{\partial\tau}(t)\,dx + \int\limits_{C(\tau)} (\overline{\varphi}_{int}(\tau)-\varphi)^2\, \boldsymbol{v}\cdot\boldsymbol{N}\,ds \\ & + 2\int\limits_{\text{ext}(C(\tau))} (\overline{\varphi}_{ext}(\tau)-\varphi)\,\frac{\partial\overline{\varphi}_{ext}}{\partial\tau}(\tau)\,dx - \int\limits_{C(\tau)} (\overline{\varphi}_{ext}(\tau)-\varphi)^2\, \boldsymbol{v}\cdot\boldsymbol{N}\,d\sigma \Bigg) \\ & + \beta\int\limits_{C(\tau)} \kappa\boldsymbol{v}\cdot\boldsymbol{N}\,ds. \end{aligned}$$

As $\overline{\varphi}_{int}(\tau)$ and $\overline{\varphi}_{ext}(\tau)$ are piecewise constant functions on $\text{int}(C(\tau))$ and $\text{ext}(C(\tau))$, these functions, along with their time derivatives, can be put outside the integrals in the above expression. This gives

$$\int\limits_{\text{int}(C(\tau))} (\overline{\varphi}_{int}(\tau)-\varphi)\,\frac{\partial\overline{\varphi}_{int}}{\partial\tau}(\tau)\,dx = \frac{\partial\overline{\varphi}_{int}}{\partial\tau}(\tau)\left(A_{int}\overline{\varphi}_{int}(\tau) - \int\limits_{\text{int}(C(\tau))} \varphi\,dx\right),$$

$$\int\limits_{\text{ext}(C(\tau))} (\overline{\varphi}_{ext}(\tau)-\varphi)\,\frac{\partial\overline{\varphi}_{ext}}{\partial\tau}(\tau)\,dx = \frac{\partial\overline{\varphi}_{ext}}{\partial\tau}(\tau)\left(A_{ext}\overline{\varphi}_{ext}(\tau) - \int\limits_{\text{ext}(C(\tau))} \varphi\,dx\right).$$

Now, using the definitions of $\overline{\varphi}_{int}(\tau)$ and $\overline{\varphi}_{ext}(\tau)$ given in (5.3), it is seen that

$$A_{int}\overline{\varphi}_{int}(\tau) - \int\limits_{\text{int}(C(\tau))} \varphi\,dx = 0,$$

$$A_{ext}\overline{\varphi}_{ext}(\tau) - \int\limits_{\text{ext}(C(\tau))} \varphi\,dx = 0.$$

As a result, the derivative $f'(\tau)$ is reduced to

$$f'(\tau) = \int\limits_{C(\tau)} \Big(\alpha\,(\overline{\varphi}_{int}(\tau) - \varphi)^2 - \alpha\,(\overline{\varphi}_{ext}(\tau) - \varphi)^2 + \beta\kappa\Big)\boldsymbol{v}\cdot\boldsymbol{N}\,ds.$$

We are interested in expressing the curvature κ in terms of the parametrized curve C. To do this, recall that the curvature can be expressed as

$$\kappa\boldsymbol{N} = -\boldsymbol{T}_s,$$

where $\boldsymbol{T}$ is the unit tangent vector, $\boldsymbol{N}$ is the unit outward normal, and s is the arc-length of C. This unit tangent vector, as a function of arc-length, can be written as

$$T(s) = C'(s),$$

and it is easily verified that this is indeed a unit vector. This leads to

$$\kappa\boldsymbol{N} = -C''(s),$$

and thus

$$f'(\tau) = \int\limits_{C(\tau)} \Big(\alpha\Big[(\overline{\varphi}_{int}(\tau) - \varphi)^2 - (\overline{\varphi}_{ext}(\tau) - \varphi)^2\Big]\boldsymbol{N} - \beta C''\Big)\cdot\boldsymbol{v}\,ds.$$

To find a variation for all parametrized closed curve, and not just the ones with $|C'|= 1$, note that

$$ds = |C'|dt,$$

where t is the parametrization variable. Thus, the derivative can be rewritten as

$$f'(\tau) = \int\limits_{C(\tau)} \Big(\alpha\Big[(\overline{\varphi}_{int}(\tau) - \varphi)^2 - (\overline{\varphi}_{ext}(\tau) - \varphi)^2\Big]|C'|\boldsymbol{N} - \beta|C'|C''\Big)\cdot\boldsymbol{v}\,dt.$$

Formally this is an expression for the directional derivative of the functional $I(C)$ in direction $\boldsymbol{v}$. The formal gradient is then defined as

$$\nabla I(C) = \alpha\Big[(\overline{\varphi}_{int}(\tau) - \varphi)^2 - (\overline{\varphi}_{ext}(\tau) - \varphi)^2\Big]|C'|\boldsymbol{N} - \beta|C'|C'', \tag{5.7}$$

which is an element in an infinite-dimensional space. We will use a gradient descent method to drive the evolution of the curve, and thus we set

$$\frac{\partial C}{\partial \tau} = -\nabla I(C(\tau)). \tag{5.8}$$

Combining this evolution with the evolution of the level set, we have now obtained a system for a combined evolution of the level set function and the curve C:

$$\begin{aligned}
\varphi(0) &= \varphi^0, \\
C(0) &= C^0, \\
\frac{\partial \varphi}{\partial \tau} &= |\nabla \varphi| \operatorname{div}\left(g(w) \frac{\nabla \varphi}{|\nabla \varphi|} \right) + \gamma g(w) |\nabla \varphi|, \\
\frac{\partial C}{\partial \tau} &= -\nabla I(C(\tau)).
\end{aligned} \tag{5.9}$$

5.2 Using splines for representing the segmentation boundary

As previously mentioned, the result of a segmentation process can either be described as a region in Ω, or a curve enclosing this region. In the framework described in the previous section it was assumed the segmentation boundary can be described as a parametrized curve, which is evolve according to equation (5.8). Now, we let the parametrized curve $C(\tau)$ be represented as a periodic cubic B-spline curve. To this end, let $\mathcal{C}^1_p([0,1],\mathbb{R}^2)$ be the space of continuously differentiable and periodic curves with the L^2-norm. Further, let $C(\tau)$ be interpreted as a mapping from K control points

$$p(\tau) = (p_1(\tau), p_2(\tau), ..., p_K(\tau)), \tag{5.10}$$

where $p_k(\tau) = (p^1_k(\tau), p^2_k(\tau))$, to a parametrized curve

$$C : (\mathbb{R}^2)^K \to \mathcal{C}^1_p([0,1],\mathbb{R}^2).$$

Additionally, denote by $\psi = (\psi^k : [0,1] \to \mathbb{R})_{1 \le k \le K}$ the basis of periodic cubic B-splines with uniformly distributed knots. The mapping above is then given by

$$C(p) = \sum_{k=1}^{K} p_k(\tau) \psi^k,$$

where $p_k(\tau)$ are the time evolving spline control points. We now want to map the evolution of the curve to an evolution of the spline control points. To this end, let $DC(p)$ denote the derivative of the curve with respect to the spline control points. Then $DC(p)$ is a linear mapping that maps a vector $v \in (\mathbb{R}^2)^K$ to a curve $DC(p)v \in \mathcal{C}^1_p([0,1],\mathbb{R}^2)$. The gradient descent evolution can now be written as

$$DC(p) \frac{\partial p}{\partial \tau} = -\nabla I\,(C\,(p)).$$

This is an overdetermined system, since p is a vector of control points with length K and the gradient on the right hand side is defined on an infinite-dimensional space. Thus, to solve this equation we minimize

$$\|DC(p)\frac{\partial p}{\partial \tau} + \nabla I(C)\|^2$$

with respect to the derivative $\frac{\partial p}{\partial \tau}$. This is a convex quadratic minimization problem, and it is solved by the normal equations,

$$DC(p)^*\left(DC(p)\frac{\partial p}{\partial \tau} + \nabla I(C)\right) = 0,$$

where $DC(p)^*$ denotes the adjoint of $DC(p)$. Since $DC(p)v \in \mathcal{C}^1_p([0,1],\mathbb{R}^2)$ is a B-spline for any $v \in (\mathbb{R}^2)^K$, we can write

$$DC(p)v = \sum_k^K v_k \psi^k. \tag{5.11}$$

From the definition of the adjoint operator, we have for some $h \in C^1\left([0,1],\mathbb{R}^2\right)$

$$\langle DC(p)^* h, v\rangle_{(\mathbb{R}^2)^K} = \langle h, DC(p)v\rangle_{L^2\left([0,1],\mathbb{R}^2\right)},$$

where the inner product on the left hand side is taken in the space of control points, and the inner product on the right hand side is an L^2 inner product taken in the space of continuously differentiable functions. From (5.11) we get

$$\begin{aligned}\langle h, DC(p)v\rangle_{L^2([0,1],\mathbb{R}^2)} &= \int_0^1 h(t)\sum_{k=1}^K v_k \psi^k(t)\,dt \\ &= \sum_{k=1}^K v_k \int_0^1 h(t)\psi^k(t)\,dt \\ &= \langle v, \vartheta\rangle_{(\mathbb{R}^2)^K},\end{aligned}$$

where

$$\vartheta = \left[\int_0^1 h(t)\psi^k(t)\,\mathrm{dt}\right]_{k=1}^K$$

denotes an element in $(\mathbb{R}^2)^K$. As the inner product is symmetric in $(\mathbb{R}^2)^K$ we can conclude that

$$DC(p)^* h = \vartheta$$

for every $h \in C^1([0,1],\mathbb{R}^2)$. Letting

$$h = DC(p)\frac{\partial p}{\partial \tau},$$

it is seen that, using equation (5.11),

$$\begin{aligned} DC(p)^* DC(p)\frac{\partial p}{\partial \tau} &= \left[\int_0^1 \sum_j^K \frac{\partial p_j}{\partial \tau}\psi^j(t)\psi^k(t)\,\mathrm{d}t\right]_{k=1}^K \\ &= \left[\sum_j^K \frac{\partial p_j}{\partial \tau}\int_0^1 \psi^j(t)\psi^k(t)\,\mathrm{d}t\right]_{k=1}^K \\ &= A\frac{\partial p}{\partial \tau}, \end{aligned}$$

where

$$A_{ij} = \int_0^1 \psi^i(t)\psi^j(t)\,\mathrm{d}t \quad \text{and} \quad \frac{\partial p}{\partial \tau} = \left[\frac{\partial p_k}{\partial \tau}\right]_{k=1}^K. \tag{5.12}$$

The system governing the evolution of the control points can now be written as

$$\begin{aligned} A\frac{\partial p}{\partial \tau}^c &= -\left[\int_0^1 \nabla I(C(p))(t)^c \psi^k(t)\,\mathrm{d}t\right]_{k=1}^K && (5.13) \\ &= \Phi^c, && (5.14) \end{aligned}$$

where $\Phi \in (\mathbb{R}^K)^2$ and $c \in \{1,2\}$ denotes the spatial component. This is a system for the evolution of the control points of the periodic cubic B-spline curve. The combined evolution can now be stated as

$$\begin{aligned} \varphi(0) &= \varphi^0, \\ p(0) &= p^0, \\ \frac{\partial \varphi}{\partial \tau} &= |\nabla\varphi|\,\mathrm{div}\left(g(\boldsymbol{w})\frac{\nabla\varphi}{|\nabla\varphi|}\right) + \gamma g(\boldsymbol{w})|\nabla\varphi|, \\ A\frac{\partial p}{\partial \tau}^c &= \Phi^c(C(p(\tau))), \end{aligned} \tag{5.15}$$

for $c \in \{1,2\}$. The next chapter will give the numerical details on how to solve this system.

6 Implementation

This chapter presents the numerical details for the optical flow estimation and the segmentation process. Recall that the system to solve for computing the optical flow can be formulated as the coupled system of PDEs

$$\begin{aligned} \partial_u M(\boldsymbol{w}) - \frac{1}{\xi^2}\operatorname{div}(\Theta_u \nabla u) = 0, \\ \partial_v M(\boldsymbol{w}) - \frac{1}{\xi^2}\operatorname{div}(\Theta_v \nabla v) = 0, \end{aligned} \tag{6.1}$$

where the matrices $\Theta_u = \Theta_u(x^1, x^2, \nabla u, \nabla v)$ and $\Theta_v = \Theta_v(x^1, x^2, \nabla u, \nabla v)$ are called the diffusion matrices, since they control the direction and strength of the diffusion process. The segmentation process is driven by

$$\begin{aligned} \frac{\partial \varphi}{\partial \tau} &= |\nabla\varphi| \operatorname{div}\left(g(\boldsymbol{w}) \frac{\nabla\varphi}{|\nabla\varphi|} \right) + \gamma g(\boldsymbol{w}) |\nabla\varphi|, \\ A\frac{\partial p^c}{\partial \tau} &= \Phi^c\left(C\left(p\left(\tau\right)\right)\right) \text{ for } c \in \{1,2\}. \end{aligned} \tag{6.2}$$

Note that these systems, (6.1) and (6.2), are partially coupled in the sense that the solution of the latter depends on the solution of the former, but not the other way around. In this thesis, we will solve them separately, solving (6.1) to find a flow field $\boldsymbol{w} = (u, v)$, and then segmenting moving regions of this flow field by solving the combined evolution (6.2) and ultimately finding the cubic B-spline that describes this region. We will start by outlining the numerical methods for solving the optical flow system.

6.1 Solving the optical flow system

Let Ω denote a flattened $m \times n$ pixel grid, so that the mapping from a pixel at position $x = i$ in the flattened grid is given by

$$(x^1, x^2) = (\lfloor i/m \rfloor, i - \lfloor i/m \rfloor),$$

where (x^1, x^2) denotes the coordinates of the rectangular grid.

6.1.1 The data term

Recall from chapter 3 that the contribution to the Euler-Lagrange system coming from the data term is

$$\partial_{\boldsymbol{w}} M = 2\frac{\nabla f^T \boldsymbol{w} + f_t}{|\nabla f|^2 + \zeta^2} + 2\gamma_{OF}\left(\frac{\nabla f_{x^1}^T \boldsymbol{w} + f_t}{|\nabla f_{x^1}|^2 + \zeta^2} + \frac{\nabla f_{x^2}^T \boldsymbol{w} + f_t}{|\nabla f_{x^2}|^2 + \zeta^2} \right).$$

From the above, it is seen that we need to find approximations for the spatial derivatives and the time derivatives. As the distance between grid points in fixed, there is little to gain from choosing a higher order derivative approximation. Thus, we will use the forward difference to approximate the derivatives in both time and space. Let g be a vector in $\mathbb{R}^K$, the forward differences are defined as the operators

$$\begin{aligned}\delta_{x^1}^{fw} g_k &= g_{k+m} - g_k,\\ \delta_{x^2}^{fw} g_k &= g_{k+1} - g_k,\end{aligned}$$

where $\delta_{x^1}^{fw}$ and $\delta_{x^2}^{fw}$ denotes the forward difference in directions x_1 and x_2 respectively.

6.1.2 The divergence operator

The divergence operator in the smoothness term will be approximated by a backward difference. The backward difference operators are defined as

$$\begin{aligned}\delta_{x^1}^{bw} g_k &= g_k - g_{k-m},\\ \delta_{x^2}^{bw} g_k &= g_k - g_{k-1},\end{aligned}$$

where $\delta_{x^1}^{bw}$ and δbw_{x^2} denote the backward differences in directions x_1 and x_2, respectively. The divergence operator is then approximated as

$$\begin{aligned}\operatorname{div}(\nabla g_k) &\approx \delta_{x^1}^{bw}\delta_{x^1}^{fw} g_k + \delta_{x^2}^{bw}\delta_{x^2}^{fw} g_k\\ &= \delta_{x^1}^{bw}(g_{k+m} - g_k) + \delta_{x^2}^{bw}(g_{k+1} - g_k)\\ &= (g_{k+m} - 2g_k + g_{k-m}) + (g_{k+1} - 2g_k + g_{k-1}),\end{aligned}$$

or what is more commonly known as the second order central difference.

6.1.3 The boundary conditions

The optical flow system is solved using Neumann boundary condition, which says that the normal derivatives on the boundaries vanish. For the eastern boundary Γ_E and the southern boundary Γ_S we will impose that the backward difference is zero in x^1- and x^2-direction respectively. For our flattened grid, this leads to

$$\begin{aligned}u(x-m) &= u(x),\\ v(x) &= v(x),\end{aligned}$$

for $x \in \Gamma_E$. Equivalently, when $x \in \Gamma_S$,

$$\begin{aligned}u(x-1) &= u(x),\\ v(x-1) &= v(x).\end{aligned}$$

On the two other boundaries, the western boundary Γ_W and the northern boundary Γ_N, we will enforce the forward differences to be zero. For $x \in \Gamma_W$ this leads to

$$\begin{aligned} u(x) &= u(x+m), \\ v(x) &= v(x+m), \end{aligned}$$

and likewise for $x \in \Gamma_N$,

$$\begin{aligned} u(x) &= u(x+1), \\ v(x) &= v(x+1). \end{aligned}$$

These conditions are enforced on the boundary when solving the system.

6.1.4 The lagged diffusivity iterations

Recall that the flow driven regularization lead to a nonlinear system. We proposed to solve this system using the method of lagged diffusivity. The iteration scheme can be formulated as

$$\begin{aligned} \partial_{u^{k+1}} M - \frac{1}{\xi^2}\left(\frac{\partial}{\partial x}\left[\frac{u^{k+1}_{x^1}}{\sqrt{|\nabla u^k|^2 + |\nabla v^k|^2 + \epsilon^2_{OF}}}\right] + \frac{\partial}{\partial y}\left[\frac{u^{k+1}_{x^2}}{\sqrt{|\nabla u^k|^2 + |\nabla v^k|^2 + \epsilon^2_{OF}}}\right]\right) &= 0, \\ \partial_{v^{k+1}} M - \frac{1}{\xi^2}\left(\frac{\partial}{\partial x}\left[\frac{v^{k+1}_{x^1}}{\sqrt{|\nabla u^k|^2 + |\nabla v^k|^2 + \epsilon^2_{OF}}}\right] + \frac{\partial}{\partial y}\left[\frac{v^{k+1}_{x^2}}{\sqrt{|\nabla u^k|^2 + |\nabla v^k|^2 + \epsilon^2_{OF}}}\right]\right) &= 0. \end{aligned}$$

To improve the computation time of the iterative scheme, we will use GMRES to solve the linear system in each iteration. For details on the GMRES algorithm we refer to [26, chapter 6, p. 164].

6.2 Solving the segmentation system

The segmentation system (6.2) evolves the level set function and the cubic B-spline function. Since the evolution of the spline curve depends on the evolution of the level set function, but not the other way around, the system is partially decoupled. Using the terminology of [12], we denote the algorithm evolving the level set and the curve simultaneously as *the combined evolution*. One iteration of the combined evolution will consist of a specified number of level set evolutions and a specified number of curve evolutions. In the discussion of the segmentation results we will use a special notation to denote the number of level set iterations and spline iterations in the combined evolution. If one iteration of the combined evolution consists of k level set iteration and l spline iterations, the evolution is called a "$k : l$ combined evolution scheme" or simply a "$k : l$ combined evolution". The number of iterations used for each combined evolution scheme will be given in each experiment. The following subsections will present the numerical framework for these evolutions. A notational change is made from the previous section; the domain Ω is now a rectangular pixel grid of size $m \times n$. We will start by discretizing the evolution the level set function.

6.2.1 The level set evolution

Given a level set solution $\varphi(\tau)$ and a time step $\delta > 0$, the authors of [12] used the semi-implicit time discretization

$$\frac{\varphi(\tau+\delta)-\varphi(\tau)}{\delta} = |\nabla\varphi(\tau)|\operatorname{div}\left(g(\boldsymbol{w})\frac{\nabla\varphi(\tau+\delta)}{|\nabla\varphi(\tau)|}\right) + \gamma g(\boldsymbol{w})|\nabla\varphi(\tau)|. \tag{6.3}$$

to find the level set at $\varphi(\tau+\delta)$. A finite difference scheme is used to solve this, where the first derivatives are computed using a forward difference approximation (shown in section 6.1.1), and the divergence is computed as in section 6.1.2.

The boundary condition

A Neumann boundary condition is used For the level set function. As argued in [1], the image Ω is some part of a larger scene, thus it is natural to impose Neumann boundary conditions on the level set solution. This condition is given as

$$\nabla\varphi \cdot \boldsymbol{N} = 0 \quad \text{on } \partial\Omega \tag{6.4}$$

for the level set function, where $\partial\Omega$ is the boundary of Ω. Similarly to the optical flow components, this condition results in equations for the value of the level set function at the boundary pixels. These equations are equivalent to the ones for the flow field, and will be imposed in each evolution of the level set function.

The initial data

As proposed in [7] we will use the signed distance function to some curve σ as the initial level set function φ^0. The signed distance function of σ can be defined as

$$\overline{d}(\boldsymbol{x},\sigma) = \begin{cases} d(\boldsymbol{x},\sigma) & \text{if } \boldsymbol{x} \in \operatorname{ext}(\sigma), \\ -d(\boldsymbol{x},\sigma) & \text{if } \boldsymbol{x} \in \operatorname{int}(\sigma), \end{cases} \tag{6.5}$$

where $d(\boldsymbol{x},\sigma)$ is the Euclidean distance. We will use some scaling of this function, defining a scaling parameter s, and setting $\varphi(\boldsymbol{x}) = s\overline{d}(\boldsymbol{x},\sigma)$. As Fuchs et al. [12] noted, steeper level set functions are more sensitive to noise. Thus, in flow fields with a lot of noise or irregularity, we may choose a low value for s to obtain global stability for our level set evolution. The curve σ used in the defininition of the initial level set function, will always be a circle, where the moving object is at least partly inside σ. The signed distance function satisfies

$$|\nabla \bar{d}(\boldsymbol{x}, \sigma)| = 1,$$

and thus,

$$|\nabla \varphi^0| = s. \tag{6.6}$$

ϵ-regularization

In order to avoid numerical issues for small values of $|\nabla \varphi|$ a regularization term is added [12]. In the numerical computations we will replace $\sqrt{\varphi_{x1}^2 + \varphi_{x2}^2}$ with $\sqrt{\varphi_{x1}^2 + \varphi_{x2}^2 + \epsilon}$, and the approximation will be called the ϵ-regularization. From (6.6) we have that the gradient of the initial function is scaled by the parameter s, and thus the regularization parameter ϵ should be scaled to match this scaling. However, choosing a too low value for ϵ might lead to numerical issues. Hence, a compromise must be made between numerical stability and the scaling of ϵ.

Reinitialization

As noted in [2, chapter 4, p. 194], the gradients of our level set function may become unbounded, which can cause the curve evolution to become unstable. Thus, in some cases it might be necessary to reinitialize the level set function. This is done by simply setting the level set function to the signed distance function we started with, but more advanced methods of reinitialization have been proposed (see [2, chapter 4, p. 194]). The need to reinitialize can be assessed by measuring the size of the norm of the gradient φ. To this end, define the energy

$$H(\varphi) = \int_\Omega |\nabla \varphi| \, dx. \tag{6.7}$$

If this energy becomes relatively large, we may need to reinitialize the level set function. This function is computed using the forward difference approximation for the gradient, and summing up the values over the discretized domain Ω. This provides a sufficiently accurate estimate of the energy for the purpose of determining the need for reinitialization.

The edge detector

Details of the edge detector $g(\nabla u, \nabla v)$ has not yet been discussed, other than the properties

$$g(r, q) \to 0 \text{ as } r \to \infty \text{ or } q \to \infty.$$

A common choice [8] in a segmentation controlled by image edges is the edge detector

$$g = \frac{1}{1+\eta|\nabla f|^p}, \tag{6.8}$$

for $p = 1$ or $p = 2$, where η is some sensitivity parameter. Thus, in a segmentation process steered by the edges of the flow, we propose to use

$$g = \frac{1}{1+\eta\,(|\nabla u|^p + |\nabla v|^p)}. \tag{6.9}$$

This function would stop the evolving contour at what the authors of [7] considered an "ideal" edge. In our numerical experiments we have used $p = 1$.

6.2.2 The curve evolution

For representing the curve $C(\tau)$ we will use periodic cubic B-splines, for which an example is shown in figure 6.1. We now interpret $C(\tau)$ as a mapping from K control points

$$p(\tau) = (p_1(\tau), p_2(\tau), ..., p_K(\tau)),$$

where $p_k(\tau) = (p_k^x(\tau), p_k^y(\tau))$, to a parametrized curve

$$C : (\mathbb{R}^2)^K \to \mathcal{C}_p^1([0,1], \mathbb{R}^2).$$

Let $\psi = (\psi^k : [0,1] \to \mathbb{R})_{1\le k\le K}$ be the basis of periodic cubic B-splines with uniformly distributed knots. The mapping above is then given by

$$C(p) = \sum_{k=1}^{K} p_k(\tau)\psi^k,$$

where $p_k(\tau)$ are the time evolving spline control points. Each basis function $\psi^k(t)$ is a piecewise cubic polynomial defined over an interval of five knots with a uniform knot spacing of $t_k - t_{k-1} = \frac{1}{K}$, such that

$$\psi^k(t) = \begin{cases} a(K(t-t_{k-2})) & \text{for } t_{k-2} \le t \le t_{k-1}, \\ b(K(t-t_{k-1})) & \text{for } t_{k-1} \le t \le t_k, \\ c(K(t-t_k)) & \text{for } t_k \le t \le t_{k+1}, \\ d(K(t-t_{k+1})) & \text{for } t_{k+1} \le t \le t_{k+2}, \end{cases} \tag{6.10}$$

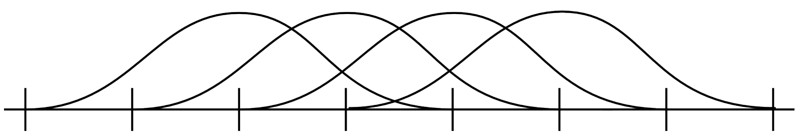


Figure 6.1: Example of a series of cubic B-splines.

where the functions $a(s)$, $b(s)$, $c(s)$ and $d(s)$ are defined as

$$a(s) = \frac{s^3}{6}, \tag{6.11}$$

$$b(s) = \frac{-3s^3 + 3s^2 + 3s + 1}{6}, \tag{6.12}$$

$$c(s) = \frac{3s^3 - 6s^2 + 4}{6}, \tag{6.13}$$

$$d(s) = \frac{-s^3 + 3s^2 - 3s + 1}{6}, \tag{6.14}$$

for $s \in [0,1]$. These elements are shown in figure 6.2. This essentially means that for any given value $\tilde{t}$ on any line element $[t_k, t_{k+1}] \subset [0,1]$, the parametrized curve is evaluated as a sum of these four function elements. Let $\gamma(t) \in \mathcal{C}_p^1([0,1], \mathbb{R}^2)$ be the parametrized curve. Then, using the transformation

$$s(t) = K(t - t_k) \in [0,1], \tag{6.15}$$

we get

$$\gamma(\tilde{t}) = p_{k+2}a(s(\tilde{t})) + p_{k+1}b(s(\tilde{t})) + p_k c(s(\tilde{t})) + p_{k-1}d(s(\tilde{t})) \tag{6.16}$$

The transformation (6.15) linearly maps the subinterval $[t_k, t_{k+1}]$ to the unit interval.

Computing derivatives

To find the derivative of the curve with respect to the parametrization variable t we need to know the derivatives[1] of the basis functions. As previously mentioned, a basis function will have compact support over five knots, and the derivative of each basis function will have compact support over these five knots. For each element, the curve is a sum of the four basis functions shown in figure 6.2, and so the evaluation of the derivative at a point $\tilde{t} \in [0,1]$ is a sum of four derivatives. For the basis function ψ^k shown

[1] The differentials in this subsection is denoted by an "upright" d in order to avoid confusion with the basis function d.

