March 2015

# Long Range Motion Estimation and Applications

Laura Sevilla-Lara
*University of Massachusetts Amherst*

# LONG RANGE MOTION ESTIMATION AND APPLICATIONS

A Dissertation Presented

by

LAURA SEVILLA-LARA

Submitted to the Graduate School of the
University of Massachusetts Amherst in partial fulfillment
of the requirements for the degree of

DOCTOR OF PHILOSOPHY

February 2015

School of Computer Science

# LONG RANGE MOTION ESTIMATION AND APPLICATIONS

A Dissertation Presented

by

LAURA SEVILLA-LARA

Approved as to style and content by:

Erik Learned-Miller, Chair

Michael J. Black, Member

Andrew Cohen, Member

Allen Hanson, Member

Ben Marlin, Member

Lori Clarke, Department Chair
School of Computer Science

*To my parents.*

# ACKNOWLEDGMENTS

I would like to thank my advisor Erik Learned-Miller for invaluable help during this process: for trusting me to come, pushing me to start, and giving me freedom and support towards the end. I would also like to thank Michael J. Black, who has been a wonderful collaborator, mentor, friend and source of inspiration over the years. I would like to thank the members of my committee Allen Hanson, Andrew Cohen and Benjamin Marlin for their input and help. I would like to thank my collaborators from Adobe Research, Kalyan Sunkavalli and especially Eli Shechtman, who hosted me and helped me during my last year. I thank Deqing Sun for being a great collaborator and teacher. I want to thank the rest of the faculty I have had the chance to learn from: Andrew McCallum, James Allan, Hanna Wallach and Sridhar Mahadevan. I'd like to thank the members of the Vision Lab at UMass, who have been an invaluable source of knowledge and support: Jacqueline Field, Marwan Mattar, Andrew Kae, David L. Smith, Manjunath Narayana, Gary Huang and Benjamin Mears. I could not wish for better company on this adventure. I also want to thank the great group at Max Planck in Tübingen, Jonas Wulff, Silvia Zuffi, Aggeliki Tsoli, Martin Kiefel, Cristina Garcia, Naejin Kong, Javier Romero, Gerard Pons-Moll, Naureen Mahmood, Matthew Loper, Thomas Nestmeyer, Andreas Lehrmann, Chaohui Wang, Eric Rachlin, Jessica Purmort, David Hirschberg, and Fatma Gutney. I also want to thank other computer vision colleagues who have helped and encouraged me through the process: Fabio Viola, David Balduzzi, Mohammed Rastagari, Nima Kalantari, Antonio Torralba and Alyosha Efros. I want to thank the administrative people who have made it all happen, Melanie Feldhofer, Kristine Watson, and especially Leeanne Leclerc, for being my bureaucratic guardian angel and a great friend.

I want to thank my friends from Spain for staying with me through the distance and years, keeping me grounded: Mercedes Briones, Angel Luis Martinez, Jose Javier Guerrero, Marta Fernandez, Lourdes Alameda, Alfredo Martinez, Armando Morcillo, Maria del Mar Muñoz and Juan Martinez. I want to thank my friends from Amherst, for making this journey so enjoyable: Marc Cartright, Ilene Magpiong, Dirk Ruiken, Maria Turrero, Borja de Cossio, Bruno Ribeiro, Philip Thomas, Scott Niekum, Rachael Singer, Jae-Hyun Park, Bobby, Shiraj Sen, Lawrence de Geest and George Konidaris. I want to thank my roommates for being a family to me: Rodrigo Gramajo, Marie Dieres, Gajin Jeong, Joseph Meiring and specially Katerina Marazopoulou who has been incredible to me.

Finally I want to thank my family, for being the source of craziness, love and enlightenment that any human could wish for: my sisters, brothers-in-law and nieces and nephews. And finally, I want to thank my parents, for being curious, fearless, humble and so loving to me.

# ABSTRACT

## LONG RANGE MOTION ESTIMATION AND APPLICATIONS

FEBRUARY 2015

LAURA SEVILLA-LARA

B.Sc., UNIVERSITY OF GRANADA

M.Sc., BROWN UNIVERSITY

Ph.D., UNIVERSITY OF MASSACHUSETTS AMHERST

Directed by: Professor Erik Learned-Miller

Finding correspondences between images underlies many computer vision problems, such as optical flow, tracking, stereovision and alignment. Finding these correspondences involves formulating a matching function and optimizing it. This optimization process is often gradient descent, which avoids exhaustive search, but relies on the assumption of being in the basin of attraction of the right local minimum. This is often the case when the displacement is small, and current methods obtain very accurate results for small motions.