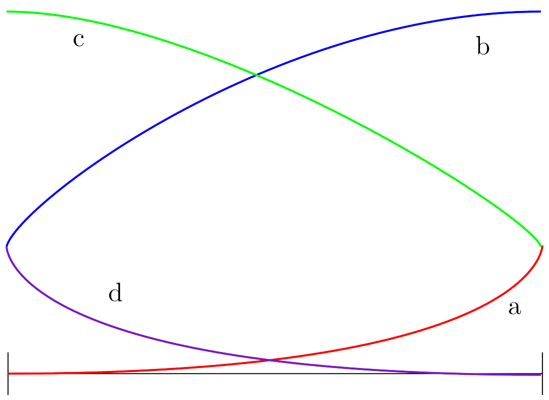


Figure 6.2: Contributing basis functions inside each element.

in equation (6.10) we obtain the derivative

$$\frac{\mathrm{d}\psi^k}{\mathrm{d}t}(t) = \begin{cases} \frac{\mathrm{d}a}{\mathrm{d}t}(K(t-t_{k-2})) & \text{for } t_{k-2} \le t \le t_{k-1}, \\ \frac{\mathrm{d}b}{\mathrm{d}t}(K(t-t_{k-1})) & \text{for } t_{k-1} \le t \le t_k, \\ \frac{\mathrm{d}c}{\mathrm{d}t}(K(t-t_k)) & \text{for } t_k \le t \le t_{k+1}, \\ \frac{\mathrm{d}d}{\mathrm{d}t}(K(t-t_{k+1})) & \text{for } t_{k+1} \le t \le t_{k+2}, \end{cases} \tag{6.17}$$

where the derivatives are computed as

$$\begin{aligned} \frac{\mathrm{d}a}{\mathrm{d}t}(s) &= \frac{\mathrm{d}s}{\mathrm{d}t}\frac{\mathrm{d}a}{\mathrm{d}s}(s) = K\frac{s^2}{2}, \\ \frac{\mathrm{d}b}{\mathrm{d}t}(s) &= \frac{\mathrm{d}s}{\mathrm{d}t}\frac{\mathrm{d}b}{\mathrm{d}s}(s) = K\frac{-3s^2+2s+1}{2}, \\ \frac{\mathrm{d}c}{\mathrm{d}t}(s) &= \frac{\mathrm{d}s}{\mathrm{d}t}\frac{\mathrm{d}c}{\mathrm{d}s}(s) = K\frac{3s^2-4s}{2}, \\ \frac{\mathrm{d}d}{\mathrm{d}t}(s) &= \frac{\mathrm{d}s}{\mathrm{d}t}\frac{\mathrm{d}d}{\mathrm{d}s}(s) = K\frac{-s^2+2s-1}{2}, \end{aligned}$$

for $s \in [0,1]$. Letting $\gamma(t)$ be the parametrized curve, we can now evaluate the derivative at a point $\tilde{t} \in [t_k, t_{k+1}] \subset [0,1]$ as

$$\frac{\mathrm{d}\gamma}{\mathrm{d}t}(\tilde{t}) = K\left(p_{k+2}\frac{\mathrm{d}a}{\mathrm{d}s}(s(\tilde{t})) + p_{k+1}\frac{\mathrm{d}b}{\mathrm{d}s}(s(\tilde{t})) + p_k\frac{\mathrm{d}c}{\mathrm{d}s}(s(\tilde{t})) + p_{k-1}\frac{\mathrm{d}d}{\mathrm{d}s}(s(\tilde{t}))\right),$$

where as before, we have used the transformation $s(t) = K(t-t_k) \in [0,1]$.

The second derivatives are computed in the same manner with the same set of compact support for each basis function. That is,

$$\frac{\mathrm{d}^2\psi^k}{\mathrm{d}t^2}(t) = \begin{cases} \frac{\mathrm{d}^2a}{\mathrm{d}t^2}(K(t-t_{k-2})) & \text{for } t_{k-2} \le t \le t_{k-1}, \\ \frac{\mathrm{d}^2b}{\mathrm{d}t^2}(K(t-t_{k-1})) & \text{for } t_{k-1} \le t \le t_k, \\ \frac{\mathrm{d}^2c}{\mathrm{d}t^2}(K(t-t_k)) & \text{for } t_k \le t \le t_{k+1}, \\ \frac{\mathrm{d}^2d}{\mathrm{d}t^2}(K(t-t_{k+1})) & \text{for } t_{k+1} \le t \le t_{k+2}, \end{cases} \tag{6.18}$$

with the second derivatives of the polynomials given as

$$\frac{\mathrm{d}^2 a}{\mathrm{d}t^2}(s) = \left(\frac{\mathrm{d}s}{\mathrm{d}t}\right)^2 \frac{\mathrm{d}^2 a}{\mathrm{d}s^2}(s) = K^2 s,$$

$$\frac{\mathrm{d}^2 b}{\mathrm{d}t^2}(s) = \left(\frac{\mathrm{d}s}{\mathrm{d}t}\right)^2 \frac{\mathrm{d}^2 b}{\mathrm{d}s^2}(s) = K^2(-3s+1),$$

$$\frac{\mathrm{d}^2 c}{\mathrm{d}t^2}(s) = \left(\frac{\mathrm{d}s}{\mathrm{d}t}\right)^2 \frac{\mathrm{d}^2 c}{\mathrm{d}s^2}(s) = K^2(3s-2),$$

$$\frac{\mathrm{d}^2 d}{\mathrm{d}t^2}(s) = \left(\frac{\mathrm{d}s}{\mathrm{d}t}\right)^2 \frac{\mathrm{d}^2 d}{\mathrm{d}s^2}(s) = K^2(-s+1).$$

The second derivative of the parametrized curve itself is then straightforward, and is given by

$$\frac{\mathrm{d}^2\gamma}{\mathrm{d}t^2}(\tilde{t}) = K^2\left(p_{k+2}\frac{\mathrm{d}^2 a}{\mathrm{d}s^2}(s(\tilde{t})) + p_{k+1}\frac{\mathrm{d}^2 b}{\mathrm{d}s^2}(s(\tilde{t})) + p_k\frac{\mathrm{d}^2 c}{\mathrm{d}s^2}(s(\tilde{t})) + p_{k-1}\frac{\mathrm{d}^2 d}{\mathrm{d}s^2}(s(\tilde{t}))\right).$$

Evolving the control points

Recall that the evolution of the spline control points is driven by the equation

$$A\frac{\partial p^c}{\partial \tau} = \Phi^c, \tag{6.19}$$

where

$$\Phi^c = -\left[\int_0^1 \nabla I(C(p))(t)^c \psi^k(t)\,\mathrm{d}t\right]_{k=1}^{K},$$

and $c \in \{1,2\}$ denotes the spatial component. Since the curve is traced clockwise with increasing curve parameter, we have

$$|C'|\boldsymbol{n}(t) = \left[-\sum_{k=1}^{K} p_k^2 \frac{\mathrm{d}\psi^k}{\mathrm{d}t}(t), \sum_{k=1}^{K} p_k^1 \frac{\mathrm{d}\psi^k}{\mathrm{d}t}(t)\right],$$

and components of the gradient of the energy functional are given as

$$\begin{aligned}\nabla I(C(p))(t)^1 = &-\alpha\left[(\varphi_{int} - \varphi(C(p)(t)))^2 - (\varphi_{ext} - \varphi(C(p)(t)))^2\right]\sum_{k=1}^{K} p_k^2 \frac{\mathrm{d}\psi^k}{\mathrm{d}t}(t)\\ &-\beta\sqrt{\left(\sum_{k=1}^{K} p_k^1 \frac{\mathrm{d}\psi^k}{\mathrm{d}t}(t)\right)^2 + \left(\sum_{k=1}^{K} p_k^2 \frac{\mathrm{d}\psi^k}{\mathrm{d}t}(t)\right)^2}\,\sum_{k=1}^{K} p_k^1 \frac{\mathrm{d}^2\psi^k}{\mathrm{d}t^2}(t),\end{aligned}$$

$$\nabla I(C(p))(t)^2 = \alpha\left[(\varphi_{int} - \varphi(C(p)(t)))^2 - (\varphi_{ext} - \varphi(C(p)(t)))^2\right]\sum_{k=1}^{K} p_k^1 \frac{\mathrm{d}\psi^k}{\mathrm{dt}}(t)$$
$$-\beta\sqrt{\left(\sum_{k=1}^{K} p_k^1 \frac{\mathrm{d}\psi^k}{\mathrm{dt}}(t)\right)^2 + \left(\sum_{k=1}^{K} p_k^2 \frac{\mathrm{d}\psi^k}{\mathrm{dt}}(t)\right)^2}\sum_{k=1}^{K} p_k^2 \frac{\mathrm{d}^2\psi^k}{\mathrm{dt}^2}(t).$$

Numerical Integration Scheme

For solving the integral in (6.19) we divide the interval between each knot $[t_k, t_{k+1}]$ into T subintervals, each with length

$$\Delta t = \frac{t_{k+1} - t_k}{T} = \frac{1}{TK}. \tag{6.20}$$

On each subinterval we compute the contribution to the integral by using the trapezoidal rule for numerical integration. Each such subinterval $[t_k, t_{k+1}]$ is contained in the support of four basis functions, shown in figure 6.2. So the contribution $\Delta\Phi_k^c \in \mathbb{R}^K$, from the integration over the interval $[t_k, t_{k+1}]$, to Φ^c on the right hand side of (6.19) is

$$\Delta\Phi_k^c = \sum_{i=1}^{T} -\nabla I(C(p))(t_k + i\Delta t)^c (a(s_i^k)e_{k+2} + b(s_i^k)e_{k+1} + c(s_i^k)e_k + d(s_i^k)e_{k-1}), \tag{6.21}$$

where $s_i^k = s(t_k + i\Delta t)$ with the transformation $s(t)$ given in (6.15), and $e_j \in \mathbb{R}^K$ being the j-th unit basis vector. The right hand side of (6.19) is computed as

$$\Phi^c = \sum_{k=1}^{K} \Delta\Phi_k^c. \tag{6.22}$$

We do not need to give special treatment to the endpoints, since the curve is periodic.

Solving the normal equations

Since the curve is periodic, the matrix A in the system given in (6.19) will be a circulant matrix. The elements of the matrix are given as

$$A_{ij} = \int_0^1 \psi^i(t)\psi^j(t)\,dt.$$

The functions $\psi(t)$ are known basis functions given in (6.10), and so we can compute the elements of the matrix A. The calculations can be found in appendix B. We define the circulant vector $a \in \mathbb{R}^K$ as

$$a = [a_0, a_1, a_2, a_3, 0, \ldots, 0, a_3, a_2, a_1], \tag{6.23}$$

where

$$\begin{aligned} a_0 &= \frac{604}{35}\frac{1}{36K} & a_1 &= \frac{1191}{140}\frac{1}{36K} \\ a_2 &= \frac{6}{7}\frac{1}{36K} & a_3 &= \frac{1}{140}\frac{1}{36K}. \end{aligned} \tag{6.24}$$

The matrix $A \in \mathbb{R}^{K\times K}$ is the matrix with the vector a as the first row, and the remaining rows being cyclic permutations of this row. Now, since A is a circulant matrix, the linear system

$$A\frac{\partial p}{\partial \tau}^c = \Phi^c,$$

can be written as the convolution

$$a * \frac{\partial p}{\partial \tau}^c = \Phi^c, \tag{6.25}$$

with cyclically extended vectors $\frac{\partial p}{\partial \tau}^c$ and Φ^c. Using the circulant convolution theorem, we can write the system as

$$\mathcal{F}_K\left(a * \frac{\partial p}{\partial \tau}^c\right) = \mathcal{F}_K(a)\mathcal{F}_K\left(\frac{\partial p}{\partial \tau}^c\right) = \mathcal{F}_K(\Phi^c),$$

where $\mathcal{F}_K$ denotes the discrete Fourier transform (DFT). Thus, we can solve the system by taking the inverse Fourier transform:

$$\frac{\partial p}{\partial \tau}^c = \mathcal{F}_K^{-1}\left(\frac{\mathcal{F}_K(\Phi^c)}{\mathcal{F}_K(a)}\right). \tag{6.26}$$

Parameter scaling

In an attempt to obtain some comparable parameter values, we want to scale the parameters. This scaling can be obtained by doing a dimensional analysis of the two terms in the modified Mumford-Shah functional,

$$I(C) = \alpha \int\limits_{\text{int}(C)} (\overline{\varphi}_{int} - \varphi)^2\, dx + \alpha \int\limits_{\text{ext}(C)} (\overline{\varphi}_{ext} - \varphi)^2\, dx + \beta L(C).$$

We want to find characteristic sizes for the dimensions that occur in this energy functional. To this end, we define the scaling constants ϖ_α and ϖ_β and let

$$\alpha = \alpha_0 \varpi \alpha \qquad \text{and} \qquad \beta = \beta_0 \varpi_\beta, \tag{6.27}$$

where α_0 and β_0 are chosen parameter values.

We start by looking at the dimensions of the terms

$$(\overline{\varphi}_{int} - \varphi)^2 \qquad \text{and} \qquad (\overline{\varphi}_{ext} - \varphi)^2.$$

The value of the level set function is scaled by the initial scale s, and thus, these term will be scaled by s^2. We then integrate over $\text{int}(C)$ and $\text{ext}(C)$. The areas of these subsets of Ω are A_{int} and A_{ext} respectively. Since the sum of these two areas constitute the area of the whole image, their sizes will be in some sense comparable to the size of the image, which is $m \times n$. From the reasoning above, we have that

$$\varpi_\alpha = \frac{1}{s^2 mn}. \tag{6.28}$$

The second term of the energy functional is just the length of the curve C. From [9] we have that if $\Omega \subset \mathbb{R}^N$ then $L(C)^{N/(N-1)}$ is in some sense comparable with A_{int}, which is scaled by the size of the domain. Thus, letting ϖ_β be the scaling for β, we let

$$\varpi_\beta = \frac{1}{\sqrt{mn}}. \tag{6.29}$$

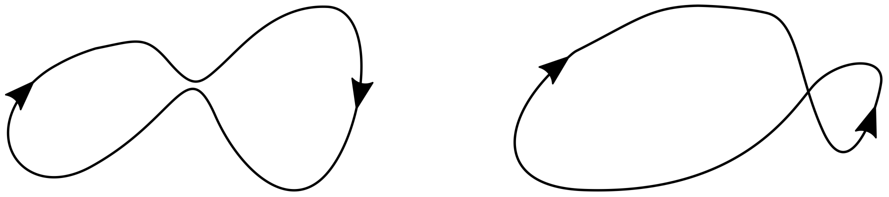
6.2.3 A simple splitting algorithm

In the evolution of the spline curve, we might encounter self-intersections. These self-intersections can be of two types, orientation-preserving self-intersections and orientation-reversing self-intersections, and they arise for different reasons. An example of an orientation-preserving self-intersection is the self-intersection formed when joining the close intersecting parts of the curve shown in figure 6.3a. It is seen that this self-intersection preserves the orientation of the curve, which means it also preserves the direction of the inward normal vector, which is important in the calculation of the gradient. An example of an orientation-reversing self-intersection is shown in figure 6.3b, and it is seen that this self-intersection reverses the orientation of the curve; the left part is traced clockwise and the right part is traced counter-clockwise. Usually, only orientation-preserving self-intersections can be caused by the actual shape of the objects in the image, and the occurrence of the orientation-reversing self-intersection is mainly due to numerical issues. Nevertheless, if either of these self-intersections occur, we will remove them by a naive splitting algorithm.

The splitting algorithm can be divided into two parts, namely the detection of a self-intersection and the splitting of the curve. The function *find_selfintersection* of the JordanCurve class (shown in appendix D.2) finds the values of the curve parameter where the curve intersects itself. These values are sent to the splitting algorithm, which can be found in the function *split_curve*. A pseudo-code version of the splitting procedure is shown in algorithm 1, which returns the number of added curves to the collection of curves.

Algorithm 1 Split curve

```
  $t_1, t_2$ are the values for self-intersection
  $K$ are the number of control points
**Require:** $t_2 \neq 0$
  $k_1 = \lfloor t_1 K \rfloor$
  $k_2 = \lfloor t_2 K \rfloor$
  $split = 0$
  **if** $|k_2 - k_1| > 1$ **then**
    **if** $|k_2 - k_1| > 5$ **then**
      Initialize curve $C_1$ with points between $p_{k_1}$ and $p_{k_2}$
    **end if**
    **if** $K - |k_2 - k_1| > 5$ **then**
      Initialize curve $C_2$ with the rest of the points
    **end if**
    **if** $C_1$ or $C_2$ has been initialized **then**
      Delete curve $C$ from collection of curves
      $split = split - 1$
      **if** $C_1$ is initialized **then**
        Add $C_1$ to collection of curves
        $split = split + 1$
      **end if**
      **if** $C_2$ is initialized **then**
        Add $C_2$ to collection of curves
        $split = split + 1$
      **end if**
    **end if**
  **end if**
  **return** $split$
```



(a) Orientation-preserving self-intersection.

(b) Orientation-reversing self-intersection.

Figure 6.3: Types of self-intersections.

6.2.4 A backtracking line search

As a measure to enhance the stability of the evolution, it can be useful to employ a line search to find a suitable step size. Thus, in some cases we will make use of the backtracking line search shown in algorithm 2. Assume we have a flattened vector of control points $p_k \in \mathbb{R}^{2K}$, and a search direction

$$\Delta p = \frac{\nabla I(C(p))}{|\nabla I(C(p))|}. \tag{6.30}$$

Then the backtracking line search will find a step length that yields a sufficient decrease in our functional I. The sufficient decrease condition is given by the Armijo condition. We refer to [23, p. 33–37] for more information on the Armijo condition and the backtracking line search.

Algorithm 2 Backtracking line search

```
Given a starting point $p_k \in \mathbb{R}^{2K}$, and a unit length vector $\Delta p \in \mathbb{R}^{2K}$
Choose $\lambda > 0$, $p, c \in (0,1), \rho \in (0,1)$
**while** $I(C(p + \lambda\delta p)) - I(C(p)) > \lambda c \nabla I(C(p))^T \Delta p$ **do**
    $\lambda = \rho\lambda$
**end while**
Set $p = p + \lambda\Delta p$
```

6.3 The Python implementation

All of the systems above is implemented in Python with extensive use of the *Numpy* and *Scipy* libraries. The optical flow system was built using sparse matrices from the *sparse* package of the *Scipy* library, and solved using a built-in GMRES solver from the submodule *linalg*.

The linear equation for the solution of the level set function was also solved by building sparse matrices, but was solved using the sparse linear solver *spsolve* from the *linalg* submodule. The numerical integration for the spline evolution was done by fixing a small increment and looping through the interval $[0,1]$. The linear equation was solved using the *solve_circulant*-function of the *linalg* submodule, which takes the vector a associated with the circulant matrix A and the right hand side vector Φ, and performs a division in Fourier space.

7 Numerical results and discussion

This section presents the most important results, in order to show some of the features of the algorithm. Section 7.1 compares the three different methods of regularization presented in chapter 3, namely the isotropic smoothing of Horn and Schunck, the image driven regularization and the flow driven regularization. Section 7.2 presents the results for the segmentation algorithm using an optical flow field computed using a flow driven regularization.

The image sequences used in these experiments is the Hamburg taxi sequence, for which two consecutive frames are shown in figures 7.2a and 7.2b. The sequence shows a white taxi turning right onto the street coming down from the top of the image, a black right-moving car on the left side of the image, and a left-moving van on the right side, partly occluded by branches from a tree. In addition to the cars moving, there is a pedestrian walking on the sidewalk in the upper left part of the image.

7.1 Optical flow results

This section presents some results for the optical flow estimation, ordered by the methods of smoothness regularization presented in section 3.3. We will measure the quality of the optical flow field by features that are important in the segmentation process. Due to the property

$$g(r,q) \to 0 \text{ as } r \to \infty \text{ or } q \to \infty,$$

of the edge detector $g(\nabla u(C), \nabla v(C))$ presented in section 4.1.2, we seek a flow field with sharp edges and high gradients. Also, since the active contour will stop at these sharp edges, we want the amount of oversegmentation in the optical flow to be as little as possible.

To display the computed flow field each flow vector was mapped to the colorwheel shown in figure 7.1, where the hue represents the direction and the value represents the length of the vector. This mapping was computed using a flow code provided by Sun [31].

7.1.1 Results for the isotropic regularization

As noted in section 3.3.1, the isotropic smoothness term of Horn and Schunck performs a homogeneous diffusion. Figure 7.3a shows the result using this regularization method with quadratic data term penalization. The method successfully reveals the motion of the three moving cars in the image, and the directions correspond to the directions of motion in the first considered frame of the Hamburg taxi sequence seen in figure 7.2a. However, the flow boundaries are smoothed out, especially for the right moving car. As stated

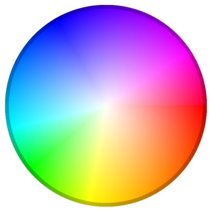
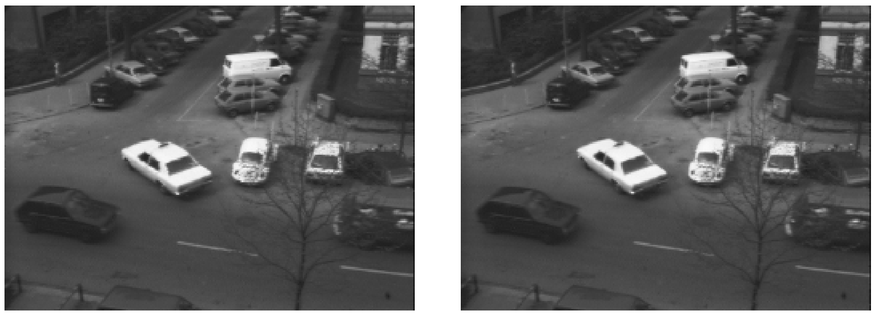


Figure 7.1: The colorwheel.



(a) First image.

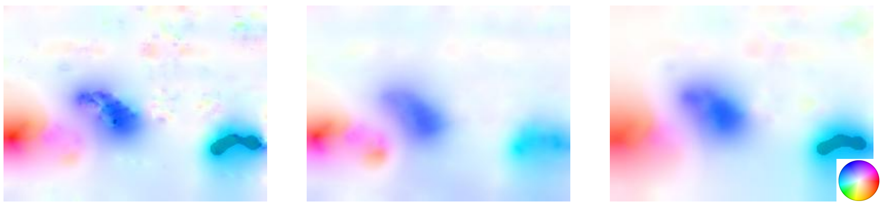
(b) Second image.

Figure 7.2: Frame 10 and 11 of the Hamburg Taxi sequence.

above, this is to be expected as the Horn and Schunck smoothness term performs a homogeneous diffusion that smooths the flow field uniformly in all directions. In addition to the moving cars, the method also detects a lot of image structure and background detail from the street, possibly due a slight movement of the camera. The movement of the camera will result in many of the image structures in the original frame being picked up as movement.

The issue of noise detection and visible background details can to some extent be enhanced by normalizing the data term. This is seen in figure 7.3b, which shows the resulting flow field using the Horn and Schunck smoothness term with a normalized quadratic data term. The normalization is seen to remove a lot of the inner image structures of the moving objects along with the image structures of the surroundings, though there is still some visible image structure in the right-moving car. The normalization also seems to separate the flow boundaries of the taxi and the right-moving car on the left, which are very smudged in the result for the original Horn and Schunck method.

In figure 7.3c the gradient constancy assumption (GCA) is included in the data term, which is normalized and quadratically penalized. To prevent oversegmentation and to reduce the effect of noise, the spatial Gaussian used in the experiment was $\sigma = 1.5$. The Gaussian presmoothing blurs the image, and noisy pixels are smoothed out over a larger area. This also serves as a remedy for not detecting unwanted image structures. However, the presmoothing has the unwanted effect that the objects are more smudged, and the right moving car and the taxi are now almost overlapping due to the strong smoothing. On the other hand, the flow field is not as faded as in the case of the normalized data term shown in Figure 7.3b, but there is a considerable amount of detected image structures from the background if compared with the flow field in Figure 7.3b. Also, the high value for the standard deviation in the Gaussian smoothing results in objects appearing to be larger than they actually are, because the movement is smoothed out over the neighbouring pixels. This is seen from comparing the size of the right-moving car with figure 7.3b.


(a) Quadratic penalization, $\beta = 0.02$, $\sigma = 0.5$.

(b) Normalized quadratic penalization, $\zeta = 5.0, \beta = 0.15, \sigma = 0.5$.

(c) Normalized quadratic penalization with GCA, $\beta = 0.05$, $\sigma = 1.5, \zeta = 5.0$, $\gamma_{OF} = 20$.

Figure 7.3: Directional mapping of the flow field obtained using the Horn and Shunck smoothness term with different data terms. The directions correspond to the motion of the objects in the image. Flow boundaries are smoothed out as the Horn and Schunck method performs a homogenous diffusion. The normalized data term 7.3b reduces the degree of internal image structure for moving objects along with image structures from the surroundings. The flow field using the data term with the gradient constancy assumption 7.3c requires a high value of the spatial Gaussian to avoid oversegmentation and noise. This results in nonlocalized flow boundaries but less internal structure than seen in 7.3a.

7.1.2 Results for the image driven method

The Horn and Schunck method clearly suffers from smoothing out important flow boundaries, which can make the task of segmentation difficult. Figure 7.4a shows the result for the image-driven method of Nagel and Enkelmann with quadratic data term penalization, as presented in section 3.3.2. The anisotropic regularization produces flow boundaries that are more localized than for the homogeneous regularizer, but the image structures of the taxi are slightly stronger in this case. The method also suffers from oversegmentation and captures a lot of the unwanted image structure from the background. Nagel and Enkelmann used a local smoothness term with no weighting of neighboring pixels. With increasing standard deviations ρ, the structure

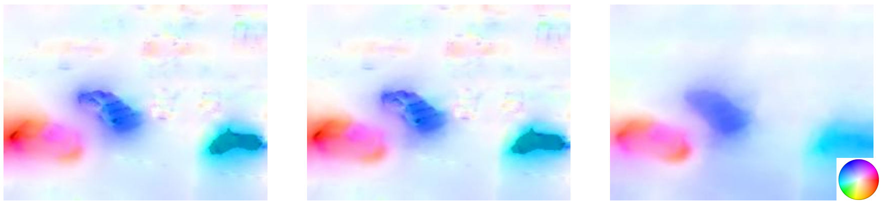

(a) $\beta = 0.015$, $\rho = 0.0$ (b) $\beta = 0.015$, $\rho = 1.0$ (c) $\beta = 0.06$, $\zeta = 3.0$, $\rho = 1.0$

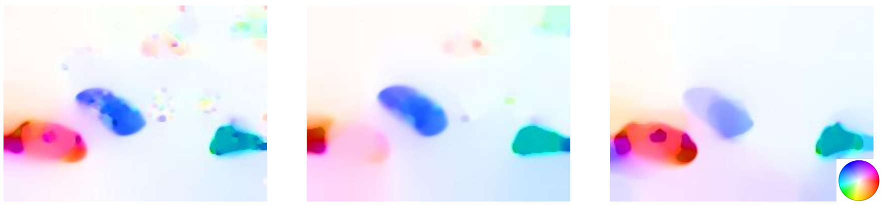
Figure 7.4: Image driven smoothness term with quadratically penalized data terms. 7.4a shows the result of using the eigenvectors of a structure matrix without integrating neighborhood information as smoothing directions. 7.4b shows the result when integrating over a neighborhood with standard deviation $\rho = 1.0$. 7.4c shows the flow field when using an image driven smoothness term with a normalized data term. This removes a lot of the background details.

matrix in equation (3.26) integrates more of the neighborhood around a given pixel. Figure 7.4b shows the estimated flow field found by using $\rho = 1.0$. This has the benefit of reducing the effect of noise, but at the same time gives nearby pixels with large gradients a higher weighting. Furthermore, a higher value for ρ results in smoother flow discontinuities giving objects more rounded corners. As noted above, the method suffers from oversegmentation, but normalization can offer some improvement in this direction as seen from figure 7.4c. The image structures of the background are almost invisible and there is no visible image structure inside the taxi.

7.1.3 Results for the flow driven method

The flow-driven regularization of section 3.3.3 aims to reduce the smoothing at flow edges, and the smoothing strength is determined by the TV-functional given in (3.30), which essentially models the L^1-norm of the flow gradients for small values of ϵ_{OF}. Figure 7.5a shows the result for the flow driven method with quadratic data term penalization. The method reduces the amount of smoothing close to flow boundaries so that these are more localized than for the image driven method. Moreover, there is less internal structure in the taxi, which has a more homogeneous flow pattern than for the result seen in figure 7.4a. This is expected as the smoothing of the flow driven method is not so much dependent on the structure information of the image. In the right-moving car (the red part of figure 7.4a) the flow pattern is nonhomogeneous, with darker shades of red in some parts of the car. This might cause the level set function to describe this internal stucture instead of the whole object.