However, when the motion is large and the matching function is bumpy this assumption is less likely to be true. One traditional way of avoiding this abruptness is to smooth the matching function spatially by blurring the images. As the displacement becomes larger, the amount of blur required to smooth the matching function becomes also larger. This averaging of pixels leads to a loss of detail in the image.

Therefore, there is a trade-off between the size of the objects that can be tracked and the displacement that can be captured.

In this thesis we address the basic problem of increasing the size of the basin of attraction in a matching function. We use an image descriptor called distribution fields (DFs). By blurring the images in DF space instead of in pixel space, we increase the size of the basin attraction with respect to traditional methods. We show competitive results using DFs both in object tracking and optical flow. Finally we demonstrate an application of capturing large motions for temporal video stitching.

# TABLE OF CONTENTS

# LIST OF TABLES

# LIST OF FIGURES

**Figure** **Page**

# CHAPTER 1

# INTRODUCTION

Finding correspondences between images is a wide, long-standing area of study in computer vision. Given two images, it is the problem of having the machine automatically understand where things moved from one image to the next. It underlies many different problems for example optical flow [50], object tracking [28], image alignment [99], stereo vision [74], structure from motion [57], background subtraction [33] or motion segmentation [90]. In other words, it is an important building block for the advancement of computer vision. All these vision problems differ from each other in the characteristics of the input images and the desired output. Input images can be related temporally, like in optical flow and tracking, or they might be related spatially like in stereo vision. They might contain rigid changes, like in image alignment, or they might contain non rigid deformations like in optical flow. Also, the output correspondences might be dense, containing a correspondence at each pixel, or sparse. They might be parametric or non-parametric. They might have pixel or sub-pixel accuracy, etc. All these phenomena in which the problems differ have made methods evolve in diverse ways. Still, they often have in common the formulation of an objective function that evaluates the similarity of the matches. This objective function is optimized to compute the parameters of the correspondences.

There are at least two aspects that make this process difficult. First, the design of this similarity function is hard, because regions of the image change as a result of different illumination changes, occlusions and disocclusions, deformations, pose changes, etc. Second, optimizing this function is also hard, because the space of

parameter values that represent the correspondences can be too large to explore exhaustively. Consider for example the case of optical flow, where for a pair of images of size $m \times n$, we want to compute a two-dimensional displacement at each pixel, with sub-pixel accuracy. We have $2 \times m \times n$ continuous variables to optimize. To avoid this computational cost, methods often use variants of gradient or coordinate descent as the optimization scheme.

The main problem of this optimization method is that it relies on the assumption that the start of the search is within the basin of attraction of the correct local minimum. This is often the case when the displacement is small, and current methods obtain very accurate results for small motions [8]. As motions become larger, the start of the search will be further from the correct solution, and less likely to lie within its basin of attraction. A common practice to avoid this problem is to blur the input images spatially. This blur translates into the smoothing of the objective function. As the displacement becomes larger, the amount of blur required to smooth the objective function also becomes larger. This averaging of pixels leads to a loss of detail in the image. Therefore, there is a trade-off between the size of the objects that can be tracked and the displacement that can be captured.

In this thesis we address the basic problem of increasing the size of the basin of attraction in the objective function. For this, we use an image descriptor called distribution fields (DFs). Instead of blurring the image, we convert the images into a DF and blur the DF instead. We show that this increases the size of the basin attraction with respect to traditional methods. We show competitive results using DFs both in object tracking and optical flow. DFs have appeared in the literature before under the name of Channel Representations for different applications, like image denoising. To our knowledge, the work presented in this thesis is the first application of this image representation to motion estimation. In this document we use the name of Channel Representations and Distribution Fields interchangeably.

In the last part of this thesis we demonstrate an application of capturing large motions for temporal video stitching. While the first two parts focus on the basic vision problem of motion estimation, this last part goes a step further and shows an actual application of motion estimation to the graphics problem of video stitching. We describe the tools needed for this application, and the insights learned. This adds a broader, more application-based perspective on the basic problem of motion estimation.