Increasing the standard deviation in the presmoothing process (figure 7.5b) leads to a more uniform flow pattern in the taxi, but not the right-



(a) Quadratic data term penalization, $\beta = 0.2$, $\sigma = 1.5$

(b) Quadratic data term penalization, $\beta = 0.2$, $\sigma = 3.0$

(c) Normalized quadratic data term penalization, $\beta = 0.6$, $\sigma = 3.0$, $\zeta = 1.0$

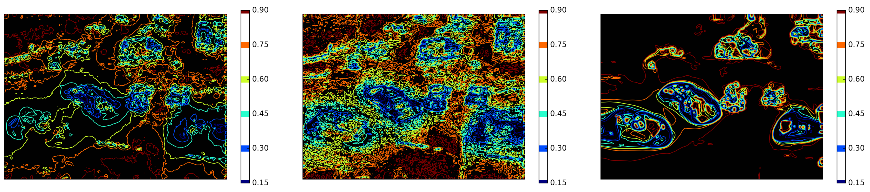
Figure 7.5: Flow driven smoothness term with different data terms and penalizations of data terms. 7.5a shows the flow driven smoothness term with a quadratically penalized data term. Some background noise is present and the flow has some discontinuities inside the moving objects. 7.5b shows the situation in 7.5a but with an increased standard deviation for the presmoothing process. The influence of the background movement is reduced but the right-moving car is faded. Boundary effects caused by the TV-regularization are present. The flow is more uniform inside the taxi and the left-moving van. 7.5c shows situation with a normalized data term and a higher regularization parameter for the smoothness term. No background details are present, but internal flow discontinuities are introduced.

moving car. The method is also seen to suffer from boundary effects (the flow pattern of the right moving car is enclosed over the boundary, when the whole car is actually in the image). This is a result of minimizing the (approximation of) the L^1-norm of the gradient of the flow components.

The result for the normalized data term with quadratic penalization is shown in figure 7.5c. The normalization removes a lot of the noise from the background, but some additional discontinuities arise in the objects.

7.1.4 Comparing the regularization methods

As previously noted, to aid the segmentation process, we are looking for flow fields that will cause the level sets of the level set function φ to describe the outer structure of the objects. The evolution of the level sets will stop at edges detected by the edge detector function g, and thus it makes sense to compare how the edge detector performs in detecting edges for each of the regularization methods in the above sections. Figure 7.6 shows the contour lines of the edge detector function, using $p = 1$ and $\eta = 100$, for the flow fields in figures 7.3a, 7.4a and 7.5a. The edges of the isotropic regularized flow shown in figure 7.6a are seen to give a poor description of the objects in the image. Only the shape of the taxi can be recognised in the contour lines, which are partially merged with the contour lines from the right moving car. There is also a lot of noise present, which will lead to a slower evolution of the level set, and may also cause these objects to be segmented instead of the moving cars.



(a) Isotropic regularization. (b) Image driven regularization. (c) Flow driven regularization.

Figure 7.6: The contour lines of the edge detector g using $p = 1$ and $\eta = 35.0$ for flows with different regularization. The contour lines of figure 7.6a are computed from the isotropic regularized flow field shown in figure 7.3a. The flow edges of different objects are not clearly separated, and there is a lot of noise present. Figure 7.6b shows the edges deteted from the flow in figure 7.4a, using image driven regularization. Edges are more separated, but there is a great deal of oversegmentation from objects in the background. Figure 7.6c shows the edges detected from the flow driven regularized flow field shown in figure 7.5a. The edges of different objects are more separated, and they describe the outer structure of the objects nicely. There are still some artifacts present from the background, but fewer than for the two other flow fields.

Figure 7.6b shows the flow edge detection for the flow field with image driven regularization. There is clearly a great deal of oversegmentation, even more than for the isotropic smoothing. The flow edges consists of a number of smaller disconnected parts, and does not give a smooth representation.

The edges detected from the flow using a flow driven regularization are seen in figure 7.6c. The flow boundaries of different objects are more separated than for the two other flow fields. Furthermore, the edges are smoother and more localized, and there is not as much noise from the background. Some objects from the background are visible, which might cause a slower evolution for the level set. Still, the amount of oversegmentation is less than for the two other flow fields, and the edges give a good description of the outer structure of the objects.

As a result of the improved performance with respect to the measure of quality mentioned in the beginning of section 7.1, the flow driven regularization will be used to compute flow fields for the segmentation process. The downside of this method is the increased computation time as compared to the two previous methods due to the lagged diffusivity iteration. The iteration scheme converges fairly quickly to a result that does not admit any visible differences in the flow field. However, the convergence rate slows down after a few iterations. Thus, computing a flow field with high accuracy takes considerably more time. This will not be needed in practice, since there is no change in the visual result of the flow estimation after a few iterations.

7.2 Segmentation results for the Hamburg taxi sequence

In the previous section the different types of optical flow regularization was compared. It was seen that the flow driven regularization gives the best result with regards to obtaining a successful segmentation. This section aims at showing the performance of the segmentation algorithm, given a flow field computed using this regularization method. To demonstrate different situations that might occur, and how we will handle them, the Hamburg taxi sequence will be used, for which two consecutive images are shown in figure 7.2. The size of each frame in this image sequence is 190×256.

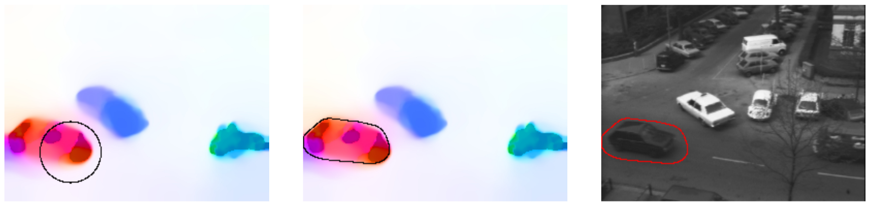
The following subsections will look at some interesting segmentation cases. The level set function used in the experiment is the signed distance function of a circle with varying radius and center, and the initial spline curve will also be a circle, but not necessarily with the same radius and center as the initial level set function.

7.2.1 Single curve evolution

Figure 7.7 shows the segmentation of the black car. The initial spline curve, a circle with center approximately on the hood of the black car, is shown in figure 7.7a and is formed by $K = 20$ control points. The parameters used in the spline evolution are $\alpha = 5.0 \cdot 10^{-4}/(0.001^2 \times 190 \times 256)$, $\beta = 5.0 \cdot 10^{-5}/\sqrt{190 \times 256}$ and $\delta' = 5.0$. The initial level set function used is the signed distance function of this initial spline curve, using a scaling $s = 0.001$, and the evolution of this level set function uses parameter values $\delta = 50$, $\gamma = 0.5$, $\eta = 10$ and $\epsilon = 0.0001$. The result shown in figure 7.7b was obtained after 11 iterations of a 1:50 combined evolution. Figure 7.7c shows the resulting segmented region in the original shown in figure 7.2a. It is seen that the curve successfully describes the edge of the flow field, but due to a non-localized flow boundary, the region is seen to enclose a relatively large area around the taxi which is not moving.

7.2.2 Different initial data and the effect of the balloon force

The previous experiment used the same curve for initializing the level set function and the spline curve. The following experiment uses non-coinciding curves for the initial level set and the initial spline curve. Figure 7.8a shows selected contours of the initial level set function and 7.8b shows the initial spline curve. The center of the level set contours is seen to lie in between the black car and the taxi, and the spline curve is centered inside the taxi. Figure 7.8c shows the selected contours after an initialization process of 30 level set iterations, and figure 7.8d shows the resulting curve after 9 iterations of a 1:30 combined evolution. With the right choice of parameters for the spline evolution, one can obtain a stable evolution of the spline curve when the initial contours are no longer describing a signed distance function, and the initial spline curve is relatively far from coinciding with



(a) Flow field with initial curve.

(b) Flow field with curve after segmentation process.

(c) Original image with curve after segmentation.

Figure 7.7: Segmenting the black car of the Hamburg taxi sequence. Figure 7.7a shows the flow field computed using the flow driven regularization along with the initial curve. Figure 7.7b shows the flow field and the curve after the segmentation process. Figure 7.7c shows the segmented region in the original image.

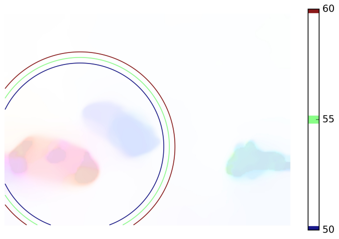
these contours. Now, the result of the segmentation shown in figure 7.8d is not a particularly good one. The contours of the level set function is describing the black car and the taxi as one connected component, and as a consequence the spline curve will try to do the same.

The level set evolution of figure 7.8c used $\gamma = 0.1$ as the value for the balloon force parameter. By using a higher balloon force parameter, one can drive the contours of the level set function past these local minima. This is shown in figure 7.8e, which shows the same contours after the same amount of initial level set evolutions using $\gamma = 0.2$. It it seen that the level curves have separated into two disconnected regions describing the boundary of the flow. The initial spline 7.8b is closer to segmenting the taxi than the black car. The result shown in figure 7.8f is obtained in one combined evolution of one level set evolution and 30 spline evolutions, using spline parameters $\alpha = 5.0 \cdot 10^{-3}/(1.0^2 \times 190 \times 256)$, $\beta = 2.0 \cdot 10^{-5}/\sqrt{190 \times 256}$ and $\delta' = 10$. This result can also be controlled by the value of β. Higher values of β will give more penalization to the length of the curve, making regions of high curvature unattractive.

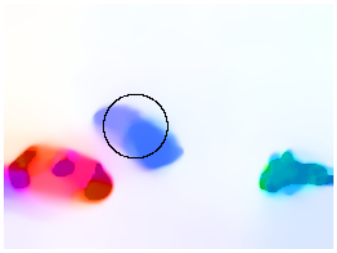
7.2.3 Detecting self-intersections and curve splitting

Section 6.2.3 presented the algorithm for splitting a curve into two parts if a self-intersection is encountered. Figure 7.9a shows a spline curve after 6 iterations of a 1:30 combined evolution scheme. The level set function is evolving using parameters $\delta = 100$, $\epsilon = 0.001$, $\gamma = 0.5$ and $\eta = 100$. The spline curve consists of $K = 30$ control points, and is evolved with parameters $\alpha = 5.0 \cdot 10^{-3}/(1.0^2 \times 190 \times 256)$, $\beta = 2.0 \cdot 10^{-5}/\sqrt{190 \times 256}$ and $\delta' = 0.5$. It is seen from figure 7.9a that the curve is close to self-intersecting, forming a orientation-preserving self-intersection.

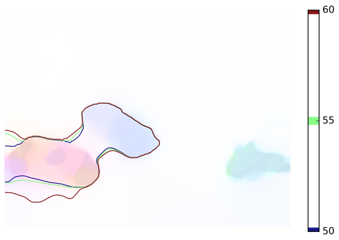
Figure 7.9b shows the situation after doing one more spline evolution. The self-intersection was the detected by the detection algorithm, and the splitting algorithm performed a splitting, leaving two closed curves. These



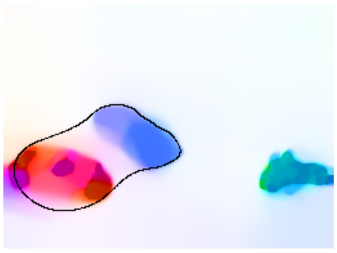
(a) Contours of initial level set.



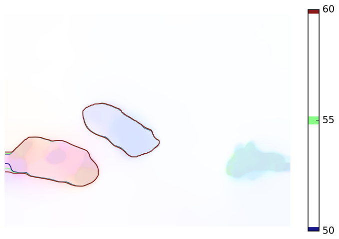
(b) Initial spline curve.



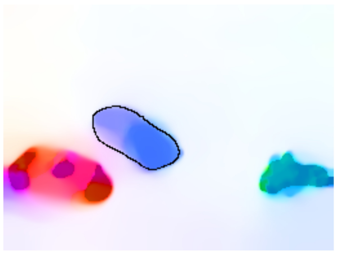
(c) Contours after 30 iterations with $\gamma = 0.1$.



(d) Curve after 9 combined evolutions of level set and curve.

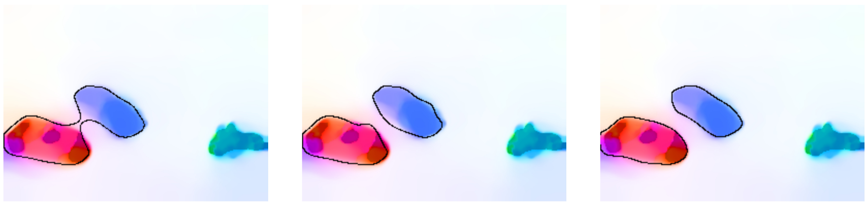


(e) Contours after 30 iterations with $\gamma = 0.2$.



(f) Curve after 1 combined evolutions of level set and curve.

Figure 7.8: The effect of the balloon force. Figure 7.8a shows some selected contours of the signed distance function for a circle centered between the black car and the taxi. Figure 7.8b shows the initial spline curve as a circle with center inside the taxi. Figure 7.8c shows the selected contours of the level set function shown in figure 7.8a after 30 iterations using $\gamma = 0.1$ and step length $\delta = 30$. The contours are seen to enclose both the black car and the taxi as connected curves. Figure 7.8d shows the result of the following segmentation, using 9 iterations of a 1:30 combined evolution, after the initial 30 iterations of the level set function. The spline curve is seen to enclose both the taxi and the black car, describing them as one connected component. Figure 7.8e shows the resulting level set curve when running 30 initial iterations with $\gamma = 0.2$. The connected component is seen to break off, and form two separate regions. Figure 7.8f shows the result of one iteration of a 1:30 combined evolution after the initial 30 level set iterations. The curve initialized inside the taxi is seen to successfully describe the computed flow field of the taxi. The parameters used for the spline iterations is $\alpha = 5.0 \cdot 10^{-3}/(1.0^2 \times 190 \times 256)$, $\beta = 2.0 \cdot 10^{-5}/\sqrt{190 \times 256}$ and $\delta' = 10$, and the edge detector used $\eta = 100$.



(a) Curve before splitting. (b) Curves after splitting. (c) Separately evolved curves after splitting.

Figure 7.9: The results of the simple splitting algorithm. Figure 7.9a shows the spline curve enclosing both the taxi and the black car, with a level set topology leading to a self intersection in the curve. Figure 7.9b shows the spline curves after the detection of a self-intersection and the successive curve splitting, resulting in two closed curves. Figure 7.9c shows the result after independently evolving the two spline curves.

curves are added to the collection of curves, and their further evolutions are independent. Figure 7.9c shows the result after doing one combined evolution after the splitting. The spline curves are seen to lock on to the level sets shown in figure 7.8e

8 Segmentation results using real-world data

In the previous chapter we saw the results of the segmentation for the Hamburg taxi sequence. This sequence is a particular nice image sequence for estimating optical flow. The size of the images is relatively small which encourages the use of the direct solvers for sparse systems found in the Python module scipy. The brightness is also fairly homogeneous in the image scene, which is an important assumption for estimating optical flow. In this chapter we will look at segmentation results using two real-world data sets. The image sets were provided by the *Norwegian Defence Research Establishment*, or *Forsvarets Forskningsinsitutt* (FFI), which is the prime institution responsible for defence-related research in Norway. We will denote the two image sets by *Walking man in forest* (WMF) and *Drone flying over man* (DFM).

Figure 8.1 shows a cropped selection of the DFM image sequence. The DFM image sequence is taken from a drone (UAV) flying over a field, following the movement of a walking person. The movement of the person relative to the field can be seen by looking at the position of the person relative to the diagonal lines, which are a part of the field. The sun is shining in from the right in the image, and thus the person is casting a large shadow compared to the size of the person. The lighting conditions of the image scene are fairly homogeneous, and thus one may expect the brightness constancy assumption to hold. The image size for the DFM sequence is 600×800.

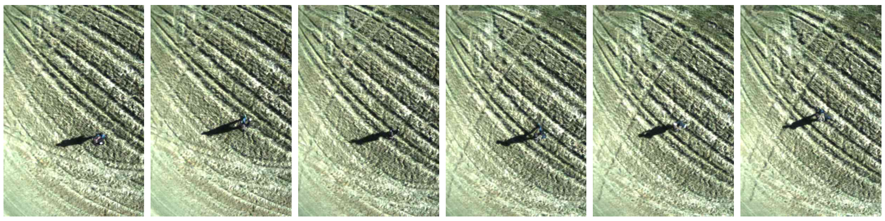


Figure 8.1: Cropped selection of images from the DFM sequence.

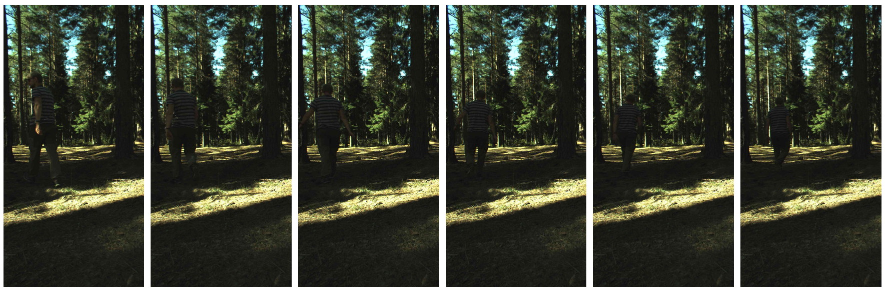


Figure 8.2: Cropped selection of images from the WMFR sequence.

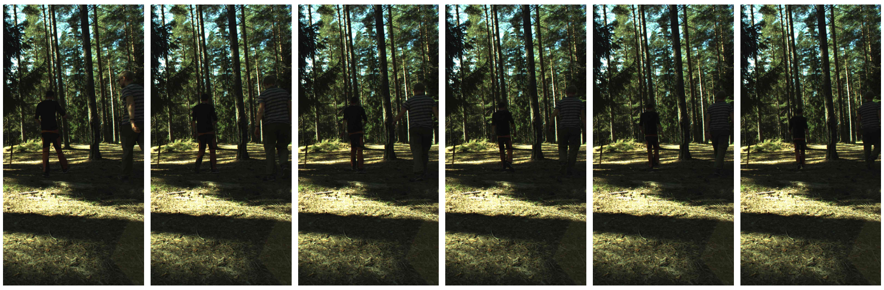


Figure 8.3: Cropped selection of images from the WMFL sequence.

The WMF sequence consists of two sets of images, taken from two different angles. We will denote the image sequence taken by the left camera as WMFL, and the sequence taken by the right camera as WMFR. A cropped selection of the WMFR image sequence is shown in figure 8.2. It shows a man walking into a shaded area of a forest, moving away from the camera. The camera is stationary, and there is no movement in the image sequence except from the walking man. The illumination in the image is clearly changing as the person moves into the shadow, and one can expect a considerable violation of the brightness constancy assumption in the first image of figure 8.2. However, once in the shadow, there is relatively little change in brightness for the internal pixels of the walking man, and thus one could expect this part of the sequence to yield the best optical flow results. A cropped selection of the WMFL image sequence, corresponding to the selection from the WMFR sequence, is shown in figure 8.3. From this angle one can see another person moving in the image in addition to the person from WMFR. The person from the WMFR image sequence is seen to enter the image from the right boundary. The original image size for each frame in these sequences is 2048×2048, which is relatively large for the computation of optical flow. In the pre-processing step for the optical flow computation the images were resized to 30% of the original size and then cropped to the size 614×307. The resizing was done using a billinear interpolation of pixels.

8.1 Segmenting concave regions

This section presents segmentation results using two consecutive images in the WMFR image sequence. These two images are shown in figures 8.4a and 8.4b. Figure 8.4c shows the resulting flow field computed using a flow driven regularization as presented in section 3.3.3. The flow field is seen to describe the features of the person rather well, successfully depicting the head, the legs and the right arm. The flow was computed using a flow driven regularization with parameters $\sigma = 1.0$, $\gamma_{OF} = 2.0$, $\zeta = 0.5$, $\epsilon_{OF} = 0.01$ and $\xi = 0.7$. The smoothness parameter ξ was chosen relatively high to allow more flow discontinuities. This is seen from the localized flow boundary

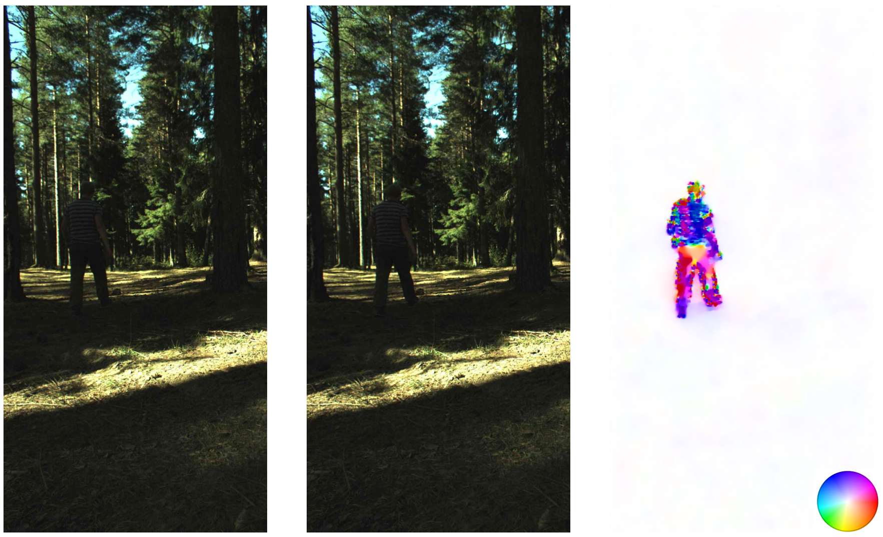
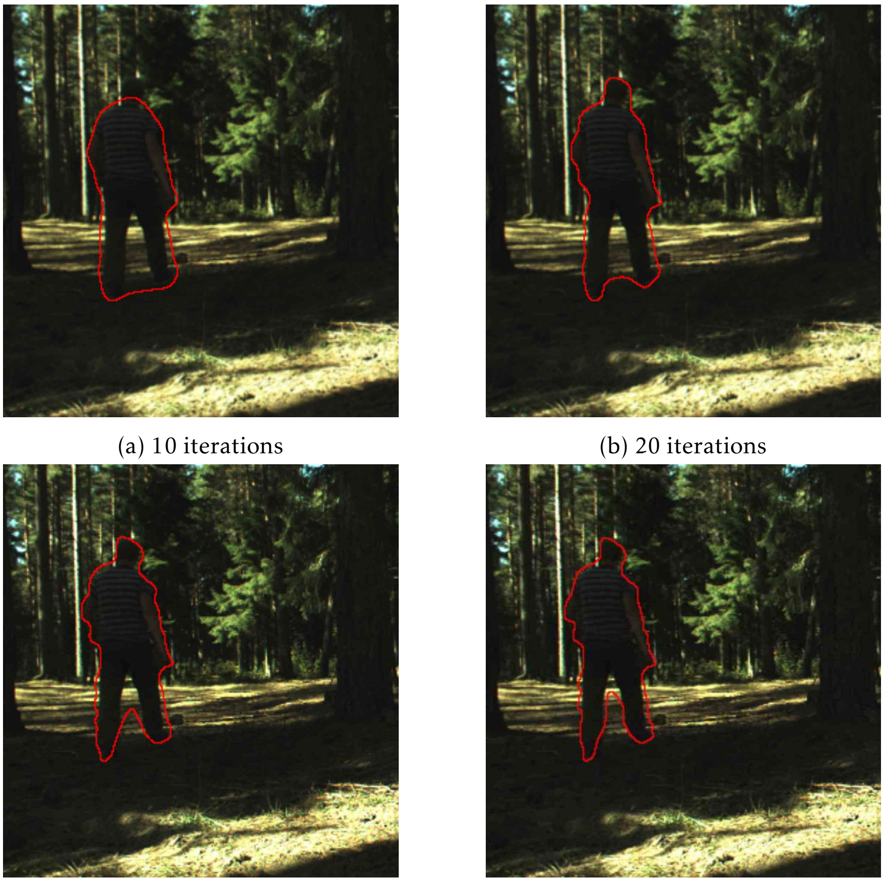

(a) Image 1 (b) Image 2 (c) Flow field

Figure 8.4: Figures 8.4a and 8.4b are two consecutive images in a sequence of a person walking in a forest, taken by a still camera. The illumination is seen to change through the image scene. The optical flow field computed from the two images, using a flow driven smoothness term, is shown in figure 8.4c.

around the outer structure of the person, and the discontinuous flow inside the person. In an attempt to compensate for illumination changes, the GCA parameter γ_{OF} is chosen to be nonzero.

As the flow field has a lot of discontinuities inside the walking man, there is a lot of noise the edge detector can pick up on. To make the level set function more robust to this noise, we choose a smaller scaling for the initial level set function ($s = 0.0001$). We can also make sure the sensitivity parameter for the edge detector is not too high by plotting the edge detector for different values of η. It was found that $\eta = 100$ gave a good compromise between detecting outer structure and not detecting noise. Moreover, we need to avoid unbounded gradients of the level set function for the edges inside the person. This is done by choosing a high value of the balloon force ($\gamma = 0.5$) in combination with a lower value for the step size ($\delta = 20$). The added velocity of the level set evolution coming from the balloon force will help to drive the level set flow past the internal flow boundaries. Using this along with a low value for δ avoids the need for reinitializing the level set function while still achieving convergence for the spline curve.

Furthermore, to avoid numerical issues causing self-intersecting curves of the orientation-reversing type, we let the spline curve evolve with a more stringent length penalization, using $\alpha = 1.0 \cdot 10^{-7}/(0.0001^2 \times 614 \times 307)$ and $\beta = 5.0 \cdot 10^{-5}/\sqrt{614 \times 307}$. To further assist the curve in maintaining a robust



(a) 10 iterations (b) 20 iterations

(c) 50 iterations (d) 100 iterations

Figure 8.5: Segmentation of the flow field shown in figure 8.4c. The parameters used in the level set evolution is $\gamma = 0.5$, $\eta = 100$, $\epsilon = 0.01$, $\delta = 20$ with φ^0 being the signed distance function with $s = 0.0001$. The spline consists of $K = 50$ points and the parameters used is $\alpha = 1.0 \cdot 10^{-7}/(0.0001^2 \times 614 \times 307)$, $\beta = 5.0 \cdot 10^{-5}/\sqrt{614 \times 307}$. The experiment was run iteratively using a 1:15 combined evolution scheme. Figures 8.5a, 8.5b, 8.5c and 8.5d shows the spline curve after 10, 20, 50 and 100 iterations respectively. It is seen that the convex region of body is segmented relatively fast, but the curve requires considerably more iterations to successfully describe the concave region between the legs.

evolution, we use a backtracking line search for the step size, using $\lambda = 5$ as the initial step size.

The curve is initialized inside the person, using $K = 50$ control points. A 1:15 combined evolution scheme is used to evolve the level set and the spline curve. Figure 8.5 shows the evolution of the spline curve. Figure 8.5a shows the spline curve after 10 iterations of the combined evolution.

It is seen that the spline curve describes the outline of the body, but not the head or the legs. Figure 8.5b shows the spline curve after 20 combined evolutions. The curve is describing the head, and the shape of the body is more detailed. The curve still has problems describing the legs of the person, but it is seen to have started deforming upwards. Figure 8.5c shows the spline curve after 50 combined iterations. The head and shoulders are well described, the shape of the body is more detailed than in figure 8.5b. The legs are more clearly described in this figure, but there is still a large area beneath the pelvic region not described by the curve. Figure 8.5d shows the curve after 100 iterations. The curve is very close to accurately describing the most important features of the shape of the person. The legs, the right arm, the head, the shoulders and even the elbow of the left arm arm can be recognised in the shape of the spline curve.

From the above analysis it is clear that convex regions are described relatively fast, but concave regions, like the region between the legs, take longer time to segment. This is due to the strong length penalization of the Mumford-Shah functional, which will try to avoid regions of high curvature in the curve. Moreover, in the initial steps of the algorithm control points are spreading out along the length of the curve. After these eight iterations, the points are forced to move along the curve to be able to deform into the concave region seen in figure 8.5d. There are several ways of approaching this issue. One suggestion might be to reduce the value of β, allowing the curve to more easily deform into concave regions. In some cases this might be sufficient. However, in this case we need the high value of β to avoid the numerical issues discussed. A better alternative might be to implement a control point insertion algorithm, adding a control point along the concave segment of the curve.

8.1.1 Initialization process for the level set function

In some cases the spline curve will evolve very slowly in the initial iterations of the combined evolutions. It can be advantageous to perform some initial level set iterations before starting the combined evolution of the spline curve and the level set function. This process is referred to as the initialization process. Figure 8.6 demonstrates the result of this initialization process on a flow field computed using the Horn and Schunck method. The level set function is initialized as the signed distance function of the curve seen in Figure 8.6a with scaling $s = 0.001$. The level set function is evolved using parameters $\delta = 50$, $\epsilon = 0.001$, $\gamma = 0.9$ and $\eta = 100$. Before the combined evolution, the level set is evolved using five iterations, without evolving the spline curve. After this initialization process, a 1:50 combined evolution scheme is initialized. Figure 8.6b shows the spline curve after two combined iterations. The curve is shown in black, drawn onto the color coded flow field. Figure 8.6c shows the curve drawn onto the original image. It is seen that the curve gives a good segmentation of the flow field after two combined iterations. The spline curve surrounds the objects, and has a small concavity between the feet of the person.

From this experiment we can also compare the segmentation results for the Horn and Schunck (HS) method with the flow driven (FD) regularization. From the flow field itself we can see that there is more noise present in the HS flow field compared to the FD flow field. There are also some image structures visible in the HS flow, like the two trees close to the left and right boundaries. This can be a challenge for the segmentation of objects, as detection of image structures and noise can lead to the edge detector stopping the evolution at these boundaries. However, for this particular flow field, initial spline curve and parameter choice, this is not a problem. We can also note the stronger flow boundaries for the FD flow field, due to the decreased smoothing in these areas. The isotropic smoothing will lead to a more smudged flow field, which can make small concave regions difficult to segment. Nevertheless, the HS method is appealing due to its simplicity and computation time. This will be further discussed in section 8.5.

Figure 8.7 shows the result when doubling δ and δ' from the previous experiment. Two initial level set iterations are run before the combined evolution. The result shown is obtained after only one combined evolution. The result is seen to give a slightly less accurate description of the flow boundary than for the result seen in figure 8.6, but the curve is still able to segment the head and parts of the legs, using approximately half the computation time from the previous experiment.

8.2 Segmenting multiple objects

Figures 8.8a and 8.8b show two images from the WMFL image sequence, and figure 8.8c shows the computed flow field using the flow driven regularization with parameters $\sigma = 1.0$, $\xi = 0.7$, $\gamma_{OF} = 1.0$, $\zeta = 0.1$ and $\epsilon_{OF} = 0.01$. A successful segmentation of these objects would result in two separated regions describing the shapes of two persons. The following segmentation experiment is run with level set parameters $\delta = 50$, $\gamma = 0.9$, $\eta = 150$ and $\epsilon = 0.001$. The level set function is initialized as the signed distance function of the circle shown in figure 8.9a with scaling $s = 0.001$. As an initialization procedure the level set was evolved using four iterations without performing any spline iterations. After this, the spline curve is initialized with $K = 100$ control points as the circle shown in figure 8.9a, and evolved using a 1:50 combined evolution scheme. The parameters for the spline curve evolution is $\delta' = 0.25$, $\alpha = 1 \cdot 10^{-5}/(0.001^2 \times 614 \times 307)$ and $\beta = 2.0 \cdot 10^{-5}/\sqrt{614 \times 307}$. The ratio α_0/β_0 is chosen relatively high as to allow for the curve to deform into concave regions more easily. Figure 8.9b shows the spline curve drawn on the color coded flow field after seven combined iterations. The curve is seen to enclose both objects, and the balloon force is forcing the curve into a concave segmentation between the objects. After eight combined iterations the curve has self-intersected, and the splitting algorithm has performed

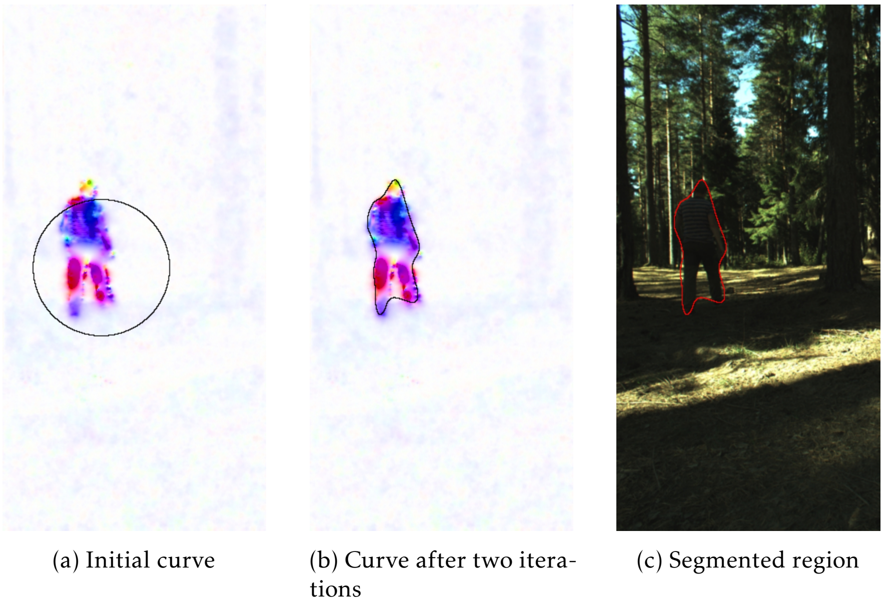

(a) Initial curve (b) Curve after two iterations (c) Segmented region

Figure 8.6: The result of segmenting a flow field using the Horn and Schunck smoothing term, doing five level set iterations before the combined evolution. The level set function is evolved using parameters $\delta = 50$, $\epsilon = 0.001$, $\gamma = 0.9$ and $\eta = 100$, and the spline curve is evolved using $\alpha = 1.0 \cdot 10^{-4}/(0.001^2 \times 614 \times 307)$, $\beta = 2.0 \cdot 10^{-5}/\sqrt{614 \times 307}$ and $\delta' = 1.0$. Figure 8.6a shows the initial spline curve formed by $K = 30$ control points. Figure 8.6b shows the spline curve after two 1:50 combined evolutions. The splitting algorithm has reduced the number of points from 30 to 21. The result is seen to surround the convex areas of the person, and forms a slightly concave regions between the legs. Figure 8.6c shows the segmented region in the original image.

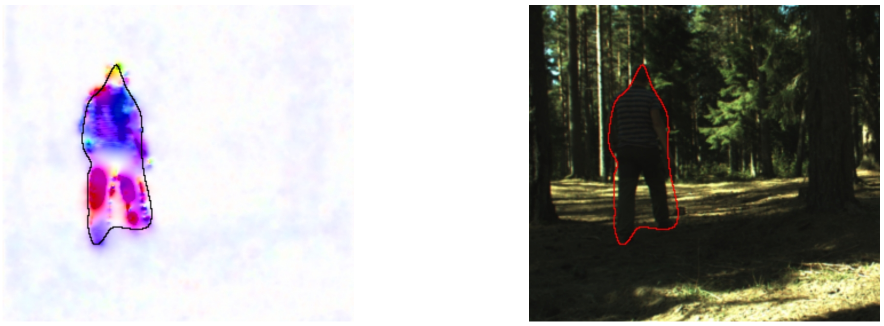
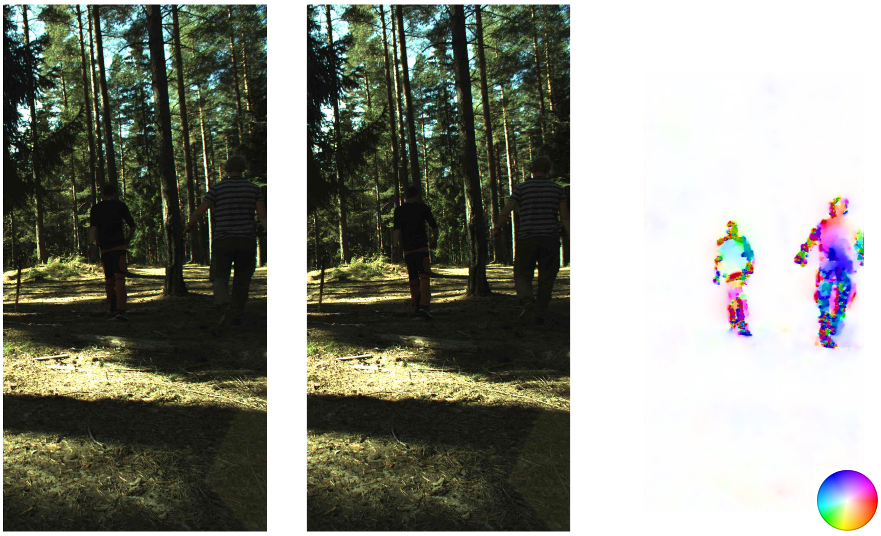


Figure 8.7: The segmentation result of a Horn and Schunck flow field. The level set function is evolved using the same parameters as in the experiment shown in figure 8.6, but with $\delta = 100$ and $\delta' = 2.0$. Two initial level set evolutions was run before commencing with the combined evolution. The result seen above is obtained after one iteration of a 1:50 combined evolution.



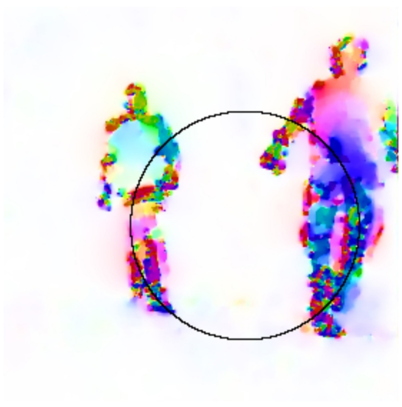
(a) Image 1 (b) Image 2 (c) Flow field

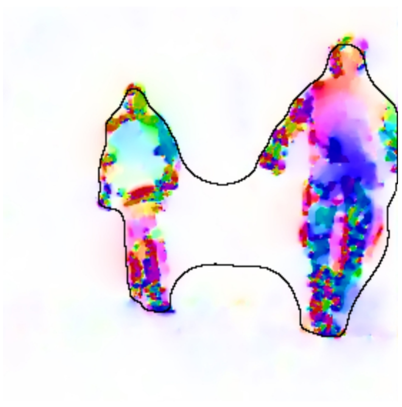
Figure 8.8: Figures 8.8a and 8.8b are two consecutive images in a sequence of two people walking, taken by a still camera. The optical flow field computed from the two images, using a flow driven smoothness term, is shown in figure 8.8c.

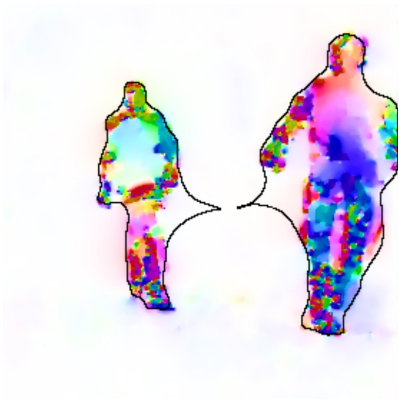
an automatic splitting, as shown in figure 8.9c. This splitting results in two sharp edges. The further evolution will remove those points that would cause the curve to self-intersect, so that the curve can lock onto the strongest flow boundaries. This is shown in figure 8.9d, which shows the segmentation result after 10 combined evolutions. These regions are also shown in figure 8.10. It is seen that the spline curve has failed to successfully describe the shoulder of the person on the right. This happens because the flow boundary is too weak for the edge detector to stop the level set evolution at this part of the boundary. Because of this, the curve will enter the body of the person.

A possible amendment to this issue is to choose a higher value for the sensitivity parameter in the edge detector. The segmentation result when running the same experiment, but with $\eta = 225$ is shown in figure 8.13. The curve is seen to give a better description of the shape of the shoulder, but the increased sensitivity parameter also leads to some issues. This is seen from noting the bad segmentation of the feet of the right person due to some noise in this area. The choice of sensitivity parameter leads to this noise being segmented as a part of the person.

Note that the combined evolution must be monitored to avoid numerical instability. Due to the added velocity term coming from the balloon force, the gradients of the level set function are unbounded, and the norms will increase linearly. However, this term may lead to problems if the parameters δ, γ and ϵ are not chosen correctly, which will cause an exponential increase


(a) Initial curve


(b) Seven iterations


(c) Eight iterations

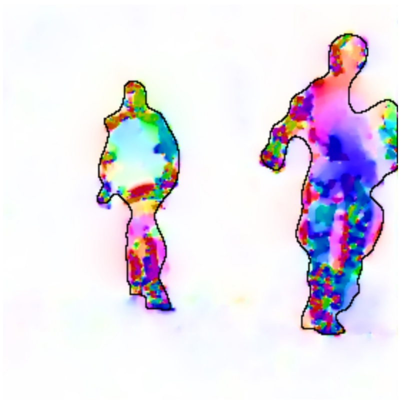

(d) Ten iterations

Figure 8.9: The segmentation of two people walking. The four figures show the spline curve in black drawn onto the color coded flow field. The experiment was run using level set parameters $\delta = 50$, $\eta = 150$, $\gamma = 0.9$ and $s = 0.001$, and spline parameters $\delta' = 0.25$, $\alpha = 1 \cdot 10^{-5}/(0.001^2 \times 614 \times 307)$ and $\beta = 2.0 \cdot 10^{-5}/\sqrt{614 \times 307}$. Four initial level set iteration were run as an initialization procedure, before a series of 1:50 combined evolutions. The curve was initiated as a circle with $K = 100$ control points. This initial spline curve is shown in figure 8.9a. Figure 8.9b shows the spline curve after seven combined evolutions. The spline curve is seen to segment both regions as one. Figure 8.9c shows the spline curve after eight combined evolutions. The automatic detection and splitting algorithm has detected a self-intersection, and split the curve, leaving two independent closed curves. Figure 8.9d shows the curve after ten combined evolutions. The spline curve is unable to correctly segment the shoulder of the right person due to a weak flow boundary in this area.

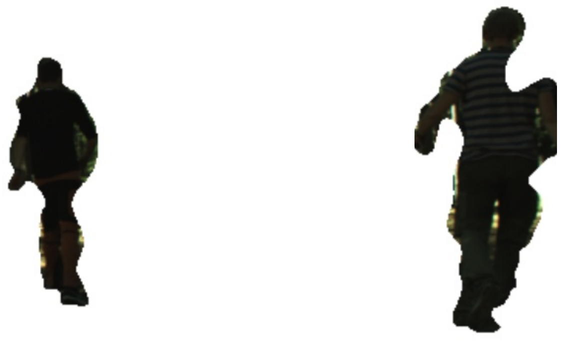
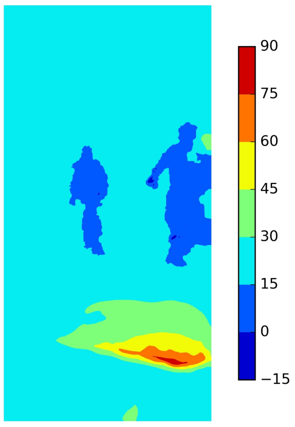
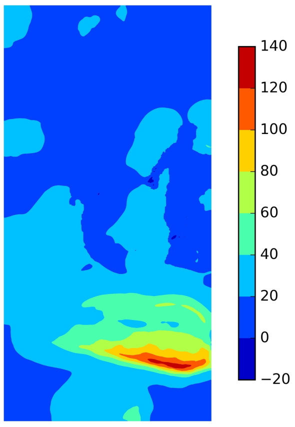

(a) Region 1 (b) Region 2

Figure 8.10: The two segmented regions from the experiment shown in figure 8.9. The concave regions are segmented relatively fast, resulting from a high balloon force parameter. A weak flow boundary on the shoulder of the person in figure 8.10b leads to the curve entering the body in this region, and fails to give a satisfying segmentation of the shoulder area.

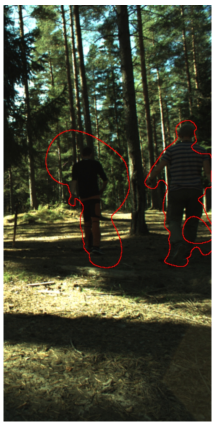
in the norms of the gradients of the level set function. In particular, if we choose the balloon force parameter γ too high, the level set evolution may become unstable. An example of such an instability is seen in figures 8.11a and 8.11b, which show contour plots of the level set function after 10 and 11 combined iterations respectively (a total of 14 and 15 iterations due to the initialization procedure of four level set iterations). The growing region beneath the two persons is not due to any flow gradients, but to a numerical instability. In this case, the value of φ_{ext} will become too large, and the curves will try to segment this area, as shown in figure 8.11c. Figure 8.12a shows a plot of the approximation of the energy $H(\varphi)$ given in equation (6.7) for different values of γ, keeping the other parameters fixed. A straight line indicates a stable increase in $H(\varphi)$, while an exponential increase indicates instability. The plot shows that lower values of γ will lead to a stable level set evolution for a longer time. For $\gamma = 0.9$ the curve starts increasing exponentially after $\tau = 400$, while the curve is seen to maintain a stable increase until $\tau = 1000$ for $\gamma = 0.7$. Figure 8.12b shows the same energy for two different values of the time step δ for $\gamma = 0.5$. It is seen that using $\delta = 100$ leads to the evolution becoming unstable at $\tau \approx 1500$. Reducing the step size from $\delta = 100$ to $\delta = 50$ yields a stable evolution after time $\tau = 3000$. The instability for the high values of the time step implies that the unstable evolution can be due to the semi implicit scheme that is used to solve the PDE for the level set function. Sethian [27, 28] suggested to solve the level set PDE using methods from hyperbolic conservation laws, based on higher-order upwind schemes, as such methods are more stable.



(a) Level set function after 10 iterations

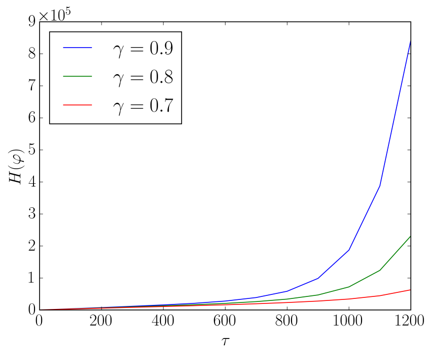


(b) Level set function after 11 iterations

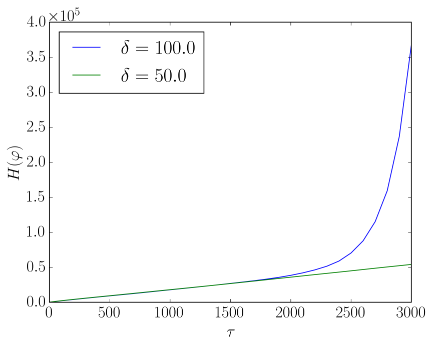


(c) Spline curve after 12 iterations

Figure 8.11: Unstable evolution of the level set function for the experiment shown in figure 8.8. Figures 8.11a and 8.11b show the contour plots of the level set function after 10 and 11 iterations respectively. It is seen that the level set function does no longer give a good description of the flow boundaries after 11 iterations. This is due to the velocity added by the balloon force. Figure 8.11c shows the resulting segmentation after 12 combined evolutions. It is seen that the continued evolution results in unsatisfactory segmentation results.



(a) The energy $H(\varphi)$ for different values of γ.



(b) The energy $H(\varphi)$ for different values of δ.

Figure 8.12: The figures show plots for the energy $H(\varphi)$. Note the different scaling on the y-axis for the two plots. Figure 8.12a shows the value of the energy $H(\varphi)$ (given in (6.7)) as a function of time τ for different values of γ, with the other parameters being constant. It is seen that lower values for γ leads to a more stable evolution. Figure 8.12b shows the same energy as a function of time for two different values of the time step size δ, keeping the other parameters constant. The figure shows that decreasing the step length can serve to stabilize the evolution.

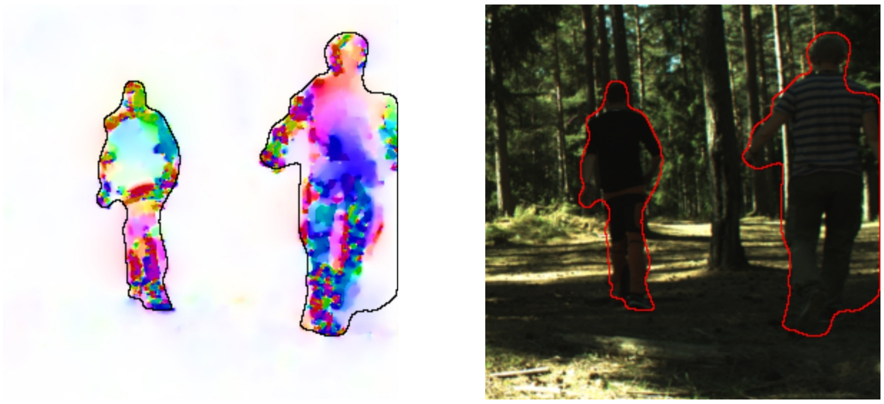
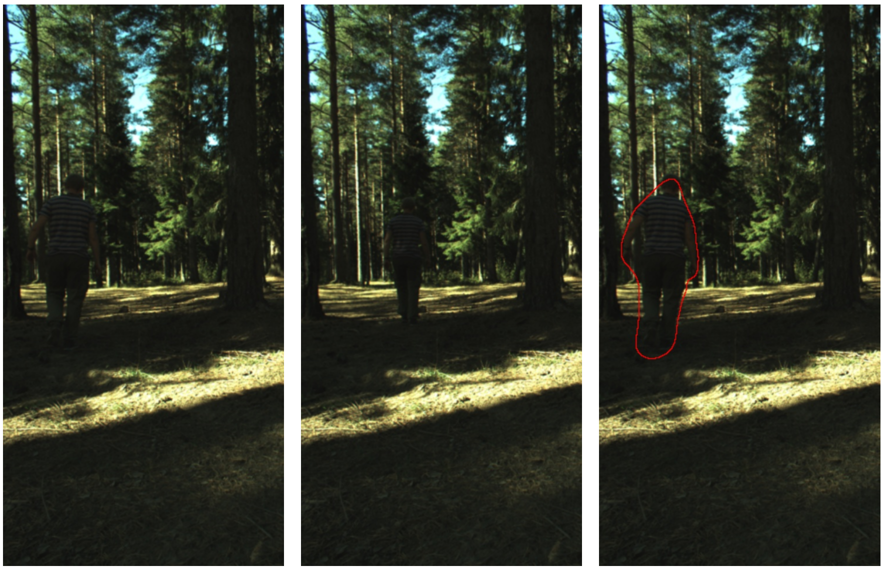


Figure 8.13: The segmentation result for the same experiment as shown in figure 8.9, but with a higher sensitivity parameter for the edge detector ($\eta = 225$). The shoulder of the person on the right is not as badly described as in the previous experiment, but the curve segments a large region close to the boundary, near the feet, due to some noise in the flow field.

8.3 Tracking movement

In this section we will look at how the spline curve can be used to track the motion of an object. We will use a sequence of images from the WMFR image sequence, where the first and last image of this sequence is shown in figures 8.14a and 8.14b respectively. The process consists of an initialization process and a tracking process, both using the same evolution parameters as in section 8.1, except from the changes $\delta = 200$, $\delta' = 1.3$ and $K = 30$. The initialization of the spline curve was performed using an evolution of one level set iteration followed by 200 spline iterations, and the resulting spline curve is shown as the red line in figure 8.14c.

In the tracking process we use a reinitialization of the level set for each step followed by 20 spline evolutions, using the spline curve from the previous segmentation as the starting point. Figure 8.15 shows the segmented region after each combined evolution in the tracking process. It is seen that the spline curve follows the shape of the body fairly well for most of the sequence. In the last step (the bottom right image of figure 8.15) the movement projected onto the camera is large, since the person is moving his feet together in one image. The resulting segmentation is seen to be affected by this, as the last image has a ”tail” hanging from the feet of the person. A possible explanation for this is that there may be a clustering of control points in this region due to the curve trying to describe the concave region between the legs five steps earlier. When the person is closing his feet together, the whole length of this slightly concave region in the bottom left of figure 8.15 is reduced to the small length occupied by the feet in the last image (bottom right). In this transition the points must distribute



(a) First image (b) Last image (c) Initial segmentation.

Figure 8.14: Figures 8.14a and 8.14b show the first and last image used for segmenting the movement of a person walking in a forest. This sequence consists of 18 images. Figure 8.14c shows the result of the initialization procedure of one level set iteration and 200 spline iterations. The movement segmentation for the rest of the sequence is shown in figure 8.15.

themselves along this short length.

8.4 Tracking movement with a moving camera

In this section we will look at a part of the DFM image sequence. The aim is to track the movement of the person in the DFM sequence using a similar strategy as in section 8.3. Starting from the signed distance function with scaling $s = 0.0001$ the level set is evolved using three consecutive iterations with parameters $\gamma = 0.5$, $\epsilon = 0.01$, $\eta = 200$ and $\delta = 200$. Then, we initialize the spline curve as a circle enclosing the person and do 400 evolutions using parameters $\alpha = 1.0 \cdot 10^{-7}/(0.0001^2 \times 600 \times 800)$, $\beta = 1.0 \cdot 10^{-5}/\sqrt{600 \times 800}$ and $\delta' = 1.3$. Figure 8.16c shows the result of this initial segmentation process.

The further tracking process is executed slightly different than the one in section 8.3. Instead of reinitializing the level set function at each step, we set $\delta = 20$ for the rest of the evolution. By using a small value for the step length of the level set function, we avoid the need for reinitialization due to unbounded gradients. The curve is seen to segment a part of the area under the shadow. This can be interpreted as the level set not evolving enough to closely segment the shadow of the person. Moreover, the curves displayed

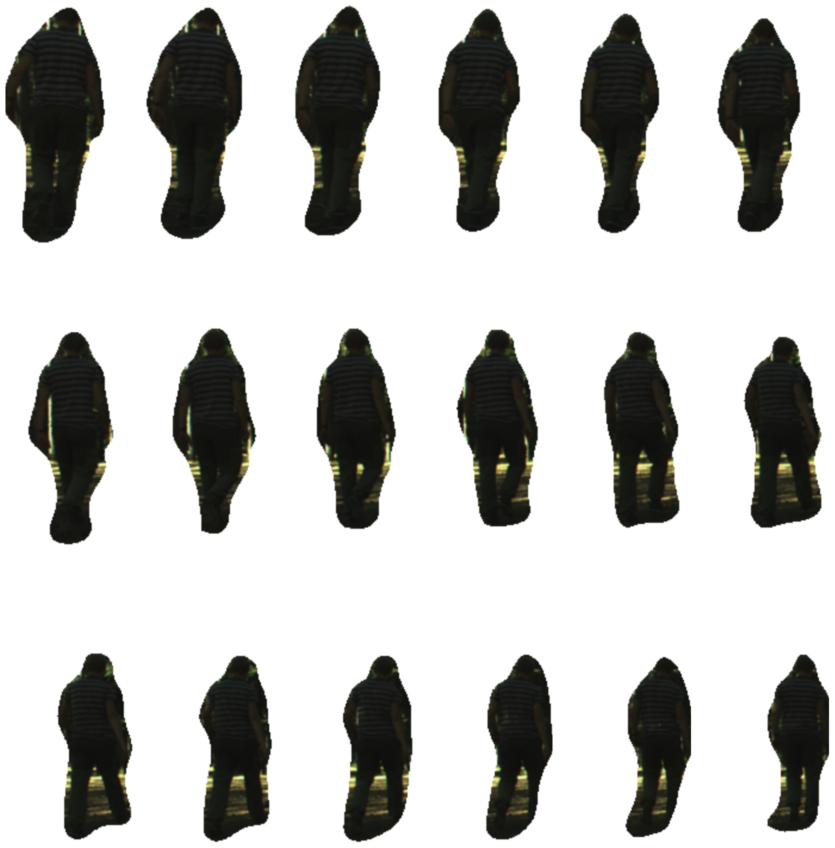
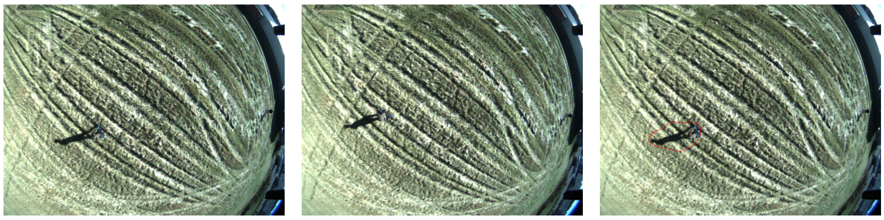


Figure 8.15: Tracking the movement of a person using flow segmentation. The parameters used in the level set evolution is $\gamma = 0.5$, $\eta = 100$, $\epsilon = 0.01$, $\delta = 20$ with φ^0 being the signed distance function with $s = 0.0001$. The spline curve consists of $K = 50$ points and the parameters used is $\alpha = 1.0 \cdot 10^{-7}/(0.0001^2 \times 614 \times 307)$, $\beta = 5.0 \cdot 10^{-5}/\sqrt{614 \times 307}$ and $\delta' = 1.3$. The initialization process was executed using one level set iteration and 200 spline iterations. Then, each consecutive segmentation is found by reinitializing the level set function (with the parameters above) and running 20 spline evolutions. The reinitialization is done to prevent numerical instability.