## 1.1 Contributions

Our contributions are:

- Demonstrating the use of DFs for object tracking by providing an algorithm that obtained state-of-the-art results at the time of publication. This was published in the proceedings of the conference in Computer Vision and Pattern Recognition in 2012.

- Demonstrating the use of DFs for optical flow by providing an algorithm that obtains competitive results and improves over the baseline. This was published in the proceedings of the European Conference in Computer Vision in 2014.

- Providing an application of long range motion analysis to temporal video stitching. This was submitted to the International Conference on Computational Photography of 2015.

## 1.2 Outline

The rest of the document is organized as follows. Chapter 2 describes DFs, the key experiment that shows their power in widening the basin of attraction and their application to object tracking. Chapter 3 describes how to integrate DFs into the

traditional optical flow framework, and shows experimental results. Finally in chapter 4 we explore an application of our methods to the graphics problem of temporal video stitching. We find that different methods provide different advantages for our specific application.

# CHAPTER 2

# LONG RANGE OBJECT TRACKING

## 2.1 Introduction

Object tracking is the problem of finding an object in an image sequence given the location of the object in the first frame. Typically both the input initial location and the output locations along the sequence are bounding boxes. Object tracking is a basic problem in computer vision, and it is useful for different applications including surveillance or alignment, and a wide variety of problems in robotics.

In this chapter we address the problem of tracking an object that had large displacement. In object tracking this is important for two reasons. First, because it is often the case that the object undertakes a large displacement. Second, because object tracking is an unsupervised problem, where the template of the target needs to be updated online. Thus, mistakes due to convergence to the wrong local optima can accumulate easily, causing the track to drift.

We first show that our DF representation for tracking has a wider *basin of attraction* around a target's location than a large number of other descriptors that have been used in the tracking literature (see Section 3.5). Second, we show that a simple tracker built from this descriptor outperforms all other trackers on a standard tracking data set. Finally, we show that our tracker does not drift in a very long video sequence.

The rest of chapter is organized as follows. In Section 2.2 we describe the previous work on descriptors for tracking. In Section 2.3 we define DFs and the operators over them. In Section 2.4 we describe the tracking algorithm. Section 3.5 shows