(a) First image (b) Last image (c) Initial segmentation.

Figure 8.16: Figures 8.16a and 8.16b shows the first and last image used for segmenting the movement of a person tracked by a flying drone. The person is moving across a part of the field in a sequence of 56 images. Figure 8.16c shows the result of the initial segmentation.

in the images on the fourth row are entering the shadow around its head. This issue can also be due to an under-evolved level set function. A solution to this problem could be to either use a larger step length in the level set iteration or to use more than one iteration of the combined evolution for each frame in the image sequence.

Figure 8.18 shows the result when running the same experiment as above, but now using two combined evolutions in each tracking step instead of one. The result is noticeably better. The first two columns of figure 8.19 shows selected frames from the two experiments. The curve is now tracking the shape of the person (and the shadow) more closely. Furthermore, the issue of the curve entering the shadow is no longer present, as seen from comparing figures 8.19a and 8.19b. Running one additional combined evolution is seen to give an apparent improvement of the segmentation result in each image frame. However, this improvement has its cost in the form of added computation time. The experiment above resulted in an average computation time of 8.4 seconds for each level set iteration and 11.4 seconds for 50 spline iterations, resulting in a total of 19.8 seconds for each additional combined evolution.

The computation time for each level set iteration is quite substantial, but can be significantly reduced by using a GMRES solver for solving the linear system. Running the same experiment as above with a GMRES solver instead of an exact solver for the level set iterations resulted in an average computation time of 2.3 seconds for each level set iteration. The residual tolerance was set to $tol = 0.0001$. The third column of figure 8.19 shows selected frames from the experiment using two combined evolutions with a GMRES solver. By comparing this with the results for the experiment using two combined evolution with an exact solver, shown in the second column, one can see that there are no visible differences. Hence, choosing an appropriate value for the tolerance in the GMRES scheme, one can obtain an apparently similar result with a considerable reduction in computation time.

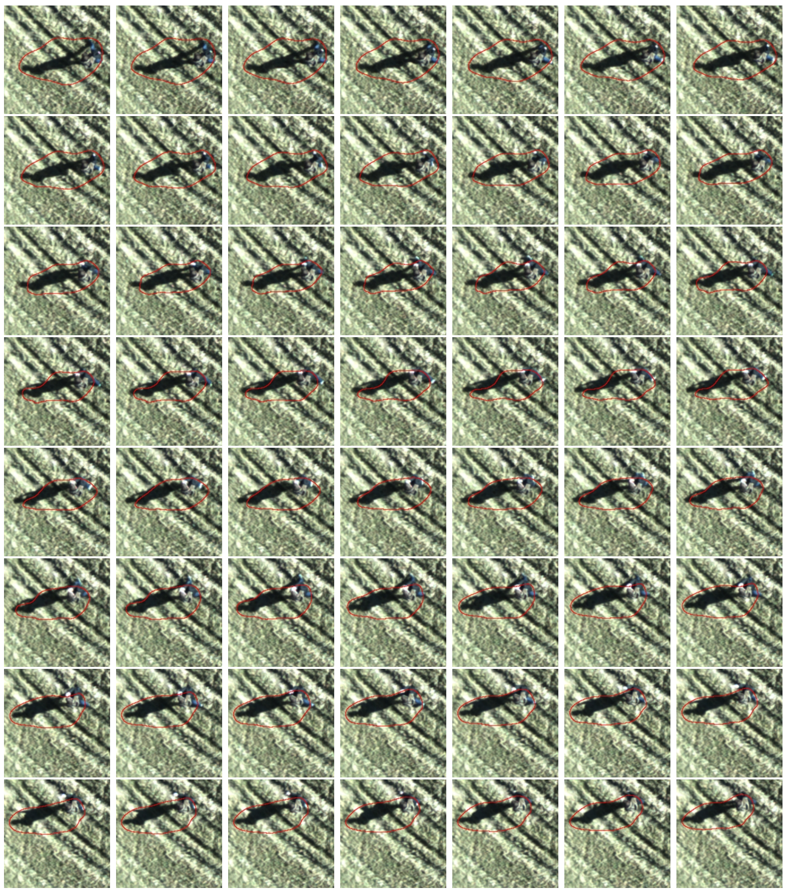


Figure 8.17: Segmentation of each frame in the drone sequence, using parameters $\gamma = 0.5$, $\eta = 200$ and φ^0 being the signed distance function with initial scaling $s = 0.0001$. The spline consists of $K = 20$ points and the parameters used are $\alpha = 1.0 \cdot 10^{-7}/(0.0001^2 \times 600 \times 800)$, $\beta = 1.0 \cdot 10^{-5}/\sqrt{600 \times 800}$. The first image shows the curve after an initialization procedure of three level set iterations with $\delta = 200$, followed by 400 spline iterations using $\delta' = 1.3$. The evolution of the consecutive curves are modelled using a decreased level set step size $\delta = 20$ and doing one level set iteration followed by 50 spline iterations with $\delta' = 1.3$. The evolution of the spline curve also uses the splitting algorithm in the case of a self-intersection.

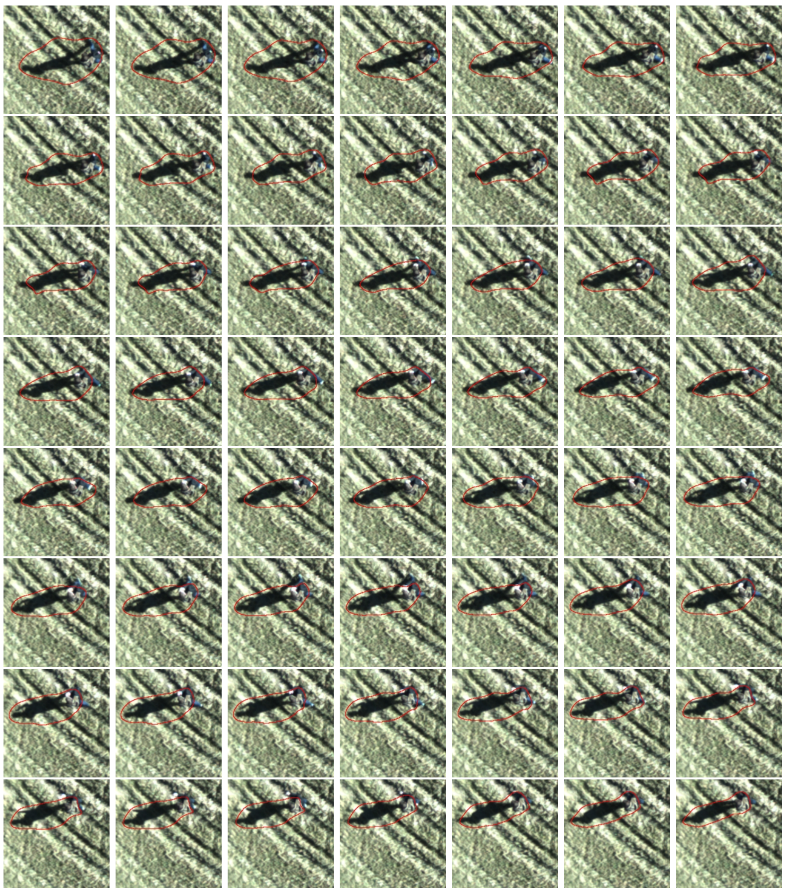
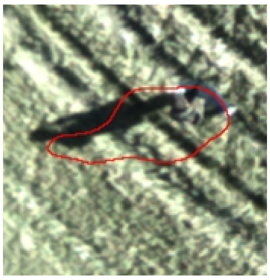
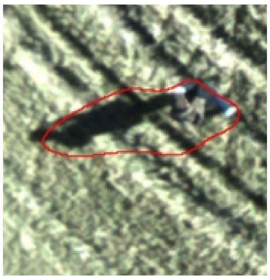
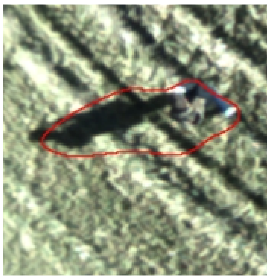
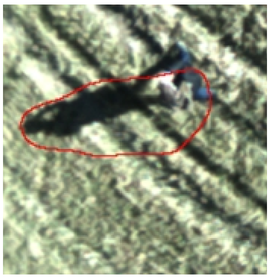


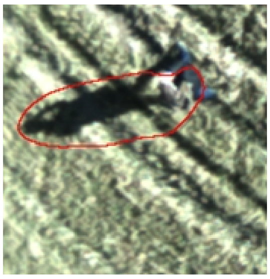
Figure 8.18: Segmentation of each frame in the drone sequence, using parameters $\gamma = 0.5$, $\eta = 200$ and φ^0 being the signed distance function with initial scaling $s = 0.0001$. The spline consists of $K = 20$ points and the parameters used are $\alpha = 1.0 \cdot 10^{-7}/(0.0001^2 \times 600 \times 800)$, $\beta = 1.0 \cdot 10^{-5}/\sqrt{600 \times 800}$. The first image shows the curve after an initialization procedure of three level set iterations with $\delta = 200$, followed by 400 spline iterations using $\delta' = 1.3$. The movement of the object is then tracked using two iterations of a 1:50 combined evolution scheme with step sizes $\delta = 20$ and $\delta' = 1.3$. The evolution of the spline curve also uses the splitting algorithm in the case of a self-intersection.

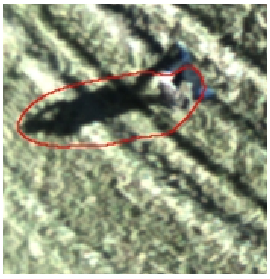

(a) Frame 25 using one combined evolution.

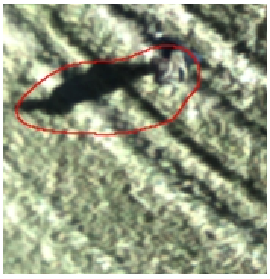

(b) Frame 25 using two combined evolutions.

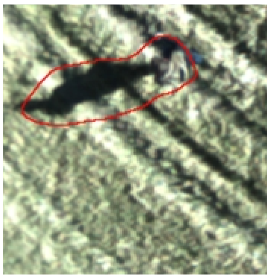

(c) Frame 25 using GMRES solver.


(d) Frame 38 using one combined evolution.


(e) Frame 38 using two combined evolutions.


(f) Frame 38 using GMRES solver.


(g) Frame 54 using one combined evolution.


(h) Frame 54 using two combined evolutions.

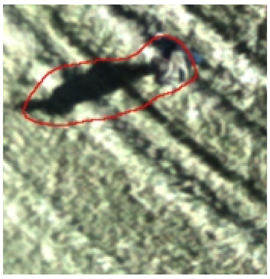

(i) Frame 54 using GMRES solver.

Figure 8.19: Comparison of the segmentation results for different frames in the DFM sequence using one combined evolution, two combined evolutions and two combined evolutions with a GMRES solver. The spline curve segmenting the image is shown in red. The first column shows the results from using one combined evolution consisting of one level set iteration and 50 spline iterations in each tracking step after the initialization procedure. The second column shows the results from using two combined evolutions in each step after the initialization. The third column shows the resulting segmentation when using two combined evolutions in each step where the level set iteration is solved using GMRES with residual tolerance 0.0001. Using two combined iterations (2nd column) is seen to be an improvement from using one combined evolution (1st column). The GMRES solver (3rd column) is seen to give the same visual result as the exact solver (2nd column).

8.5 General discussion and future work

This section presents a general discussion of the results presented in chapter 7 and chapter 8. We have seen that using the optical flow one can obtain a vector field which describes the movement in an image sequence, and that one can describe the contours of objects in this flow field by evolving a level set function. Choosing an appropriate value for the sensitivity parameter, the level sets of this function can in some sense describe the structure of the objects. By using a Mumford-Shah type segmentation on this level set function one can obtain a satisfying segmentation of the flow boundaries of objects in the image.

8.5.1 Unbounded gradients of the level set function

In the current implementation the combined evolution must be monitored. Due to the presence of an additional velocity term, which size is controlled by the parameter γ, the gradients of the level set function is unbounded. To avoid an unstable evolution of the spline curve as seen in figure 8.11 this unboundedness must either be avoided by choosing lower values of γ and δ in the evolution, or find a way to detect this issue and reinitialize the level set function when it happens. As previously discussed, the former option would lead to a slower level set evolution, and consequently a slower convergence, which is unwanted for obvious reasons. Thus, as future work we propose to implement an automatic detection and reinitialization algorithm that would handle this issue for the level set function. The detection part of the algorithm can be done by measuring the energy $H(\varphi)$ given in (6.7), and reinitialize the level set function if this energy becomes large in a relative sense. In this case we propose to reinitialize the level set function as the signed distance function of the spline curve C, that is,

$$\varphi = s\overline{d}(\boldsymbol{x}, C), \tag{8.1}$$

where $\overline{d}(\boldsymbol{x},\sigma)$ is defined in equation 6.5. This can be done ([2] p. 194) by solving

$$\frac{\partial\varphi}{\partial t} + \operatorname{sign}(C)(|\nabla\varphi| - 1) = 0, \tag{8.2}$$

$$\varphi(\boldsymbol{x}, 0) = C, \tag{8.3}$$

where

$$\operatorname{sign}(\sigma) = \left\{ \begin{array}{ll} 1 & \text{if } \boldsymbol{x} \in \operatorname{ext}(\sigma) \\ 0 & \text{if } \boldsymbol{x} \in \sigma \\ -1 & \text{if } \boldsymbol{x} \in \operatorname{int}(\sigma) \end{array} \right\}. \tag{8.4}$$

8.5.2 The splitting algorithm and the evolution of multiple curves

Some flow fields can lead to self-intersections in the curve. These self-intersections are handled by an automatic detection and splitting algorithm, which splits the curve depending on the number of control points on each side of the self-intersection. More precisely, a segment consisting of more than five control points will be made into a new closed curve. If it contains less than five control points the segment will be neglected and the control points will be dropped from the evolution.

In two of the image sequences we segmented multiple objects. This was done by initializing one curve at some appropriate starting location in the image, and choosing high values for the balloon force so that a split would occur. After the splitting the two curves are evolved separately and independently.

As seen from the segmentation of the Hamburg taxi sequence, the converged solution is dependent on the initialization of the curve. We previously noted that the initial curve must enclose the objects, but it is seen from experiments that one can segment even partially enclosed objects. However, in some cases the initialization can lead to an unwanted segmentation result (if flow boundaries are weak e.g.). As the numerical algorithm enables multiple curves to be evolved simultaneously, a possible amendment to this issue is to initialize multiple curves and evolve them in parallel.

8.5.3 Computation time

The total computation time depends on both the size of the problem, the segmentation accuracy and the method for computing optical flow.

Optical flow estimation

The Horn and Schunck method for computing optical flow involves solving a system of $2 \times m \times n$ unknowns. Hence, the computation time will depend on the size of the image. In the current implementation, the linear system is solved by either using the sparse linear solver in the Scipy library, which solves the system by LU-decomposition using the C library Super LU [17], or the Scipy implementation of the GMRES method. As previously mentioned, the computation time can be significantly reduced by using a GMRES scheme for solving the sparse linear system in the HS method. The computation of the flow image used in the experiment shown in figure 8.6 used approximately 2.7 seconds, corresponding to $\sim 14\%$ of the computation time using the built in sparse linear solver.

In their paper [6] Bruhn et al. suggested using a multigrid scheme combining the local method of Lucas-Kanade [18] with the global Horn and Schunck method to efficiently estimate the optical flow. The authors were able to compute the optical flow in real time using variatonal optical flow. This suggests that there are considerable enhancements to be made by with regards to the numerical implementation, and that variational optical flow can provide a good alternative for detecting motion in real-time.

In most of the motion segmentation experiments we have used a flow field computed using a flow driven regularization. Recall that this flow field was computed using an iterative method. Solving the system by an iterative method will increase the computation time, as one has to solve a linear system of $2 \times m \times n$ unknowns in each iteration. The method can be made to converge fast by choosing a higher value of ϵ_{OF} in the lagged diffusivity system, and the results seen in this chapter was computed using only 10 lagged diffusivity iterations. Still, the computation time using a flow driven regularization will always exceed the computation time of a method solving the system of $2 \times m \times n$ unknowns only once.

The segmentation system

In the segmentation part of the computations we are iteratively solving an equation for the level set function and an equation for the spline curve. The number of unknowns in the level set function is $m \times n$, and thus the computation time of this system will also depend on the size of the image. It has already been seen that the GMRES shceme can reduce the computation time considerably without altering the visual result of the segmentation. The number of iterations can be reduced by increasing the step size δ, as seen from figure 8.7, but care must be taken as the level set function will become unbounded for high values of γ and the spline curve evolution can become unstable when choosing a high δ'.

The spline curve iterations are in general faster than the level set iterations. This is due to the computation time not being directly dependent on the size of the image, but rather the number of control points K, the length of the curve L and the area enclosed by the curve A_{int}. To explain this, one needs to look at how the system is built inside the algorithm. In each step one needs to approximate the values $\overline{\varphi}_{int}$ and $\overline{\varphi}_{ext}$, and consequently a representation of the interior and the exterior of the curve. This is done by discretizing the curve in the same mesh as the image, and then tracing the curve to map the curve coordinates to pixels in the image, so that the traced pixels divide the mesh Ω into the interior and the exterior of the curve. A floodfill algorithm is then used to find a representation for $\text{int}(C)$. This part of the algorithm is obviously dependent on the number of pixels one has to trace to map the curve to the mesh, and also the number of iterations performed by the floodfill algorithm. Further, the right hand side of the equation is constructed by performing a numerical integration. This is done by using the trapezoidal rule, with some step size depending on the number of control points K. The numerical integration scheme has the potential to run faster than the current implementation due to restrictions in Python. For real-time computations, an adaptive integration scheme should be employed.

The computation time for solving the system using the FFT will also depend on the number of control points, and in particular whether the number of control points is even, odd or prime; the function is most efficient for even numbers and least efficient for prime numbers. The computation time of the FFT is of order $\mathcal{O}(K \log(K))$, and same for the inverse Fourier transform.

However, as $K << mn$, this computation time is negligible compared to the numerical integration.

8.5.4 The segmentation of concave regions

As seen from section 8.1 the method allows for an accurate description of concave regions. However, the required number of iterations is exceeding what is acceptable in any application expected to run in real time. This behaviour is expected, as the regularization term controlled by the parameter β will aim to minimize the length of the curve. Possible remedies was discussed in section 8.1, one of them being a control point insertion along segments where the curve is trying to describe a concave region. For the detection of these regions one could use the average value of the signed curvature over a given segment of the curve. If the signed curvature is large and negative the curve is concave. Having located this segment of the curve ΔC, one can find the point along this segment with the highest absolute value of the signed curvature and insert a new control point close to this point. Some care must be taken as to not insert a new control point too close to an existing control point to avoid numerical issues.

As the convex segments of the curve are segmented faster than the concave region, an additional remedy could be to evolve only the control points of concave regions after some initial combined evolutions. This is done by extracting a submatrix of the circulant matrix in (5.12) and the corresponding elements of the right hand side vector, and solving this system in some additional evolution steps independent of the rest of the control points.

8.5.5 Choosing the method for optical flow estimation

Which method to use for computing the optical flow depends on the required accuracy of the segmentation, and the need for an accurate description of objects. For practical applications the use of optical flow data may be time critical. One example of such an application is real-time surveillance and automated movement detection. In such a scenario a coarse contour may be of more use than a perfectly matched segmentation, as the validity and value of the data may decrease rapidly over time. Another example is the use of motion segmentation computed by small unmanned vehicles, these may carry a camera and be able to collect large amounts of data but the size of the platform will limit the available computational resources and a video stream may quickly overwhelm even high capacity wireless links. In this case, optical flow and segmentation methods could be used to reduce the required bandwidth by only transferring part of images with movement, given that the required computation can be done at sufficient speed. This encourages the use of a non-iterative method for optical flow estimation, like the method of Horn and Schnuck. As seen from figure 8.7, a satisfying result with respect to the requirements stated above can be obtained in only one combined iterations for the right choice of optical flow parameters. However, the method is less robust than the flow driven approach, and noisy images or poor choice of parameters may lead to noise in the flow field, which can lead to a failed segmentation.

The experiment shown in figure 8.8 shows that weak flow boundaries can lead to issues in the segmentation process. The shoulder of the right person has a non-localized flow boundary, and thus the edge detector fails to stop the evolution at this edge. This is a known drawback from the flow driven approach. The image and flow driven approach [32] combines the sharp boundaries of the image driven approach with the reduced oversegmentation of the flow driven approach. The method is anisotropic as it uses the eigenvectors of the structure matrix in equation (3.26) to smooth the flow field along image edges, while using the subquadratic penalizer shown in equation (3.30) to decrease smoothing at flow boundaries. Chapter 7 of [33], based on the framework presented by Zimmer et al. [35], gives the details of this method for the choice $\rho = 0$ in the structure matrix, and suggests to solve the system by a lagged diffusivity iteration. The results are reported to give more localised flow boundaries than the isotropic flow driven method. This combined image and flow driven regularization method can be a good alternative if the application requires a more accurate and robust segmentation then the Horn and Schunck method can provide. Similar to the flow driven regularization, the combined image and flow driven regularization results in a nonlinear system, and must be solved iteratively. Hence, the computation time can not be expected to improve from the flow driven method.

8.5.6 Combined algorithm for localizing flow boundaries

An idea that has been considered as a viable approach for improving the estimation of the flow field is to construct a combined algorithm for computing the optical flow and segmentation. The aim of this combined algorithm is to use the segmentation boundary in regularizing the optical flow. As future work we propose to look into how an iterative scheme can be used to obtain better segmentation results, for which a proposed outline is shown in algorithm 3.

Algorithm 3 Combined flow and segmentation algorithm

```
Given initial flow field $w_0$ and contour $C_0$
$k = 0$
while Segmentation is non-satisfactory do
    Estimate flow field $w_{k+1}$ using flow field $w_k$ and contour $C_k$
    Estimate contour $C_{k+1}$ using flow field $w_{k+1}$ and contour $C_k$
    $k = k + 1$
end while
return $w_k$ and $C_k$
```

8.5.7 Adding a shape prior to the segmentation model

In many applications motion segmentation is used to detect a certain type of objects, and so it makes sense to incorporate this information in the segmentation model. By incorporating this information into the model prior to the search, one could guide the evolution, looking for shapes that are similar to the shapes of these objects. To do this one needs some way to measure the similarity of shapes, and to extract information about a particular shape. Appendix C gives an outline for a theory on shape representations along with a suggestion for how this can be incorporated into the existing model.

9 Conclusion

The framework proposed in this paper segments moving objects from an image sequence by performing an active contours segmentation on an optical flow field. The constructed numerical algorithm can be divided into two parts; *the optical flow estimation* and *the active contours segmentation*.

The segmentation aims to evolve an initial contour to determine shape of moving object using the boundaries of the optical flow field. This is done by using a flow edge detector to stop the evolution of the level sets of some function at these boundaries. The contour of the moving objects are represented by a cubic periodic B-spline curve, and the evolution of this contour is guided by a modified Mumford-Shah functional, resulting in a combined evolution of level sets and B-spline curves as proposed by Fusch et al. [12].

The optical flow problem is ill-posed and requires the use of regularization to obtain a unique solution. Three methods of regularizing optical flow has been compared based on the choice of edge detector and the resulting segmentation. These methods are the isotropic regularization of Horn and Schunck [13], the anisotropic image driven regularization of Nagel and Enkelmann [22] and the isotropic flow driven method [29].

The use of a purely image driven regularization was discarded due to large amounts of image structure in the optical flow field. The use of the isotropic regularization of Horn and Schunck yields a very smooth flow field with non-localized flow boundaries. Since the evolution of the segmentation boundaries will stop at the detected flow boundaries, this may lead to a segmentation where the contour gives an inaccurate description of the moving objects. In particular, the segmentation process may fail to describe concave segments due to the isotropic smoothing. The method has a clear advantage in the form of a relatively low computation time. The use of segmentation in real-time applications does not require high accuracy in the description of objects. Consequently, the isotropic smoothing of Horn and Schunck is a viable method for estimating optical flow in real-time applications. The flow driven method reduces smoothing at flow boundaries. This has the effect that the boundaries are more localized than for the Horn and Schunck method, which will lead to more localized segmentation boundaries. A number of segmentation experiments have been executed on flow fields produced by the use of this regularization method, showing both the performance of the segmentation algorithm and the quality of the flow fields. We have shown that the flow driven method gives flow fields which are in many cases suitable for segmentation. However, the flow driven method has the disadvantage of increased computation time compared to the two other methods.

The segmentation algorithm has been tested on three real-world data sets, demonstrating the performance of the combined evolution along with the splitting algorithm. If the boundaries of the flow field are well localized, the evolution will converge to the outer flow boundaries of the moving objects. It is seen that the contour is able to describe convex regions of the objects in relatively few iterations, while concave regions takes longer time to successfully segment. How fast the curve deforms into these regions can be controlled by the balloon force. The evolution, and also the converged solution, of the active contour is seen to be dependent on the parameters of the segmentation model and the initial data.

Multiple objects can be segmented by initializing a single curve. This is done by initializing the curve appropriately and choosing a high value for the balloon force. The level set evolution is seen to force the contour to self intersect. We have seen that the algorithm automatically detects these self-intersections, and handles them by either splitting the curve or removing control points. If the curve is split, the two resulting curves will evolve independently after the splitting.

Appendices

A The Euler-Lagrange equation

We derive the Euler-Lagrange equation. Let

$$J(w) = \iint\limits_{\Omega} F(x^1, x^2, w, w_{x^1}, w_{x^2})\, dx^1\, dx^2.$$

For an element w minimizing $J(w)$, the first variation must be zero, that is,

$$\delta J(w;\eta) = \frac{d}{d\epsilon}\bigg|_{\epsilon=0} [J(w+\epsilon\eta)] = 0,$$

at $\epsilon = 0$ for any arbitrary function $\eta(x^1, x^2)$. We get

$$\begin{aligned}\delta J(w;\eta) &= \iint\limits_{\Omega} \frac{d}{d\epsilon}\bigg|_{\epsilon=0} F(x^1, x^2, w+\epsilon\eta, w_{x^1}+\epsilon\eta_{x^1}, w_{x^2}+\epsilon\eta_{x^2})\, dx^1\, dx^2 \\ &= \iint\limits_{\Omega} \eta F_w + \eta_{x^1} F_{w_{x^1}} + \eta_{x^2} F_{w_{x^2}}\, dx^1\, dx^2 \\ &= \iint\limits_{\Omega} \eta F_w + \frac{d}{dx^1}(\eta F_{w_{x^1}}) + \frac{d}{dx^2}(\eta F_{w_{x^2}}) - \eta\left(\frac{d}{dx^1}F_{w_{x^1}} + \frac{d}{dx^2}F_{w_{x^2}}\right) dx^1\, dx^2.\end{aligned}$$

Now let Γ_E, Γ_W, Γ_N and Γ_S be the east, west, north and south boundary of our domain respectively. Then using Gauss' Theorem gives

$$\begin{aligned}&\iint\limits_{\Omega} \frac{d}{dx^1}(\eta F_{w_{x^1}}) + \frac{d}{dx^2}(\eta F_{w_{x^2}})\, dx^1\, dx^2 \\ &= \int_{\Gamma_E} \eta F_{w_{x^1}}\, dx^1 - \int_{\Gamma_W} \eta F_{w_{x^1}}\, dx^1 + \int_{\Gamma_N} \eta F_{w_{x^2}}\, dx^2 - \int_{\Gamma_S} \eta F_{w_{x^2}}\, dx^2.\end{aligned}$$

Using this result, we get

$$\begin{aligned}\delta J(w;\eta) &= \iint\limits_{\Omega} \eta\left(F_w - \frac{d}{dx^1}F_{w_{x^1}} - \frac{d}{dx^2}F_{w_{x^2}}\right) dx^1\, dx^2 \\ &+ \left(\int_{\Gamma_E} \eta F_{w_{x^1}}\, dx^1 - \int_{\Gamma_W} \eta F_{w_{x^1}}\, dx^1 + \int_{\Gamma_N} \eta F_{w_{x^2}}\, dx^2 - \int_{\Gamma_S} \eta F_{w_{x^2}}\, dx^2\right) = 0.\end{aligned}$$

Since this must hold for any arbitrary function $\eta(x^1, x^2)$ it follows that

$$\begin{aligned} F_w - \frac{d}{dx^1}F_{w_{x^1}} - \frac{d}{dx^2}F_{w_{x^2}} &= 0 \quad \text{in } \Omega, \\ F_{w_{x^1}} &= 0 \quad \text{on } \Gamma_E \text{ and } \Gamma_W, \\ F_{w_{x^2}} &= 0 \quad \text{on } \Gamma_N \text{ and } \Gamma_S.\end{aligned}$$

This is called the Euler-Lagrange equation of variational calculus.

B The elements of the circulant matrix

In evolving the spline curve, we are concerned with solving the system

$$A\frac{\partial p^c}{\partial \tau} = \Phi^c, \tag{B.1}$$

for $c \in \{1,2\}$, where the elements of the matrix A are

$$A_{ij} = \int_0^1 \psi^i(t)\psi^j(t)\,dt. \tag{B.2}$$

Since the curve is periodic, this matrix will be a circulant matrix. The functions $\psi^k(t)$ are the basis functions of the cubic B-spline, which are known. This means that the integral in (B.2) can be computed analytically. Recall that the basis functions are given as

$$\psi^k(t) = \begin{cases} a(K(t-t_{k-2})) & \text{for } t_{k-2} \le t \le t_{k-1}, \\ b(K(t-t_{k-1})) & \text{for } t_{k-1} \le t \le t_k, \\ c(K(t-t_k)) & \text{for } t_k \le t \le t_{k+1}, \\ d(K(t-t_{k+1})) & \text{for } t_{k+1} \le t \le t_{k+2}, \end{cases} \tag{B.3}$$

with

$$a(s) = \frac{s^3}{6}, \qquad b(s) = \frac{-3s^3+3s^2+3s+1}{6},$$
$$c(s) = \frac{3s^3-6s^2+4}{6}, \qquad d(s) = \frac{-s^3+3s^2-3s+1}{6}.$$

The $K \times K$ circulant matrix can be defined by cyclically permuting a circulant vector a of length K. We will define this vector as the first row of the matrix, and calculate its values. As each basis function has compact support over four subintervals $[t_k, t_{k+1}] \subset [0,1]$, the vector has seven nonzero entries where four of them are distinct. Figures 6.1 and 6.2 can help to convince the reader of this statement. The vector can be written as

$$a = [a_0, a_1, a_2, a_3, 0, \dots, 0, a_3, a_2, a_1], \tag{B.4}$$

and the elements are given as

$$\begin{aligned} a_0 &= \int_0^1 \psi^i(t)\psi^i(t)\,dt & a_1 &= \int_0^1 \psi^i(t)\psi^{i-1}(t)\,dt \\ a_2 &= \int_0^1 \psi^i(t)\psi^{i-2}(t)\,dt & a_3 &= \int_0^1 \psi^i(t)\psi^{i-3}(t)\,dt. \end{aligned} \tag{B.5}$$

The following shows the computation of these four integrals:

$$
\begin{aligned}
a_0 &= \int_0^1 \psi^i(t)\psi^i(t)\,dt = \int_{t_{i-2}}^{t_{i+2}} \psi^i(t)\psi^i(t)\,dt \\
&= \int_{t_{i-2}}^{t_{i-1}} \psi^i(t)\psi^i(t)\,dt + \int_{t_{i-1}}^{t_i} \psi^i(t)\psi^i(t)\,dt + \int_{t_i}^{t_{i+1}} \psi^i(t)\psi^i(t)\,dt + \int_{t_{i+1}}^{t_{i+2}} \psi^i(t)\psi^i(t)\,dt \\
&= \int_0^1 a(s)a(s)\frac{dt}{ds}\,ds + \int_0^1 b(s)b(s)\frac{dt}{ds}\,ds + \int_0^1 c(s)c(s)\frac{dt}{ds}\,ds + \int_0^1 d(s)d(s)\frac{dt}{ds}\,ds \\
&= \frac{1}{36K}\int_0^1 \left(s^3\right)^2 + \left(-3s^3+3s^2+3s+1\right)^2 \\
&\qquad\qquad + \left(3s^3-6s^2+4\right)^2 + \left(-s^3+3s^2-3s+1\right)^2\,ds \\
&= \frac{1}{36K}\int_0^1 20s^6 - 60s^5 + 42s^4 + 16s^3 - 18s^2 + 18\,ds \\
&= \frac{1}{36K}\frac{604}{35}, \\
a_1 &= \int_0^1 \psi^i(t)\psi^{i-1}(t)\,dt = \int_{t_{i-2}}^{t_{i+1}} \psi^i(t)\psi^{i-1}(t)\,dt \\
&= \int_{t_{i-2}}^{t_{i-1}} \psi^i(t)\psi^{i-1}(t)\,dt + \int_{t_{i-1}}^{t_i} \psi^i(t)\psi^{i-1}(t)\,dt + \int_{t_i}^{t_{i+1}} \psi^i(t)\psi^{i-1}(t)\,dt \\
&= \int_0^1 a(s)b(s)\frac{dt}{ds}\,ds + \int_0^1 b(s)c(s)\frac{dt}{ds}\,ds + \int_0^1 c(s)d(s)\frac{dt}{ds}\,ds \\
&= \frac{1}{36K}\int_0^1 \left(s^3\right)\left(-3s^3+3s^2+3s+1\right) + \left(-3s^3+3s^2+3s+1\right)\left(3s^3-6s^2+4\right) \\
&\qquad\qquad + \left(3s^3-6s^2+4\right)\left(-s^3+3s^2-3s+1\right)\,ds \\
&= \frac{1}{36K}\int_0^1 -15s^6 + 45s^5 - 33s^4 - 9s^3 + 12s^2 + 8\,ds \\
&= \frac{1}{36K}\frac{1191}{140},
\end{aligned}
$$

$$\begin{aligned}
a_2 &= \int_0^1 \psi^i(t)\psi^{i-2}(t)\,dt = \int_{t_{i-2}}^{t_i} \psi^i(t)\psi^{i-2}(t)\,dt \\
&= \int_{t_{i-2}}^{t_{i-1}} \psi^i(t)\psi^{i-2}(t)\,dt + \int_{t_{i-1}}^{t_i} \psi^i(t)\psi^{i-2}(t)\,dt \\
&= \int_0^1 a(s)c(s)\frac{dt}{ds}\,ds + \int_0^1 b(s)d(s)\frac{dt}{ds}\,ds \\
&= \frac{1}{36K}\int_0^1 \left(s^3\right)\left(3s^3-6s^2+4\right)+\left(-3s^3+3s^2+3s+1\right)\left(-s^3+3s^2-3s+1\right)ds \\
&= \frac{1}{36K}\int_0^1 6s^6-18s^5+15s^4-3s^2+1\,ds \\
&= \frac{1}{36K}\frac{6}{7}, \\
a_3 &= \int_0^1 \psi^i(t)\psi^{i-3}(t)\,dt = \int_{t_{i-2}}^{t_{i-1}} \psi^i(t)\psi^{i-3}(t)\,dt \\
&= \int_0^1 a(s)d(s)\frac{dt}{ds}\,ds \\
&= \frac{1}{36K}\int_0^1 \left(s^3\right)\left(-s^3+3s^2-3s+1\right)ds \\
&= \frac{1}{36K}\int_0^1 -s^6+3s^5-3s^4+s^3\,ds \\
&= \frac{1}{36K}\frac{1}{140}.
\end{aligned}$$

The number K corresponds to the number of control points, and consequently the number of basis functions in the spline curve.

C Shape analysis and statistical shape priors

In the segmentation process we aim to find a curve that segments moving objects by using a level set function. Let us now assume that we know the type of objects we are looking for in advance of the segmentation. By incorporating this information into the model prior to the search, one could guide the evolution, looking for shapes that are similar to the shapes of these objects. To do this one needs some way to measure the similarity of shapes, and to extract information about a particular shape. This is the goal of statistical shape analysis.

This chapter will discuss the use of shape statistics as a measure for obtaining better results in the segmentation process. The methods presented here have not been implemented, but the theory is to be read as a suggestion for future work. We start by presenting some theory on how shapes can be presented in section C.1. We will look at a particular shape representation called the *square-root velocity*, given in section C.1.1. Further, section C.1.2 proceeds to present a shape representation called th orbits of a curve. Section C.2 gives the outline for an approach of how one might incorporate a prior shape into the active contours model using these orbits.

C.1 Shape representation

One of the most prominent ideas for representing shapes has been the concept of landmarks [16], which are finite collections of important feature points. These points could be corners or infliction points, or they could be anatomical landmarks that specify some biologically meaningful points, like the joints in a human body. One might argue that this way of representing points is too subjective, as landmarks can be chosen differently for the same objects, and that object boundaries is better seen as being continuous. Thus a recent focus in this area has been the formulation of shapes as elements in infinite-dimensional Riemannian manifolds called shape spaces. Srivastava et al. [30] introduced a convenient shape representation for parametrized curves in such a shape space, by defining a shape representation called the square-root velocity (SRV).

C.1.1 The square root velocity representation

Going back to our parametrized curve $C(t)$, the SRV of C is defined as a mapping $q : [0,1] \to \mathbb{R}^2$ given by

$$q(t) = \begin{cases} C'(t)/\sqrt{\|C'(t)\|} & \text{if } \|C'(t)\| \neq 0 \\ 0 & \text{if } \|C'(t)\| = 0 \end{cases} \tag{C.1}$$

where $C'(t)$ denotes the derivative of C with respect to t. The mapping is not injective, but for every $q \in L^2([0,1], \mathbb{R}^2)$ there exist a curve, unique up to a translation, such that the SRV of this curve is equal to q. This curve is given by

$$C(t) = \int_0^t q(s)\|q(s)\|\,ds. \tag{C.2}$$

To make the curve representation scale invariant, the authors of [30] proceeded to rescale the curves to unit length, so that

$$\int_0^1 \|C'(t)\|\,dt = \int_0^1 \|q(t)\|^2\,dt = 1.$$

Further, since all curves considered here are closed curves, we have $C(0) = C(1)$, which gives

$$\int_0^1 q(t)\|q(t)\|\,dt = 0.$$

We can now proceed to present what is called the preshape space of closed curves P represented by the SRV such that

$$P = \left\{ q \in L^2([0,1],\mathbb{R}^2) : \int_0^1 \|q(t)\|^2\,dt = 1, \int_0^1 q(t)\|q(t)\|\,dt = 0 \right\}. \tag{C.3}$$

This is a submanifold of $L^2([0,1],\mathbb{R}^2)$, and it can be shown that the tangent space of P is a well defined subset of $L^2([0,1],\mathbb{R}^2)$, so that Riemannian structures can be defined on the preshape space. In particular we have the following (see [30] for details):

Theorem 1. *The image of the SRV mapping given in (C.1) for closed curves is a submanifold of $L^2([0,1],\mathbb{R}^2)$. The normal space is given by*

$$N(q) = \operatorname{span}\left\{ q(t), \left(\frac{q_i(t)}{\|q(t)\|} q(t) + \|q(t)\| \boldsymbol{e}_i \right), i = 1,2 \right\} \tag{C.4}$$

From the theorem above we can characterize the tangent space as

$$T(q) = \left\{ v \in L^2([0,1],\mathbb{R}^2) : < v, w >= 0, \forall w \in N(q) \right\}. \tag{C.5}$$

Now, let $q_1, q_2 \in P$ and choose a smooth parametrized path $\alpha : [0,1] \to P$ starting in q_1 and ending in q_2. The length of α in P is

$$L(\alpha) = \int_0^1 \|\alpha'(\tau)\|\,d\tau, \tag{C.6}$$

and the distance between q_1 and q_2 is given by

$$d_p(q_1, q_2) = \inf_\alpha L(\alpha).$$

From [3] we have that the geodesic distance is given by the L^2-norm

$$d_p(q_1, q_2) = \int_0^1 (q_1(t) - q_2(t))^2 \, dt. \tag{C.7}$$

Let us stop here, to dwell on what was just presented: P is now a space of shape representations $q \in L^2([0,1], \mathbb{R})$, for shapes with the same scaling and where translation has been removed. The distance measure $d(\cdot,\cdot)$ is the distance between two parametrized curves in this shape space.

C.1.2 Orbits and the shape space

Preferably, we would like our shape space to be constructed so as to also consider rotations and reparametrizations, and we would like curves that have the same shape, but with different rotations and parametrizations, to map to the same point in the shape space. To this end, consider a rotation matrix O,

$$O = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix}, \tag{C.8}$$

for $\theta \in \mathbb{R}$. The rotation of a closed curve C is given by $(O, C(t)) = OC(t)$, and it is seen that the SRV $\tilde{q}$ of the rotated curve is given by

$$\tilde{q} = \frac{OC'(t)}{\sqrt{\|OC'(t)\|}} = \frac{OC'(t)}{\sqrt{\|C'(t)\|}} = Oq(t),$$

where q is the SRV of C. That is, rotating the curve $C(t) \in \mathbb{R}^2$ results in the same rotation of $q(t) \in \mathbb{R}^2$. Further, consider reparametrizations $\gamma \in \Gamma$, where Γ is the set of all orientation-preserving diffeomorphisms of $[0,1]$, that is,

$$\gamma : [0,1] \to [0,1], \tag{C.9}$$

such that $\gamma \in C^\infty$, $\gamma(0) = 0$ and $\gamma(1) = 1$. To see the result of a reparametrization γ, we let $C(\gamma(t))$ denote the reparametrized curve and $\tilde{q}$ its SRV. Letting q be the SRV of C, then $\tilde{q}$ is given by

$$\tilde{q} = \frac{C'(\gamma(t))\gamma'(t)}{\sqrt{\|C'(\gamma(t))\gamma'(t)\|}} = \frac{C'(\gamma(t))\sqrt{\gamma'(t)}}{\sqrt{\|C'(\gamma(t))\|}} = q(\gamma(t))\sqrt{\gamma'(t)}.$$

From [30] we also have that the actions of rotation and reparametrization commute, and that the action $((O,\gamma), q) = O(q \circ \gamma))\sqrt{\gamma'}$ of the product group $\Gamma \times SO(2)$ is an isometry with respect to the chosen metric. We now define our shape space S as the quotient space

$$S = \{[q] : q \in P\}, \tag{C.10}$$

where $[q]$ is called the orbit of q and is the set

$$[q] = \left\{ O(q \circ \gamma)\sqrt{\gamma'} : (\gamma, O) \in \Gamma \times SO(2) \right\}.$$

The orbit associated with a shape is unique, and will be the representation for measuring the similarity between shapes. It is shown in [30] that S inherits the Riemannian structure from P, and that the geodesic distance between points in S is

$$d_s([q_1],[q_2]) = \inf_{(\gamma,O)\in\Gamma\times SO(2)} d_p(q_1, O(q_1 \circ \gamma)\sqrt{\gamma'}). \tag{C.11}$$

For the computation of this distance we refer to the path straightening method presented in [30]. The reminder of this chapter will focus on the statistical methods of building and incorporating shape priors in the segmentation process.

C.2 Bayesian active contours

The active contours approach presented in chapter 4 aims to draw some active contour C to lock on to image boundaries by minimizing some energy. Let us now assume that we know, prior to the active contours segmentation, the type of objects in the image, or rather the shape of the subset of Ω we are trying to segment. In this section we aim to provide a framework for how one may incorporate this information in the active contours model. To do this, we will utilize the shape space S and the geodesic distance based on the SRV, presented in the previous section.

Let us assume that we have N samples of car shapes, and we want to create a prior shape segment cars from an image. A natural thing would be to use the mean of the shapes. The Karcher mean is defined as the element $\mu \in S$ that minimizes $\sum_{i=1}^{N} d([\mu],[q_i])^2$, where q_i is the SRV of the curve with parametrization $C_i(t)$. Joshi [14] presented a comprehensive framework for representing and building statistical shape priors, and we refer to his phd-thesis for a more extensive treatment than what is presented here.

C.2.1 Choosing the curve representation for shape extraction

In section 4.2 we presented the method of representing the evolving contour as the level set of some function ϕ. Chapter 5 presented a Mumford-Shah segmentation of this level set function using a parametrized curve. The question arises as to which of the two ways of describing the active contour to use for the shape extraction process. Joshi [14] argued that using a level set, one might run into difficulties when trying to build a representation that is invariant to similarity transformations. Thus, we will assume that the parametrized curve $C(t)$, found through the Mumford-Shah segmentation, describes the active contours. The parametrized curve representation also coincides with the intrinsic shape analysis of section C.1, and we can directly apply the SRV transformation to the paramterized curve. This will allow us to make use of the orbit representation for shapes, and consequently the quotient space S defined in equation (C.10). The use of this shape space can be further justified by looking at the choice of shape metric and how this metric compares shapes. To do this, we will look at the physical interpretation of the shape matching.

C.2.2 Elastic shape deformations

When going from one shape to another in the shape space, it is preferable that the changes to the curve itself corresponds to elastic deformations like stretching, compression and bending. A metric that achieves this is called an elastic metric. The following definition gives the elastic representation of shapes:

Definition 1. *Let $C : [0,1] \to \mathbb{R}^2$ be a parametrized curve and define $\phi : [0,1] \to \mathbb{R}$ by*

$$\phi(t) = \ln(\|C'(t)\|),$$

and $\theta : [0,1] \to \mathbb{R}^2$ by

$$\theta(t) = \frac{C'(t)}{\|C'(t)\|}.$$

The pair $(\phi,\theta) \in (\Phi \times \Theta)$ is called the elastic representation of C and defines a set of curves, unique up to a translation, such that

$$C'(t) = e^{\phi(t)}\theta(t).$$

In terms of the elastic representation, the SRV is given as

$$q(t) = e^{1/2\phi(t)}\theta(t).$$

Mio et. al [19] gives the following definition of an elastic Riemannian metric on $(\Phi \times \Theta)$:

Definition 2. *Let a and b be positive real numbers, $(\phi,\theta) \in (\Phi \times \Theta)$ and let h_i and f_i represent infinitesimal deformations of ϕ and θ, so that (h_1, f_1) and (h_2, f_2) are tangent vectors to $(\Phi \times \Theta)$ at (ϕ,θ) . Define the inner product*

$$\langle (h_1,f_1),(h_2,f_2)\rangle_{(\phi,\theta)} = a^2 \int_0^1 h_1(t)h_2(t)e^{\phi(t)}\,dt + b^2 \int_0^1 \langle f_1(t)f_2(t)\rangle e^{\phi(t)}\,dt. \tag{C.12}$$

The first integral in the inner product given above controls the amount of stretching, and the second integral controls the bending. The ratio a/b shows the amount of stretching versus bending, and gives a basis for comparing how metrics elastically deform curves. It can be shown that the L^2 metric on the space of SRV functions correspond to the elastic metric with $a = 1/2$ and $b = 1$. However, using the L^2 metric in the space $L^2([0,1],\mathbb{R}^2)$, and letting f_1 and f_2 be two tangent vectors at a point $q \in L^2([0,1],\mathbb{R}^2)$ we get that

$$\langle f_1, f_2 \rangle = \int_0^1 \langle f_1(t), f_2(t) \rangle \, dt,$$

which gives an efficient way of computing geodesics using the SRV.

C.2.3 Adding the prior energy term

We are now ready to form a new energy term, also taking into account the prior information discussed in the beginning of this section. Let C be the curve describing the active contour, and $[q]$ the corresponding orbit. Let $[\mu]$ be the orbit of the boundary we want to find in the optical flow image. The proposed energy to be minimized is

$$\begin{aligned} E(C) &= E_\varphi(C) + E_{prior}(C) && \text{(C.13)} \\ &= \alpha \int\limits_{\text{int}(C)} (\overline{\varphi}_{int} - \varphi)^2 \, dx + \alpha \int\limits_{\text{ext}(C)} (\overline{\varphi}_{ext} - \varphi)^2 \, dx + \lambda d_s([q],[\mu])^2, && \text{(C.14)} \end{aligned}$$

where λ is some positive constant. Comparing this energy functional with the energy given in 5.2, we see that the length term has been removed, and the energy

$$E_{prior} = \lambda d_s([q],[\mu])^2, \tag{C.15}$$

has been added. We argue that the energy E_{prior} will provide sufficient regularization and that the length term can be dropped.

From section 5.1 we have that

$$\nabla E_\varphi(C) = \alpha \left[(\overline{\varphi}_{int}(\tau) - \varphi)^2 - (\overline{\varphi}_{ext}(\tau) - \varphi)^2 \right] |C'| \boldsymbol{N}, \tag{C.16}$$

where $\boldsymbol{N}$ is the outward unit normal of the curve C. And the gradient $\nabla E(C)$ is given as

$$\nabla E(C) = \nabla E_\varphi(C) + \nabla E_{prior}(C). \tag{C.17}$$

Moreover, $\nabla E_{prior}(C)$ can be computed using C.7 and C.11. Using the gradient descent method to evolve the curve, we set

$$\frac{\partial C}{\partial \tau} = -\nabla E(C). \tag{C.18}$$

C.2.4 The evolving the spline curve

Now, as done in section 5.2, we want to represent the curve by a periodic cubic B-spline. Hence, we want to map the evolution of the curve to the evolution of the spline control points. By using the same procedure as in section 5.2, we are lead to solving the normal equations

$$DC(p)^*\left(DC(p)\frac{\partial p}{\partial \tau} + \nabla E(C)\right) = 0, \tag{C.19}$$

where $DC(p)$ denotes the derivative of the curve with respect to the control points, and $DC(p)^*$ denotes the adjoint of $DC(p)$. The element $\frac{\partial p}{\partial \tau} \in (\mathbb{R}^2)^K$ solving this equation is given by the solution to the system

$$\begin{aligned} A\frac{\partial p}{\partial \tau}^c &= -\left[\int_0^1 \nabla E(C(p))(t)^c \psi^k(t)\,\mathrm{d}t\right]_{k=1}^K && \text{(C.20)} \\ &= \Phi^c, && \text{(C.21)} \end{aligned}$$

where the elements of the matrix A is given in equation (5.12).

The energy E_{prior} is here chosen to be a quadratic penalization of the geodesic distance between the two shapes in the shape space S. However, this is not the only way of defining this energy. Joshi [14] proposed using principal component analysis (PCA) of the observed set of curves on the tangent space $T(\mu)$ defined in (C.5). Let $M \subset T(\mu)$ be this subspace. We then project the active contour onto this subspace, and use an imposed probability density to estimate its energy. We refer to [14] for more details.

By using a unique shape representation called orbits we have constructed a shape space with a distance measure. This has enables us to find a geodesic path between two orbits. Given some collection of shape representations, we can guide our segmentation towards a mean of these shapes, called the prior. This is done by perturbing the shape representation along the direction given by the geodesic path from the current shape to the prior. This perturbation in the shape representation results in a unique perturbation in the parametrized curve C. Incorporating this into a modified version of the Mumford-Shah functional can lead to enhanced segmentation results. A more extensive treatment of shape priors and their gradients can be found in [14]. See also [30] for the theory and methods of the shape space and the computation of the geodesic distance.

D Python code

This chapter presents some of the code used to produce the results in chapters 7 and 8. Section D.1 presents the optical flow code, section D.2 presents the code for the spline curve evolution and section D.3 shows the code for the level set evolution. In each section a short explanation for each class and function is given.

D.1 Optical flow code

The optical flow code consists of the assembly functions shown in section D.1.1, which is a collection of functions used to build the optical flow systems, the Horn and Schunck flow estimation shown in section D.1.2, the image driven flow estimation shown in section D.1.3 and the flow driven function shown in section D.1.4.

D.1.1 Assembly functions

The functions called the assembly functions are functions used to build the optical flow system. One can divide the collection into four parts; the data term methods, the smoothness term methods, the differentiation methods and the boundary method.

The data term methods

MotionTerms	*Function*: constructs the image derivatives used in the data term.
makeQuadraticDataTerm	*Function*: constructs the data term for a quadratic data term penalization.

The smoothness methods

smoothnessHS	*Function*: constructs the HS smoothness term.
diDeriv	*Function*: finds the eigenvectors of the structure matrix S_ρ for a given value of the standard deviation ρ (denoted in the code as *sigma*). The method also returns the trace of the structure matrix.
smoothnessNE	*Function*: constructs the ID smoothness term of Nagel and Enkelmann.
smoothnessFD	*Function*: constructs the FD smoothness term.

The differentiation methods

The differentiation methods are methods for computing the image derivatives. The methods are

forwardDifferenceImage	*Function*: computes the image derivatives using the forward difference.
sobelDerivative	*Function*: computes the image derivatives using a sobel filter.
backwarddDifferenceImage	*Function*: computes the image derivatives using the backward difference.
centralDifferenceImage1	*Function*: computes the image derivatives using the central difference type 1.
centralDifferenceImage2	*Function*: computes the image derivatives using the central difference type 2.
makeLmatrix	*Function*: constructs the $2mn \times 2mn$ differentiation operator for the forward difference.

Only the forward difference is used for the results shown in the thesis.

The boundary methods

Consist of one function, *neumann_boundary*. This function returns a matrix and a vector, used in solving the boundary equation. Solving the boundary equations essentially means setting the boundary rows and columns equal to the closest row and column, respectively.

```python
import numpy as np
from scipy import sparse, ndimage
import math


####  Data term methods #####

def MotionTerms(g1,g2,m,n,diff_method,zeta,normalize):
  g = g1

  if diff_method is 'forward':
      [gx,gy] = forwardDifferenceImage(g,m,n)
      [gxx,gxy] = forwardDifferenceImage(gx,m,n)
      [gyx,gyy] = forwardDifferenceImage(gy,m,n)
      [g2x,g2y] = forwardDifferenceImage(g2,m,n)
  elif diff_method is 'central1':
      [gx, gy] = idm.centralDifferenceImage1(g,m,n)
      [gxx, gxy] = idm.centralDifferenceImage1(gx,m,n)
      [gyx, gyy] = idm.centralDifferenceImage1(gy,m,n)
  elif diff_method is 'central2':
      [gx, gy] = idm.centralDifferenceImage2(g,m,n)
      [gxx, gxy] = idm.centralDifferenceImage2(gx,m,n)
      [gyx, gyy] = idm.centralDifferenceImage2(gy,m,n)
  elif diff_method is 'sobel':
      [gx, gy] = idm.sobelDerivative(g,m,n)
      [gxx, gxy] = idm.sobelDerivative(gx,m,n)
      [gyx, gyy] = idm.sobelDerivative(gy,m,n)

  gt = np.subtract(g2,g1)
  gxt = np.subtract(g2x,gx)
  gyt = np.subtract(g2y,gy)


  # Normalisation terms
  if normalize:
      # Normalisation terms
      theta_0 = np.sqrt(np.power(gx,2) + np.power(gy,2) + zeta**2)
      theta_x = np.sqrt(np.power(gxx,2) + np.power(gxy,2) + zeta**2)
      theta_y = np.sqrt(np.power(gyx,2) + np.power(gyy,2) + zeta**2)
      gt = np.divide(gt,theta_0)
      gxt = np.divide(gxt,theta_x)
      gyt = np.divide(gyt,theta_y)

      gx = sparse.diags(np.divide(gx,theta_0),0,format='csr')
      gy = sparse.diags(np.divide(gy,theta_0),0,format='csr')
      gxx = sparse.diags(np.divide(gxx,theta_x),0,format='csr')
      gyx = sparse.diags(np.divide(gyx,theta_x),0,format='csr')
      gxy = sparse.diags(np.divide(gxy,theta_y),0,format='csr')
      gyy = sparse.diags(np.divide(gyy,theta_y),0,format='csr')
  else:
      gx = sparse.diags(gx,0,format='csr')
      gy = sparse.diags(gy,0,format='csr')
      gxx = sparse.diags(gxx,0,format='csr')
      gyx = sparse.diags(gyx,0,format='csr')
      gxy = sparse.diags(gxy,0,format='csr')
      gyy = sparse.diags(gyy,0,format='csr')


  grad_g = sparse.hstack((gx,gy),format='csr')
  grad_gx = sparse.hstack((gxx,gyx),format='csr')
  grad_gy = sparse.hstack((gxy,gyy),format='csr')

  return grad_g, grad_gx, grad_gy, gt, gxt, gyt

def makeQuadraticDataTerm(g1,g2,m,n,diff_method,zeta,gamma,normalize):
    # Data term for quadratic penalization

    grad_g, grad_gx, grad_gy, gt, gxt, gyt = MotionTerms(g1,g2,m,n,diff_method,zeta,normalize)
    #
    # # Model term
    M = (grad_g.T).dot(grad_g) + gamma*((grad_gx.T).dot(grad_gx) + (grad_gy.T).dot(grad_gy))
    # RHS
    b = - ((grad_g.T).dot(gt) + gamma*((grad_gx.T).dot(gxt) + (grad_gy.T).dot(gyt)))

    return M,b

### Smoothness term methods #####

def smoothnessHS(m,n):
    # Computes the smoothness term of Horn and Schunck.
    # Parameters: m: number of rows in the image
    #             n: number of columns in the image
    # Returns: V: 2mn x 2mn array

    L = makeLmatrix(m,n)
    V = (L.T).dot(L)
    return V

```

```python
def dirDeriv(g,m,n,sigma,eps):
    # Finds the unit vevtors in the direction normal to the image edges and
    # parallel to the image edges (see framework in Optical Flow in Harmony)
    # Parameters: g: image as m-by-n 2-dimensional array
    # Returns: s1: direction normal to image edges
    #          s2: direction parallel to image edges

    k = eps # Small parameter to avoid singular matrices

    [Dx, Dy] = forwardDifferenceImage(g,m,n)
    S11 = np.reshape(np.power(Dx,2),[n,m]).T
    S12 = np.reshape(np.multiply(Dx,Dy),[n,m]).T
    S22 = np.reshape(np.power(Dy,2),[n,m]).T

    S11 = ndimage.filters.gaussian_filter(S11,sigma)
    S11 = np.reshape(S11.T,[1,m*n])[0]
    S12 = ndimage.filters.gaussian_filter(S12,sigma)
    S12 = np.reshape(S12.T,[1,m*n])[0]
    S21 = S12
    S22 = ndimage.filters.gaussian_filter(S22,sigma)
    S22 = np.reshape(S22.T,[1,m*n])[0]

    tmp = np.sqrt(np.power(S11,2)-2*np.multiply(S11,S22)+np.power(S22,2)+4*np.multiply(S21,S12))

    # s1 is the eigenvector corresponding to the largest eigenvalue
    s1_1 = S11 - S22 + tmp
    s1_2 = 2*S21
    norm1 = np.sqrt(np.power(s1_1,2)+np.power(s1_2,2)) + eps
    s1 = sparse.hstack((sparse.diags(np.divide(s1_1,norm1),0),sparse.diags(np.divide(s1_2,norm1),0)),format = 'csr').T

    s2_1 = S11 - S22 - tmp
    s2_2 = 2*S21
    norm2 = np.sqrt(np.power(s2_1,2)+np.power(s2_2,2)) + eps
    s2 = sparse.hstack((sparse.diags(np.divide(s2_1,norm2),0),sparse.diags(np.divide(s2_2,norm2),0)),format = 'csr').T

    trace = S11 + S22

    return s1,s2,trace

def smoothnessNE(g,m,n,kappa,sigma,eps):
    # Computes the anisotropic image driven smoothness term
    # of Nagel and Enkelmann.
    # Parameters: g: image as m-by-n array
    #          kappa: regularization parameter
    # Returns: V: smoothness array

    [s1,s2,trace] = dirDeriv(g,m,n,sigma,eps)
    A = sparse.diags(np.divide(np.power(kappa,2)*np.ones(m*n),trace+2*np.power(kappa,2)*np.ones(m*n)),0)
    A = sparse.kron(sparse.eye(2),A)
    B = sparse.diags(np.divide(trace + np.power(kappa,2)*np.ones(m*n),trace+2*np.power(kappa,2)*np.ones(m*n)),0)
    B = sparse.kron(sparse.eye(2),B)
    P1 = A.dot(s1.dot(s1.T)) + B.dot(s2.dot(s2.T))
    P = sparse.kron(sparse.eye(2),P1,format = 'csr') # Diffusion matrix

    L = makeLmatrix(m,n)

    V = ((L.T).dot(P)).dot(L)
    return V

```

```
def smoothnessFD(grad_w,m,n,eps):
    # Forms the diffusion matrix in the lagged diffusivity iteration
    #            grad_w: flow gradient (derivatives)
    #            m,n: dimensions of image
    #             eps: parameter in the convex penaliser function
    # Returns:   V: Diffusion matrix

    u_x = grad_w[0:m*n]
    u_y = grad_w[m*n:2*m*n]
    v_x = grad_w[2*m*n:3*m*n]
    v_y = grad_w[3*m*n:4*m*n]
    psi_deriv = sparse.diags(np.divide(np.ones(m*n),(np.sqrt(np.power(u_x,2) + np.power(u_y,2) +
 ↪   np.power(v_x,2) + np.power(v_y,2)+math.pow(eps,2)))),0)
    L = makeLmatrix(m,n)
    V = ((L.T).dot(sparse.kron(sparse.eye(4),psi_deriv,format = 'csr'))).dot(L)
    return V


### Differentiation methods ####

def forwardDifferenceImage(g,m,n):
    #forwardDifferenceImage Computes approximation of the image gradient using
    #forward difference
    # Boundaries: zero first derivatives

    Lx = sparse.diags([-np.ones(m*n),np.ones((n-1)*m)],[0,m],format = 'lil')
    Lx[m*(n-1):m*n,m*(n-1):m*n] = np.zeros((m,m))
    Lx[m*(n-1):m*n,m*(n-2):m*(n-1)] = np.zeros((m,m))
    Ly1 = sparse.diags([-np.ones(m), np.ones(m-1)],[0,1],format = 'lil')
    Ly1[m-1,:] = np.hstack((np.zeros(m-2),[0,0]))
    Ly = sparse.kron(sparse.eye(n),Ly1,format = 'csr')

    Dx = Lx.dot(g)
    Dy = Ly.dot(g)

    return Dx,Dy

def sobelDerivative(g,m,n):
    # Computes the Dx and Dy submatrices of the model term discretization.
    # Uses the sobel derivatives as approximation for the image gradients.
    # Parameters: g: an image as array
    # Returns: Dx: m*n 1-dimensional array with the approximation of the
    #                derivatives in the x-direction
    #          Dy: m*n 1-dimensional array with the approximation of the
    #                derivatives in the y-direction

    g = np.reshape(g,[n,m]).T

    Gx = np.array([[-1 ,0, 1],[-2, 0 ,2],[-1, 0, 1]])
    Gy = Gx.T

    Dx = signal.convolve2d(g,Gx,mode='same')
    Dy = signal.convolve2d(g,Gy,mode='same')

    Dx = np.reshape(Dx.T,[1,m*n])[0]
    Dy = np.reshape(Dy.T,[1,m*n])[0]
    return Dx,Dy

def backwardDifferenceImage(g,m,n):
    Lx = sparse.diags([-np.ones(m*n),np.ones((n-1)*m)],[0,m],format = 'lil')
    Lx[m*(n-1):m*n,m*(n-2):m*(n-1)] = np.zeros((m,m))
    Ly1 = sparse.diags([-np.ones(m), np.ones(m-1)],[0,1],format = 'lil')
    Ly1[0,:] = np.hstack((np.zeros(m-2),[0,0]))
    Ly = sparse.kron(sparse.eye(n),Ly1,format = 'csr')

```

```python

    Dx = (-Lx.T).dot(g)
    Dy = (-Ly.T).dot(g)

    return Dx,Dy

def centralDifferenceImage1(g):
    #forwardDifferenceImage Computes approximation of the image gradient using
    # central difference type1
    lx = sparse.hstack((-sparse.eye(m),np.zeros((m,m))))
    lx = sparse.hstack((lx,sparse.eye(m)))
    lx = 1.0/2*lx
    Lx = 1.0/2*sparse.diags([-np.ones(m*(n-1)),np.ones((n-1)*m)],[-m,m],format = 'lil')
    Lx[m*(n-1):m*n,m*(n-3):m*n] = lx
    Lx[0:m,0:m*3] = lx

    ly = np.array([-1.0,0,1.0])/2
    Ly1 = sparse.diags([-np.ones(m-1), np.ones(m-1)],[-1,1],format = 'lil')
    Ly1[m-1:m,m-3:m] = ly
    Ly1[0,0:3] = ly
    Ly = sparse.kron(sparse.eye(n),Ly1,format = 'csr')

    Dx = Lx.dot(g)
    Dy = Ly.dot(g)

    return Dx,Dy

def centralDifferenceImage2(g):
    # Central difference type 2
    lx = sparse.hstack((sparse.eye(m),-8*sparse.eye(m)))
    lx = sparse.hstack((lx,np.zeros((m,m))))
    lx = sparse.hstack((lx,8*sparse.eye(m)))
    lx = sparse.hstack((lx,-sparse.eye(m)))
    Lx = sparse.diags([np.ones((n-2)*m),-8*np.ones((n-1)*m),8*np.ones((n-1)*m),-np.ones((n-2)*m)],[-
  ↪   2*m,-m,m,2*m],format='lil')
    Lx[m*(n-1):m*n,m*(n-5):m*n] = lx
    Lx[0:m,0:m*5] = lx
    Lx = sparse.csr_matrix(Lx)

    ly = np.array([-8.0,1.0,0,8.0,-1.0])
    Ly1 = sparse.diags([np.ones(m-2),-8*np.ones(m-1),8*np.ones(m-1),-np.ones(m-2)],[-2,-1,1,2],format =
  ↪   'lil')
    Ly1[m-1,m-5:m] = ly
    Ly1[0,0:5] = ly
    Ly = sparse.kron(sparse.eye(n),Ly1,format = 'csr')


    Dx = Lx.dot(g)/12
    Dy = Ly.dot(g)/12

    return Dx,Dy

def makeLmatrix(m,n):
    # Forms the L matrix used in the flow derivative approximation.
    # A matrix multipication with Lx and Ly gives the approximation of the
    # derivatives in x- and y-direction assuming neumann boundary conditions.
    # Parameters: m: number of rows in the image
    #             n: number of columns in the image
    # Returns: L: 2-dimensional array of shape 4mn x 2mn. The matrix
    #           multiplication grad_w= L[u,v].T gives a
    #           4mn vector grad_w = [u_x,u_y,v_x,v_y].T

    Lx = sparse.diags([-np.ones(m*n),np.ones((n-1)*m)],[0,m],format = 'lil')
    Ly1 = sparse.diags([-np.ones(m), np.ones(m-1)],[0,1],format = 'lil')
    Ly = sparse.kron(sparse.eye(n),Ly1,format = 'csr')
    L = sparse.kron(sparse.eye(2),sparse.vstack((Lx,Ly)),format = 'csr')
    return L

#### Boundary Methods ####

def neumann_boundary(m,n):
    # Matrices for solving the boundary equations

    neumann_x = sparse.hstack((sparse.diags(-np.ones(m),0),sparse.diags(np.ones(m),0)))
    neumann_y = sparse.eye(m,format='lil')
    neumann_y[0,:2] = [-1,1]
    neumann_y[m-1,m-2:m] = [-1,1]
    neumann = sparse.kron(sparse.eye(n),neumann_y,format = 'lil')
    neumann[0:m,0:2*m] = neumann_x
    neumann[m*(n-1):m*n,m*(n-2):m*n] = neumann_x
    elim_y = np.ones(m)
    elim_y[0] = 0
    elim_y[m-1] = 0
    elimination_vector =
 ↪   np.hstack((np.hstack((np.zeros(m),np.kron(np.ones((n-2)),elim_y))),np.zeros(m)))

    elimination_vector = np.hstack((elimination_vector,elimination_vector))
    neumann = sparse.kron(sparse.eye(2),neumann)

    return neumann, elimination_vector
```

D.1.2 Horn and Schunck method

The *estimateFlow_HS* function estimates the flow using the isotropic data term described as the Horn and Schunck method. The function takes two consecutive images, a global regularization parameter and a standard deviation for the presmoothing. The default values of the optional parameters are set to estimate the flow using the original Horn and Schunck method [13], using only the BCA, but this can be changed by passing the parameters γ and ζ to the function in addition to the required arguments.

```
import numpy as np
from scipy import  ndimage,sparse
from scipy.sparse.linalg import spsolve, gmres
import assemble_flow_systems as afs
import math


def estimateFlow_HS(g1,g2,xi,sigma_image,gamma = 0.0,zeta = None,diff_method='forward',gmres_tol =
 ↪   None):
    # Estimates the flow using isotropic smoothing of Horn and schunck
    # g1 and g2 are two consecutive images represented as numpy arrays
    # xi: global regularization parameter
    # sigma_image: std deviation in Gaussian convolution
    # gamma: GCA parameter
    # zeta: normalization parameter
    # diff_method gives the differentiation method for image derivatives
        # 'forward' gives forward difference
        # 'central1' gives central difference type 1
        # 'central2' gives central difference type 2
        # 'sobel' gives sobel derivative

    [m,n] = g1.shape

    # Gaussian Smoothing
    g1 =  np.array(g1, dtype=np.double)
    g2 = np.array(g2, dtype=np.double)
    g1 = ndimage.filters.gaussian_filter(g1,sigma_image)
    g2 = ndimage.filters.gaussian_filter(g2,sigma_image)
    g1 = g1.flatten(order='F')
    g2 = g2.flatten(order='F')

    if zeta:
        normalize = True
    else:
        normalize = False

    # Using Horn and Schunck Smoothness term
    V = afs.smoothnessHS(m,n)
    M,b = afs.makeQuadraticDataTerm(g1,g2,m,n,diff_method,zeta,gamma,normalize)
    G = M + math.pow(xi,-2)*V

    neumann, elimination_vector = afs.neumann_boundary(m,n)
    if gmres_tol:
        # Interior
        w,info = gmres(G,b,tol=gmres_tol)
        # Boundary
        w, info = gmres(neumann,np.multiply(w,elimination_vector),x0=w,tol=gmres_tol)
    else:
        # Interior
        w = spsolve(G,b)
        # Boundary
        w = spsolve(neumann,np.multiply(w,elimination_vector))

    return w
```

D.1.3 Image driven method

The function *estimateFlow_ID* estimates the flow using the ID regularization term of Nagel and Enkelmann. Similar to the HS method, the function takes a set of required parameters, with a set of optional parameters having default values corresponding to the original method proposed by Nagel and Enkelmann [22].

```
import numpy as np
from scipy import  ndimage,sparse
from scipy.sparse.linalg import spsolve, gmres
import assemble_flow_systems as afs
import math


def estimateFlow_ID(g1,g2,xi,sigma_image,kappa,mu_regTensor,eps=0.001,gamma = 0.0,zeta =
 ↪   None,diff_method='forward',gmres_tol = None):
    # Estimates the flow using Image driven smoothing
    # g1 and g2 are two consecutive images represented as numpy arrays
    # xi: global regularization parameter
    # sigma_image: std deviation in Gaussian convolution
    # kappa: regularization parameter in projection matrix
    # mu_regTensor: std deviation for Gaussian convolution of structure matrix
    # eps: small parameter to avoid singular matrices
    # gamma: GCA parameter
    # zeta: normalization parameter
    # diff_method gives the differentiation method for image derivatives
        # 'forward' gives forward difference
        # 'central1' gives central difference type 1
        # 'central2' gives central difference type 2
        # 'sobel' gives sobel derivative

    [m,n] = g1.shape

    # Gaussian Smoothing
    g1 =  np.array(g1, dtype=np.double)
    g2 = np.array(g2, dtype=np.double)
    g1 = ndimage.filters.gaussian_filter(g1,sigma_image)
    g2 = ndimage.filters.gaussian_filter(g2,sigma_image)
    g1 = g1.flatten(order='F')
    g2 = g2.flatten(order='F')

    if zeta:
        normalize = True
    else:
        normalize = False

    # Using Nagel and Enkelmann image driven method
    V = afs.smoothnessNE(g1,m,n,kappa,mu_regTensor,eps)
    M,b = afs.makeQuadraticDataTerm(g1,g2,m,n,diff_method,zeta,gamma,normalize)
    G = M + math.pow(xi,-2)*V
    G.tocsr()

    neumann, elimination_vector = afs.neumann_boundary(m,n)
    neumann.tocsr()
    if gmres_tol:
        # Interior
        w,info = gmres(G,b,tol=gmres_tol)
        # Boundary
        w, info = gmres(neumann,np.multiply(w,elimination_vector),x0=w,tol=gmres_tol)
    else:
        # Interior
        w = spsolve(G,b)
        # Boundary
        w = spsolve(neumann,np.multiply(w,elimination_vector))
    return w
```

D.1.4 Flow driven method

```
import numpy as np
from scipy import misc, ndimage,sparse, signal
from scipy.sparse.linalg import spsolve, gmres
import math
import assemble_flow_systems as afs

def estimateFlow_FD(g1,g2,xi,sigma_image,eps= 0.01, gamma = 0.0,zeta =
 ↪   None,diff_method='forward',gmres_tol = None):
    # Computes the flow using lagged diffusivity
    # g1 and g2 are two consecutive images represented as numpy arrays
    # xi: global regularization parameter
    # sigma_image: std deviation in Gaussian convolution
    # kappa: regularization parameter in projection matrix
    # mu_regTensor: std deviation for Gaussian convolution of structure matrix
    # eps: small parameter to avoid singular matrices
    # gamma: GCA parameter
    # zeta: normalization parameter
    # diff_method gives the differentiation method for image derivatives
        # 'forward' gives forward difference
        # 'central1' gives central difference type 1
        # 'central2' gives central difference type 2
        # 'sobel' gives sobel derivative

    [m,n] = g1.shape

    # Gaussian Smoothing
    g1 =  np.array(g1, dtype=np.double)
    g2 = np.array(g2, dtype=np.double)
    g1 = ndimage.filters.gaussian_filter(g1,sigma_image)
    g2 = ndimage.filters.gaussian_filter(g2,sigma_image)
    g1 = g1.flatten(order='F')
    g2 = g2.flatten(order='F')

    if zeta:
        normalize = True
    else:
        normalize = False

    L = afs.makeLmatrix(m,n)
    # Initial flow values
    w = np.zeros(2*m*n)
    # Flow derivatives
    grad_w = L.dot(w)
    # Smoothness term
    V = afs.smoothnessFD(grad_w,m,n,eps)
    del_w = 1
    iter_nr = 0
    iter_max = 10

    [M,b] = afs.makeQuadraticDataTerm(g1,g2,m,n,diff_method,zeta,gamma,normalize)
    # Lagged Diffusivity iteration:
    while  np.max(del_w) > 1e-4 and iter_nr <iter_max:
        print iter_nr
        iter_nr += 1
        G = M + math.pow(xi,-2)*V
        neumann, elimination_vector = afs.neumann_boundary(m,n)
        if gmres_tol:
            # Interior
            w_new,info = gmres(G,b,x0=w,tol=gmres_tol)
            # Exterior
            w_new, info = gmres(neumann,np.multiply(w_new,elimination_vector),x0=w_new,tol=gmres_tol)
        else:
            # Interior
            w_new = spsolve(G,b)
            # Boundary
            w_new = spsolve(neumann,np.multiply(w_new,elimination_vector))
        grad_w = L.dot(w_new)
        V = afs.smoothnessFD(grad_w,m,n,eps)
        del_w = abs(w_new - w)
        w = w_new
    return w
```

D.2 Spline curve evolution

The following code contains the functions and classes related to the spline curve. The following is a list of classes and functions in the order they appear in the code;

JordanCurve	*Class*: contains functions for evaluating spline curve

	and its derivatives. The class also contains the functions for finding the boundary, finding the enclosed area, finding self-intersections and drawing functions.
floodfill	*Function*: finds the enclosed area given a boundary f.
split_curve	*Function*: splits curve into two. Returns the number of curves that was added to the collection of closed curves.
makeLagrangianBasis	*Function*: constructs basis coefficients for bilinear basis functions.
evaluate_function	*Function*: computes a bilinear interpolation of a function at a position(x,y) given function weights w.
MS_grad	*Function*: computes the gradient of the modified Mumford-Shah (MS) functional for a given parameter value t, a curve C and a level set function u (not flow component).
MS	*Function*: computes value of the modified Mumford-Shah (MS) functional for a given curve C.
lineSearch	*Function*: performs a backtracking line search given a search direction dp and a curve C. Returns the new control points for the curve C.
evolveControlPoints	*Function*: evolves the spline curve control points for a list of curves. The function takes a list of curves, a level set function and spline parameters.
initializeControlPoints	*Function*: initializes control points as a circle with given center and radius.

```python
import numpy as np
import math
from scipy import misc, ndimage,sparse,integrate
from PIL import Image
import math


class JordanCurve:
    # Class for a simple Jordan Curve
    # Initialize with a set of K control points
    # The control points of the periodic curve need to be in clockwise order
    def __init__(self,points):
        # points: Set of K points in clockwise order
        self.K = len(points)
        self.points = points
        # Makes vector of control points for a periodic curve
        self.control_points = np.vstack((points[self.K-1],np.vstack((points,points[0:2]))))

    def evaluate(self,t):
        # Evaluates the curve at t
        # t lies in the interval [t_k,t_k+1]
        if t == 1.0:
            k = 0
        else:
            k = np.floor(t*self.K).astype(int)
        # Position in vector of control points for point k
        # points[k] = control_points[i] for k = 0,...,K
        i = k + 1
        # Linear combination of the basis functions a, b, c and d
        return self.control_points[i+2]*self.a(t) + self.control_points[i+1]*self.b(t) +
 ↪   self.control_points[i]*self.c(t) + self.control_points[i-1]*self.d(t)

    def evaluate_d(self,t):
        # Evaluates the derivatives
        # t lies in the interval [t_k,t_k+1]
        if t == 1.0:
            k = 0
        else:
            k = np.floor(t*self.K).astype(int)

        # Position in vector of control points for point k
        # points[k] = control_points[i] for k = 0,...,K
        i = k + 1

        # Linear combination of the derivative of basis functions a, b, c and d
        return self.K*(self.control_points[i+2]*self.a_d(t) + self.control_points[i+1]*self.b_d(t) +
 ↪   self.control_points[i]*self.c_d(t) + self.control_points[i-1]*self.d_d(t))

    def evaluate_dd(self,t):
        if t == 1.0:
            k = 0
        else:
            k = np.floor(t*self.K).astype(int)
        i = k + 1
        # Linear combination of the double derivative of basis functions a, b, c and d
        return self.K*self.K*(self.control_points[i+2]*self.a_dd(t) +
 ↪   self.control_points[i+1]*self.b_dd(t) + self.control_points[i]*self.c_dd(t) +
 ↪   self.control_points[i-1]*self.d_dd(t))


    def set_points(self,points):
        # Sets new control points
        # Equivalent to the mapping
        # control_points -> [param -> points_on_curve]
        self.points = points
        self.K = len(points)
        self.control_points = np.vstack((points[self.K-1],np.vstack((points,points[0:2]))))

    def a(self,t):
        # t lies in the interval [t_k,t_k+1]
        k = np.floor(t*self.K).astype(int)
        # Maps [t_k,t_k+1] to [0,1]
        s = (t - k*1.0/self.K)*self.K
        return 1.0/6*np.power(s,3)

    def b(self,t):
        # k = int(t*self.K)
        k = np.floor(t*self.K).astype(int)
        # Maps [t_k,t_k+1] to [0,1]
        s = (t - k*1.0/self.K)*self.K
        return (-3*np.power(s,3) + 3*np.power(s,2) + 3*s + 1)*1.0/6

    def c(self,t):
        # k = int(t*self.K)
        k = np.floor(t*self.K).astype(int)
        # Maps [t_k,t_k+1] to [0,1]
        s = (t - k*1.0/self.K)*self.K
        return 1.0/6*(3*np.power(s,3)-6*np.power(s,2)+4)

    def d(self,t):
        # k = int(t*self.K)
        k = np.floor(t*self.K).astype(int)
        # Maps [t_k,t_k+1] to [0,1]
        s = (t - k*1.0/self.K)*self.K
        return 1.0/6*(-np.power(s,3)+3*np.power(s,2)-3*s+1)

    def a_d(self,t):
        # k = int(t*self.K)
        k = np.floor(t*self.K).astype(int)
        # Maps [t_k,t_k+1] to [0,1]
        s = (t - k*1.0/self.K)*self.K
        return 1.0/2*np.power(s,2)

    def b_d(self,t):
        # k = int(t*self.K)
        k = np.floor(t*self.K).astype(int)
        s = (t - k*1.0/self.K)*self.K
        return (-3*np.power(s,2) + 2*s + 1)*1.0/2

    def c_d(self,t):
        # k = int(t*self.K)
        k = np.floor(t*self.K).astype(int)
        s = (t - 1.0/self.K*k)*self.K
        return 1.0/2*(3*np.power(s,2)-4*s)

    def d_d(self,t):
        # k = int(t*self.K)
        k = np.floor(t*self.K).astype(int)
        s = (t - k*1.0/self.K)*self.K
        return 1.0/2*(-np.power(s,2)+2*s-1)

    def a_dd(self,t):
        # k = int(t*self.K)
        k = np.floor(t*self.K).astype(int)
        s = (t - k*1.0/self.K)*self.K
        return 1.0/6*3*2*s

    def b_dd(self,t):
        # k = int(t*self.K)
        k = np.floor(t*self.K).astype(int)
        s = (t - k*1.0/self.K)*self.K
        return (-3*3*2*s + 3*2)*1.0/6

    def c_dd(self,t):
        # k = int(t*self.K)
        k = np.floor(t*self.K).astype(int)
        s = (t - k*1.0/self.K)*self.K
        return 1.0/6*(3*3*2*s-6*2)

    def d_dd(self,t):
        # k = int(t*self.K)
        k = np.floor(t*self.K).astype(int)
        s = (t - k*1.0/self.K)*self.K
        return 1.0/6*(-3*2*s+3*2)

```

```python
    def draw_curve(self,img,T,rgb):
        dt = 1.0/T
        t = 0
        [m,n,c] = img.shape
        for i in range(T):
            coord = self.evaluate(t)
            if int(coord[0]) >= 0 and int(coord[0]) < m and int(coord[1]) >= 0 and int(coord[1]) < n:
                img[int(coord[0]),int(coord[1])] = rgb
            t = t + dt
        return img

    def draw_control_points(self,img,rgb,m,n):
        for k in range(self.K):
            i = k+1
            p = self.control_points[i]
            if int(p[0]) >= 0 and int(p[0]) < m and int(p[1]) >= 0 and int(p[1]) < n:
                img[int(p[0]),int(p[1])] = rgb
        return img

    def find_next_pixel(self,m,n,t):
        coord = self.evaluate(t)
        k = np.floor(t*self.K).astype(int)
        if (k+1)*1.0/(self.K-1) >= 1.0:
            coord_next = self.evaluate(0.0)
        else:
            coord_next = self.evaluate((k+2)*1.0/(self.K+1))
        cp_next = (k+2)*1.0/(self.K+1)
        if np.power(int(coord[0])-int(coord_next[0]),2) + np.power(int(coord[1])-int(coord_next[1]),2) == 0:
            return cp_next
        t0 = t
        dt = (cp_next-t0)*1.0/2
        t = t0 +dt
        if t >= 1.0:
            return 1.0
        coord_new = self.evaluate(t)
        while np.power(int(coord[0])-int(coord_new[0]),2) + np.power(int(coord[1])-int(coord_new[1]),2) > 2:
            dt = dt*1.0/2
            t = t0 + dt
            coord_new = self.evaluate(t)
        return t

    def find_boundary(self,m,n):
        t = 0.0
        f = np.zeros((m,n))
        coord = self.evaluate(t)
        if int(coord[0]) >= 0 and int(coord[0]) < m and int(coord[1]) >= 0 and int(coord[1]) < n:
            f[int(coord[0]),int(coord[1])] = 1
        while t < 1.0:
            t = self.find_next_pixel(m,n,t)
            coord = self.evaluate(t)
            if int(coord[0]) >= 0 and int(coord[0]) < m and int(coord[1]) >= 0 and int(coord[1]) < n:
                f[int(coord[0]),int(coord[1])] = 1
        return f

    def find_enclosed_area(self,m,n,circumf):
        dt = 1.0/(circumf)
        boundary = self.find_boundary(m,n)
        t = 0.0
        coord_0 = self.evaluate(t)
        while not (int(coord_0[0]) >= 0 and int(coord_0[0]) < m-1 and int(coord_0[1]) >= 0 and int(coord_0[1]) < n-1) and t < 1.0-dt:
            t += dt
            coord_0 = self.evaluate(t)
        C_deriv_0 = self.evaluate_d(t)
        normal_inward = np.array([C_deriv_0[1],-C_deriv_0[0]])*1.0/np.linalg.norm(C_deriv_0)
        # Finding an internal pizel by going in the direction of the inward normal
        while boundary[int(coord_0[0]),int(coord_0[1])] == 1:
            coord_0 = coord_0 + normal_inward
        return floodfill(boundary,int(coord_0[0]),int(coord_0[1]))

    def find_selfintersection(self,m,n):
        self_inter_1 = 0.0
        self_inter_2 = 0.0
        t = 0.0
        f = np.zeros((m,n))
        coord_0 = self.evaluate(t)
        coord = coord_0
        if int(coord_0[0]) >= 0 and int(coord_0[0]) < m and int(coord_0[1]) >= 0 and int(coord_0[1]) < n:
            f[int(coord_0[0]),int(coord_0[1])] = t
        while t < 1.0:
            t = self.find_next_pixel(m,n,t)
            coord_new = self.evaluate(t)
            if int(coord_new[0]) >= 0 and int(coord_new[0]) < m and int(coord_new[1]) >= 0 and int(coord_new[1]) < n:
                if f[int(coord_new[0]),int(coord_new[1])] != 0:
                    if not (int(coord_0[0]) == int(coord_new[0]) and int(coord_0[1]) == int(coord_new[1])):
                        self_inter_1 = f[int(coord_new[0]),int(coord_new[1])]
                        self_inter_2 = t
                        break
                    else:
                        coord = coord_new
                        candidate_t = t
                        while t < 1.0:
                            t =  self.find_next_pixel(m,n,t)
                            coord_new = self.evaluate(t)
                            if not (int(coord[0]) == int(coord_new[0]) and int(coord[1]) == int(coord_new[1])):
                                self_inter_1 = f[int(coord_new[0]),int(coord_new[1])]
                                self_inter_2 = candidate_t
                                break
            if not (int(coord[0]) == int(coord_new[0]) and int(coord[1]) == int(coord_new[1])):
                if int(coord[0]) >= 0 and int(coord[0]) < m and int(coord[1]) >= 0 and int(coord[1]) < n:
                    f[int(coord[0]),int(coord[1])] = t
                    coord = coord_new
        return self_inter_1,self_inter_2

def floodfill(f,x,y):
    [m,n] = f.shape
    toFill = set()
    toFill.add((x,y))
    while len(toFill) > 0:
        (x,y) = toFill.pop()
        if not f[x][y] == 0:
            continue
        f[x][y] = 1
        if x > 0:
            toFill.add((x-1,y))
        if x < m-1:
            toFill.add((x+1,y))
        if y > 0:
            toFill.add((x,y-1))
        if y < n-1:
            toFill.add((x,y+1))
    return f
```

```python
def split_curve(closed_curves,j,m,n):
    C = closed_curves[j]
    t1,t2 = C.find_selfintersection(m,n)
    if t2 == 0.0:
        return 0
    k1 = int(t1*C.K)
    k2 = int(t2*C.K)
    split = 0
    if np.absolute(k2 - k1) > 1:
        C1 = 0
        C2 = 0
        if np.absolute(k2 - k1) > 5:
            new_points1 = C.points[k1:k2]
            C1 = JordanCurve(new_points1)
        if C.K - np.absolute(k2-k1) > 5:
            new_points2 = np.vstack((C.points[:k1],C.points[k2+1:C.K]))
            C2 = JordanCurve(new_points2)
        if C1 != 0 or C2 !=0:
            del closed_curves[j]
            split = split - 1
            if C1 != 0:
                closed_curves.append(C1)
                split = split + 1
            if C2 != 0:
                split = split + 1
                closed_curves.append(C2)
    return split

def makeLagrangianBasis():

    # Returns a 4x4 matrix with the 4 basis coefficients of the 4 bilinear basis
    # functions on the square [-1,1]x[-1,1]
    # The coefficients are ordered as follows:
    # b_n(x,y) = basis[n,0]xy + basis[n,1]y + basis[n,2]x + basis[n,3]

    basis = np.zeros((4,4))

    # South West
    basis[0,0] = 1.0
    basis[0,1] = -1.0
    basis[0,2] = -1.0
    basis[0,3] = 1.0

    # North West
    basis[1,0] = -1.0
    basis[1,1] = 1.0
    basis[1,2] = -1.0
    basis[1,3] = 1.0

    # South East
    basis[2,0] = -1.0
    basis[2,1] = -1.0
    basis[2,2] = 1.0
    basis[2,3] = 1.0

    # North East
    basis[3,0] = 1.0
    basis[3,1] = 1.0
    basis[3,2] = 1.0
    basis[3,3] = 1.0

    basis = 1.0/4*basis

    return basis

```

```python

def evaluate_function(x,y,w,m,n):
    # Evaluates a function in point (x,y) using bilinear basis functions
    # Discrete function values are given in the vector w of length m x n
    # m x n is the size of the domain (image)

    basis = makeLagrangianBasis()

    N = m*n


    i = int(x)
    j = int(y)


    x_i = i
    y_j = j

    if i < 0:
        i = 0
    elif i > m-2:
        i = m-2
    if j < 0:
        j = 0
    elif j> n-2:
        j = n-2

    indices = [j*m+i,(j+1)*m+i,j*m+i+1,(j+1)*m+i+1]

    for ind in indices:
        assert ind > 0 and ind < N, "index " + str(ind) + " outside domain"

    q = 2*(x-x_i)-1
    z = 2*(y-y_j)-1

    return np.dot(np.array([basis[:,0]*q*z + basis[:,1]*z + basis[:,2]*q + basis[:,3]]),w[indices])



def MS_grad(t,C,alpha,beta,m,n,u1,u2,u,eps):
    # Gradient of the Mumford-Shah energy functional
    # t is the current time
    # C is a JordanCurve

    # Computes the coordinate [x,y]
    coord = C.evaluate(t)

    # Computes the derivative
    C_d = C.evaluate_d(t)

    # Computes the outward normal |C'|n (not unit vector)
    normal = np.array([-C_d[1],C_d[0]])

    # Norm of derivative
    C_d_norm = np.linalg.norm(C_d)

    # Computes double derivative
    C_dd = C.evaluate_dd(t)

    if coord[0] < 0 or coord[0]>m-1 or coord[1] < 0 or coord[1] > n-1:
        return -(beta*C_d_norm*C_dd)

    return -(alpha*(np.power(u2-evaluate_function(coord[0],coord[1],u,m,n),2) -
 ↪    np.power(u1-evaluate_function(coord[0],coord[1],u,m,n),2))*normal + beta*C_d_norm*C_dd)

```

```python
def MS(C,alpha,beta,m,n,u):
    # Computes the MS energy
    enclosed = C.find_enclosed_area(m,n,2*np.sqrt(np.power(m,2)+np.power(n,2))).flatten('F')
    enclosed_pixels = sum(enclosed)
    u_J = u*enclosed
    u_0 = u*(enclosed==0)
    assert enclosed_pixels>0, "Empty interior"
    assert enclosed_pixels < m*n, "Empty exterior"
    u_1_int =  integrate.simps(integrate.simps(np.reshape(u_J,(m,n),order= 'F')))/enclosed_pixels
    u_2_int = integrate.simps(integrate.simps(np.reshape(u_0,(m,n),order= 'F')))/(m*n-enclosed_pixels)
    beta_term = 0.0
    alpha_term = integrate.simps(integrate.simps(np.power(np.reshape(u_J,(m,n),order= 'F') -
 ↪   np.reshape(enclosed,(m,n),order='F')*u_1_int,2))) +
 ↪   integrate.simps(integrate.simps(np.power(np.reshape(u_0,(m,n),order= 'F') -
 ↪   np.reshape(enclosed==0,(m,n),order='F')*u_2_int,2)))
    t = 0.0
    dt = 1.0/1000
    while t < 1.0:
        C_d = C.evaluate_d(t)
        C_d_norm = np.linalg.norm(C_d)
        beta_term += C_d_norm*dt
        t += dt
    I = alpha*alpha_term + beta*beta_term
    return I


def lineSearch(dp,C,armijo,u,m,n,alpha,beta):
    # Line Search
    # dp: Search direction
    p0 = C.control_points[1:C.K+1].flatten('F')
    c = 1.0
    # Step reduction parameter
    step_red = 0.5
    # Some initial step size
    step_size = 5.0
    new_points = p0 + step_size*dp
    C_new = JordanCurve(new_points.reshape(C.K,2,order='F'))
    cf =  MS(C,alpha,beta,m,n,u)
    cf_new =  MS(C_new,alpha,beta,m,n,u)
    while (cf_new - cf) > step_size*c*armijo and step_size > 10-6:
        step_size = step_size*step_red
        new_points = p0 + step_size*dp
        C_new = JordanCurve(new_points.reshape(C.K,2,order='F'))
        cf_new = MS(C_new,alpha,beta,m,n,u)
    return new_points.reshape(C.K,2,order='F'),np.absolute(cf_new-cf)

def evolveControlPoints(closed_curves,delt,alpha,beta,u,m,n,lSearch,eps):
    # Method for evolving Control points
    # closed_curves is a list of JordanCurve objects
    # delt is the step length
    # alpha and beta are parameters in the MS energy functional
    # u is the level set function
    # m x n is the image size
    # lSearch: boolean. True -> lineSearch
    # eps parameter to avoid division by zero

    dp_norm_max = 0.0
    N = m*n

    for j in range(len(closed_curves)):
        C = closed_curves[j]
        # Number of intervals between each control point
        T = 20

        # Enclosed area
        enclosed = C.find_enclosed_area(m,n,np.sqrt(np.power(m,2)+np.power(n,2)))
        enclosed = enclosed.flatten('F')
        # Number of enclosed pixels
        enclosed_pixels = sum(enclosed)
        # Interior level set values
        u_J = u*enclosed
        # Exterior level set values
        u_0 = u*(enclosed==0)

        # Averages in the interior and exterior
        u1 =  scipy.integrate.simps(scipy.integrate.simps(np.reshape(u_J,(m,n),order=
 ↪   'F')))/enclosed_pixels
        u2 =  scipy.integrate.simps(scipy.integrate.simps(np.reshape(u_0,(m,n),order=
 ↪   'F')))/(m*n-enclosed_pixels)

        # Padded vector of control points
        p = C.control_points

        # constant in the A-matrix
        sc = 1.0/36*1.0/C.K

        # Nonzero values in the circulant matrix
        A_d0 = 604.0/35*sc
        A_d1 = 1191.0/140*sc
        A_d2 = 6.0/7*sc
        A_d3 = 1.0/140*sc

        # Create the circulant matrix
        circulant = np.zeros(C.K)
        circulant[0] =  A_d0
        circulant[1] =  A_d1
        circulant[2] =  A_d2
        circulant[3] =  A_d3
        circulant[C.K-3] =  A_d3
        circulant[C.K-2] =  A_d2
        circulant[C.K-1] =  A_d1

        # The right hand side of the evolution
        evolution_vector = integrate_Trap(C,T,alpha,beta,m,n,u1,u2,u,eps)

        if lSearch:
            # Linesearch
            # Solve system
            dp = scipy.linalg.solve_circulant(circulant,evolution_vector)
            dp = dp.flatten('F')
            dp_norm = np.linalg.norm(dp)
            unit_step = dp*1.0/(dp_norm)
            armijo = -(unit_step).dot(dp)
            new_points,del_MS = lineSearch(unit_step,C,armijo,u,m,n,alpha,beta)
        else:
            # Solve system
            dp = scipy.linalg.solve_circulant(circulant,evolution_vector)
            dp_norm = np.linalg.norm(dp)
            new_points = p[1:C.K+1] + delt*dp.reshape(C.K,2,order='F')

        # Set new points
        C.set_points(new_points)

        if dp_norm > dp_norm_max:
            dp_norm_max = dp_norm

        # Check for self intersections and tries to split curve
        split = split_curve(closed_curves,j,m,n)
        j = j - split

```

```
    return dp_norm_max


def integrate_Trap(C,T,alpha,beta,m,n,u1,u2,u,eps):
    coeff_vec = np.zeros((C.K+3,2))
    # Iterate over elements [t_k,t_k+1]
    for k in range(C.K):
        # Position in vector of control points
        i = k + 1
        # t_k
        t0 = k*1.0/C.K
        # t_k+1
        t1 = (k+1)*1.0/C.K
        # Time step for integration
        dt = (t1-t0)*1.0/T
        # Start at t_k
        t = t0

        for integration in range(T):
            grad = MS_grad(t,C,alpha,beta,m,n,u1,u2,u,eps)
            # Computes the contribution to basis function k - 1
            coeff_vec[i-1] = coeff_vec[i-1] - dt*grad*(C.d(t))
            # Computes the contribution to basis function k
            coeff_vec[i] = coeff_vec[i] - dt*grad*(C.c(t))
            # Computes the contribution to basis function k +1
            coeff_vec[i+1] = coeff_vec[i+1] - dt*grad*(C.b(t))
            # Computes the contribution to basis function k + 2
            coeff_vec[i+2] = coeff_vec[i+2] - dt*grad*(C.a(t))
            t = t + dt

    # This is DC(p)* o deriv(I(C))(t)
    evolution_vector = coeff_vec[1:C.K+1]
    evolution_vector[C.K-1] = evolution_vector[C.K-1] + coeff_vec[0]
    evolution_vector[0:2] = evolution_vector[0:2] + coeff_vec[-2:]

    return evolution_vector

def initializeControlPoints(n_points,r,center):
    # Clockwise initialization of control points
    control_points = []
    for t in range(n_points):
        theta = -t*1.0/(n_points)*2*math.pi
        control_points.append([r[0]*math.cos(theta) + center[0],r[1]*math.sin(theta) + center[1]])
    control_points = np.array(control_points)
    return control_points
```

D.3 Level set function evolution

The following code contains the functions related to the level set function. The following is a list of functions in the order they appear in the code;

assemble_system_LS	*Function*: assembles the matrix on the left hand side and the vector on the right hand side for the level set system.
intializeLevelSet	*Function*: initializes the level set function given a center and a radius.

neumann_boundary *Function*: assembles a matrix and a vector for the solution of the boundary equations,

```
import numpy as np
from scipy import misc, ndimage,sparse
from scipy.sparse.linalg import spsolve
from PIL import Image

def assemble_system_LS(levelSet,flow_vec,m,n,delt,gamma,eps,eta):

    # Buillds the system for evolving level set function
    # The first order derivatives are approximated by forward difference

    # Forward difference matrices in x- and y-direction
    Lx = sparse.diags([-np.ones(m*n),np.ones((n-1)*m)],[0,m],format = 'lil')
    Lx[0:m,:] = np.zeros((m,m*n))
    Lx[m*(n-1):m*n,:] = np.zeros((m,m*n))
    Ly1 = sparse.diags([-np.ones(m), np.ones(m-1)],[0,1],format = 'lil')
    Ly1[0,:] = np.zeros((1,m))
    Ly1[m-1,:] = np.zeros((1,m))
    Ly = sparse.kron(sparse.eye(n),Ly1,format = 'csr')



    # The flow componens in x- and y-direction
    flow_x = flow_vec[0:m*n]
    flow_y = flow_vec[m*n:2*m*n]

    # Flow derivatives
    flow_x_dx = Lx.dot(flow_x)
    flow_y_dx = Lx.dot(flow_y)
    flow_x_dy = Ly.dot(flow_x)
    flow_y_dy = Ly.dot(flow_y)

    # Derivatives of level set function
    levelSetx = Lx.dot(levelSet)
    levelSety = Ly.dot(levelSet)

    # Edge detector for flow boundaries
    R = np.sqrt(np.power(flow_y_dx,2) + np.power(flow_y_dy,2)) +  np.sqrt(np.power(flow_x_dx,2) +
 ↪   np.power(flow_x_dy,2))
    g = np.divide(1.0,(1.0+eta*(R)))

    # Derivative operator
    L = sparse.vstack((Lx,Ly))

    # Norm of gradient of level set function
    grad_levelSet_norm = np.sqrt(np.power(levelSetx,2) + np.power(levelSety,2)+eps)
    # Diffusion matrix for the evolution
    diffusion_matrix = sparse.kron(sparse.eye(2),sparse.diags(np.divide(g,grad_levelSet_norm),0))
    grad_levelSet_norm_mat = sparse.diags(grad_levelSet_norm,0,format ='csr')

    # Matrix on the left side
    A = sparse.eye(m*n) + delt*grad_levelSet_norm_mat.dot(L.T.dot(diffusion_matrix.dot(L)))
    # Right hand side
    b = delt*gamma*grad_levelSet_norm_mat.dot(g) + levelSet

    return A,b

def intializeLevelSet(center,r,m,n,scaling,ls_type):
    # Initializes the level set
    # ls_type = 'Signed Distance' gives the normal signed distance function
    # to a circle with a given center and radius

    sigma = r*1.0/20
    # Level Set initialization
    levelSet = np.zeros((m,n))
    if ls_type == 'Signed Distance':
        for i in range(m):
            for j in range(n):
                levelSet[i,j] = scaling*(-np.sqrt(np.power(i-center[0],2) + np.power(j-center[1],2)) + r)
        return levelSet.flatten('F')
    if ls_type == 'Circle Hat':
        for i in range(m):
            for j in range(n):
                if np.sqrt(np.power(i-center[0],2) + np.power(j-center[1],2)) < np.linalg.norm(r):
                    levelSet[i,j] = scaling
        levelSet = ndimage.filters.gaussian_filter(levelSet,sigma)
        return levelSet.flatten('F')
    if ls_type == 'Square Hat':
        for i in range(int(center[0])-int(np.linalg.norm(r)),int(center[0])+int(np.linalg.norm(r))):
            for j in range(int(center[1])-int(np.linalg.norm(r)),int(center[1])+int(np.linalg.norm(r))):
                    levelSet[i,j] = scaling
        levelSet = ndimage.filters.gaussian_filter(levelSet,sigma)
        return levelSet.flatten('F')
    if ls_type == 'Signed Distance Exterior Zero':
        for i in range(m):
            for j in range(n):
                if np.sqrt(np.power(i-center[0],2) + np.power(j-center[1],2)) < np.linalg.norm(r):
                    levelSet[i,j] = scaling*(-np.sqrt(np.power(i-center[0],2) + np.power(j-center[1],2)) + np.linalg.norm(r))
        return levelSet.flatten('F')

def neumann_boundary(m,n):
    # Matrices for solving the boundary equations

    neumann_x = sparse.hstack((sparse.diags(-np.ones(m),0),sparse.diags(np.ones(m),0)))
    neumann_y = sparse.eye(m,format='lil')
    neumann_y[0,:2] = [-1,1]
    neumann_y[m-1,m-2:m] = [-1,1]
    neumann = sparse.kron(sparse.eye(n),neumann_y,format = 'lil')
    neumann[0:m,0:2*m] = neumann_x
    neumann[m*(n-1):m*n,m*(n-2):m*n] = neumann_x
    elim_y = np.ones(m)
    elim_y[0] = 0
    elim_y[m-1] = 0
    elimination_vector = np.hstack((np.hstack((np.zeros(m),np.kron(np.ones((n-2)),elim_y))),np.zeros(m)))

    return neumann, elimination_vector
```

Bibliography

[1] L. Alvarez, P.-L. Lions, and J.-M. Morel. “Image Selective Smoothing and Edge Detection by Nonlinear Diffusion. II”. In: *SIAM Journal on Numerical Analysis* 29.3 (1992), pp. 845–866. doi: 10.1137/0729052. eprint: http://dx.doi.org/10.1137/0729052 (cit. on pp. 32, 46).

[2] G. Aubert and P. Kornprobst. *Mathematical Problems in Image Processing: Partial Differential Equations and the Calculus of Variations*. 2nd. Springer Publishing Company, Incorporated, 2010. isbn: 1441921826, 9781441921826 (cit. on pp. 27, 36, 47, 85).

[3] M. Bauer, M. Bruveris, S. Marsland, and P. W. Michor. “Constructing reparameterization invariant metrics on spaces of plane curves”. In: *Differential Geometry and its Applications* 34 (2014), pp. 139–165. issn: 0926-2245. doi: 10.1016/j.difgeo.2014.04.008 (cit. on p. 103).

[4] M. J. Black and P. Anandan. “The Robust Estimation of Multiple Motions: Parametric and Piecewise-Smooth Flow Fields”. In: *Computer Vision and Image Understanding* 63.1 (1996), pp. 75–104. issn: 1077-3142. doi: 10.1006/cviu.1996.0006 (cit. on p. 16).

[5] T. Brox, A. Bruhn, N. Papenberg, and J. Weickert. “High Accuracy Optical Flow Estimation Based on a Theory for Warping”. In: *Computer Vision - ECCV 2004: 8th European Conference on Computer Vision, Prague, Czech Republic, May 11-14, 2004. Proceedings, Part IV*. Ed. by T. Pajdla and J. Matas. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 25–36. isbn: 978-3-540-24673-2. doi: 10.1007/978-3-540-24673-2_3 (cit. on pp. 12, 13).

[6] A. Bruhn, J. Weickert, C. Feddern, T. Kohlberger, and C. Schnörr. “Real-Time Optic Flow Computation with Variational Methods”. In: *Computer Analysis of Images and Patterns: 10th International Conference, CAIP 2003, Groningen, The Netherlands, August 25-27, 2003. Proceedings*. Ed. by N. Petkov and M. A. Westenberg. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 222–229. isbn: 978-3-540-45179-2. doi: 10.1007/978-3-540-45179-2_28 (cit. on p. 86).

[7] V. Caselles, R. Kimmel, and G. Sapiro. “Geodesic Active Contours”. In: *International Journal of Computer Vision* 22.1 (1997), pp. 61–79. issn: 1573-1405. doi: 10.1023/A:1007979827043 (cit. on pp. 2, 8, 25–27, 31, 32, 46, 48).

[8] V. Caselles, F. Catté, T. Coll, and F. Dibos.
"A geometric model for active contours in image processing".
In: *Numerische Mathematik* 66.1 (1993), pp. 1–31. issn: 0945-3245.
doi: 10.1007/BF01385685 (cit. on p. 48).

[9] T. F. Chan and L. A. Vese. "Active Contours Without Edges".
In: *Trans. Img. Proc.* 10.2 (Feb. 2001), pp. 266–277. issn: 1057-7149.
doi: 10.1109/83.902291 (cit. on pp. 35, 54).

[10] I. Cohen.
"Nonlinear Variational Method for Optical Flow Computation".
In: *Proceedings of the 8th Scandinavian Conference on Image Analysis*.
IAPR. Tromso, Norway, Norway, 1993, pp. 523–530 (cit. on p. 20).

[11] M. Crandall, H. Ishii, P. Lions, and A. M. Society. *User's Guide to Viscosity Solutions of Second Order Partial Differential Equations*.
American Mathematical Society, 1992 (cit. on p. 32).

[12] M. Fuchs, B. Jüttler, O. Scherzer, and H. Yang.
"Combined evolution of level sets and B-spline curves for imaging".
In: *Computing and Visualization in Science* 12.6 (2009), pp. 287–295.
issn: 1433-0369. doi: 10.1007/s00791-008-0110-4
(cit. on pp. 2, 32, 35, 45–47, 91).

[13] B. K. Horn and B. G. Schunck. "Determining optical flow".
In: *Artificial Intelligence* 17.1 (1981), pp. 185–203. issn: 0004-3702.
doi: 10.1016/0004-3702(81)90024-2
(cit. on pp. 3, 7, 11, 12, 16, 91, 115).

[14] S. H. Joshi. "Inferences in Shape Spaces with Applications to Image Analysis and Computer Vision".
PhD thesis. Florida State University, 2007 (cit. on pp. 104, 107).

[15] M. Kass, A. Witkin, and D. Terzopoulos.
"Snakes: Active contour models".
In: *International Journal of Computer Vision* 1.4 (1988), pp. 321–331.
issn: 1573-1405. doi: 10.1007/BF00133570 (cit. on pp. 2, 7, 25).

[16] D. G. Kendall. "Shape manifolds, Procrustean metrics, and complex projective spaces".
In: *Bulletin of the London Mathematical Society* (1984) (cit. on p. 101).

[17] X. S. Li. "An Overview of SuperLU: Algorithms, Implementation, and User Interface". In: *ACM Transactions on Mathematical Software* 31.3 (Sept. 2005), pp. 302–325 (cit. on p. 86).

[18] B. D. Lucas and T. Kanade. "An Iterative Image Registration Technique with an Application to Stereo Vision". In: *Proceedings of the 7th International Joint Conference on Artificial Intelligence - Volume 2*.
IJCAI'81. Morgan Kaufmann Publishers Inc., 1981, pp. 674–679
(cit. on p. 86).

[19] W. Mio, A. Srivastava, and S. Joshi.
"On Shape of Plane Elastic Curves".
In: *International Journal of Computer Vision* 73.3 (2007), pp. 307–324.
issn: 1573-1405. doi: 10.1007/s11263-006-9968-0 (cit. on p. 105).

[20] J. M. Morel and S. Solimini.
Variational Methods in Image Segmentation.
Cambridge, MA, USA: Birkhauser Boston Inc., 1995.
isbn: 0-8176-3720-6 (cit. on p. 36).

[21] D. Mumford. "Optimal approximation by piecewise smooth functions and associated variational problems".
In: *Commun. Pure Applied Mathematics* (1989), pp. 577–685
(cit. on pp. 35, 36).

[22] H. H. Nagel and W Enkelmann.
"An Investigation of Smoothness Constraints for the Estimation of Displacement Vector Fields from Image Sequences".
In: *IEEE Trans. Pattern Anal. Mach. Intell.* 8.5 (May 1986), pp. 565–593.
issn: 0162-8828. doi: 10.1109/TPAMI.1986.4767833
(cit. on pp. 3, 18, 91, 116).

[23] J. Nocedal and S. J. Wright. *Numerical Optimization*. 2nd.
New York: Springer, 2006 (cit. on p. 56).

[24] S. Osher and J. A. Sethian.
"Fronts Propagating with Curvature-dependent Speed: Algorithms Based on Hamilton-Jacobi Formulations".
In: *J. Comput. Phys.* 79.1 (Nov. 1988), pp. 12–49. issn: 0021-9991.
doi: 10.1016/0021-9991(88)90002-2 (cit. on pp. 2, 31).

[25] L. I. Rudin, S. Osher, and E. Fatemi.
"Nonlinear total variation based noise removal algorithms".
In: *Physica D: Nonlinear Phenomena* 60.1 (1992), pp. 259–268.
issn: 0167-2789. doi: 10.1016/0167-2789(92)90242-F
(cit. on p. 20).

[26] Y. Saad. *Iterative Methods for Sparse Linear Systems*. 2nd. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 2003.
isbn: 0898715342 (cit. on p. 45).

[27] J. A. Sethian. "Numerical Methods for Propagating Fronts". In:
Variational Methods for Free Surface Interfaces: Proceedings of a Conference Held at Vallombrosa Center, Menlo Park, California, September 7–12, 1985. Ed. by P. Concus and R. Finn.
New York, NY: Springer New York, 1987, pp. 155–164.
isbn: 978-1-4612-4656-5. doi: 10.1007/978-1-4612-4656-5_18
(cit. on p. 76).

[28] J. A. Sethian. “Theory, algorithms, and applications of level set methods for propagating interfaces”. In: *Acta Numerica* 5 (1996), pp. 309–395. issn: 0962-4929 (print), 1474-0508 (electronic). doi: http://dx.doi.org/10.1017/S0962492900002671 (cit. on pp. 2, 31, 76).

[29] D. Shulman and J. Y. Herve. “Regularization of discontinuous flow fields”. In: *Visual Motion, 1989.,Proceedings. Workshop on*. Mar. 1989, pp. 81–86. doi: 10.1109/WVM.1989.47097 (cit. on pp. 3, 20, 91).

[30] A. Srivastava, E. Klassen, S. H. Joshi, and I. H. Jermyn. “Shape Analysis of Elastic Curves in Euclidean Spaces”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 33.7 (2011), pp. 1415–1428. issn: 0162-8828. doi: 10.1109/TPAMI.2010.184 (cit. on pp. 101–104, 107).

[31] D. Sun. *Flow code - MATLAB*. http://vision.middlebury.edu/flow/data/. 2016 (cit. on p. 57).

[32] D. Sun, S. Roth, J. P. Lewis, and M. J. Black. “Computer Vision – ECCV 2008: 10th European Conference on Computer Vision, Marseille, France, October 12-18, 2008, Proceedings, Part III”. In: ed. by D. Forsyth, P. Torr, and A. Zisserman. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008. Chap. Learning Optical Flow, pp. 83–97. isbn: 978-3-540-88690-7. doi: 10.1007/978-3-540-88690-7_7 (cit. on p. 89).

[33] E. J. Velsvik. “Estimating Optical Flow by Variational Methods”. Unpublished Manuscript. 2016 (cit. on p. 89).

[34] C. R. Vogel and M. E. Oman. “Fast, robust total variation-based reconstruction of noisy, blurred images”. In: *IEEE Transactions on Image Processing* 7.6 (June 1998), pp. 813–824. issn: 1057-7149. doi: 10.1109/83.679423 (cit. on p. 22).

[35] H. Zimmer, A. Bruhn, and J. Weickert. “Optic flow in harmony”. In: *Int. J. Comput. Vis.* 93.3 (2011), pp. 368–388. issn: 0920-5691. doi: 10.1007/s11263-011-0422-6 (cit. on pp. 14, 18, 89).