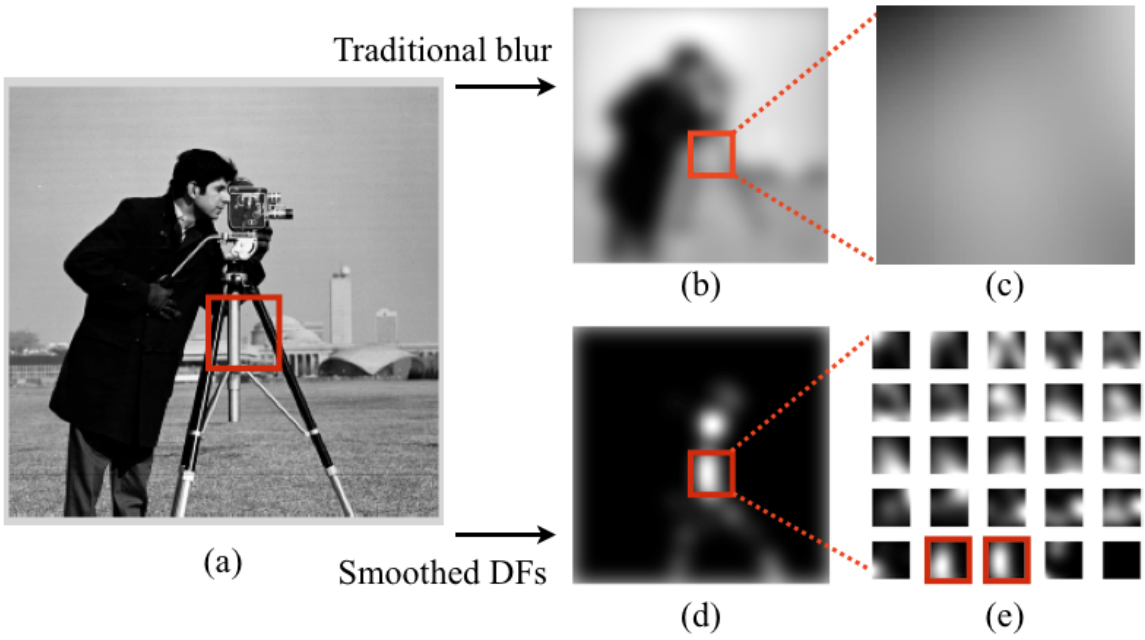



Figure 2.1: **Information preserved using smoothing on a DF. (a)** Original image. **(b)** Image smoothed with traditional blur. **(c)** Patch of image (b) where there central bar used to be. **(d)** Layer of the DF corresponding to the intensity value of the bar. **(e)** Collection of patches of the DF under the location of the central bar. When the image is blurred, the central bar of the tripod has blended in completely with the background (c). When an image is exploded into a DF and smoothed, the central bar of the tripod is still represented in one of the layers.

experimental evidence on the superiority on the basin of attraction and tracking performance of DFs.

## 2.2 Previous work

Tracking algorithms have different aspects including motion modeling, image representation and model update. The main focus of this chapter is the representation of the image using a descriptor that can overcome the challenges of visual tracking.

One common approach is to use a template to represent the object. This template can consist of the intensity values, gradient information, or other features [7]. These techniques have limitations because they might be overly sensitive to the spatial

structure of the object. This means that even if the optimization reaches the global optimum, this might not coincide with the correct position of the object due to changes in appearance. Robust metrics [75] alleviate this problem, but performance decays in long sequences [52]. DFs allow the representation of uncertainty in the descriptor, where small misalignments or occlusions can be represented as "unlikely" events as opposed to "impossible" events, mitigating the oversensitivity to spatial structure.

Another problem with template-based methods is that the objective function might not be smooth enough to reach the global optimum, as pointed out by Szeliski [100]. Often, the function is smoothed by blurring the image, for example using a Gaussian pyramid [3]. Recently, it has been proven that the Gaussian pyramid is not always the best choice for smoothing the objective function [78], and alternative blur kernels have been derived [78] specifically for smoothing the optimization landscape. Blurring the image has the undesired effect of destroying information about the pixel values. Depending on the size of the target, the levels of the pyramid and the characteristics of the background, the target might disappear completely. In the DF framework, the layered, or channel-by-channel, blurring technique allows smoothing the objective function without the mix of information that occurs in traditional blurring. This process is illustrated in Figure 2.1. The result is a smooth function with a wide basin of attraction around the object location. Figure 2.2 shows an example in one of the benchmark frames. Section 2.5.1 shows a numerical comparison between the width of the basin of attraction of different methods across many images, showing that DFs provide a wider basin of attraction than the other descriptors.

An alternative to building a template is using a histogram to describe the object [28, 26]. Histogram-based (also called kernel-based) descriptors integrate information over a large patch of the image. As a result they are not overly sensitive to spatial structure and they provide a larger basin of attraction. These methods have had

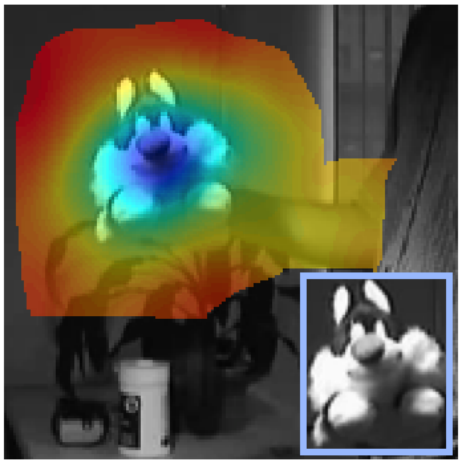


Figure 2.2: **Basin of attraction**. The target is displayed in the blue rectangle in the lower right corner of the figure. If the center of the target is located at any of the colored pixels, gradient descent leads to the correct position (within a 3×3 neighborhood). These pixels are color-coded according to the value of the objective function at that point.

a lot of success because they are fast, simple, and invariant to many pose changes. The main problem of kernel-based methods is the loss of spatial information that happens when building the histogram. This creates ambiguity in the optimization function [46] decreasing the specificity of the descriptor. In order to resolve these ambiguities, the size of the descriptor should be expanded. Recent efforts include some spatial information in the descriptor by using multiple kernels [46], or multiple patches [34]. These methods improve the performance of the single histogram descriptor, but require other mechanisms to decide the number and shape of the kernels. Fan et al. [35] propose a solution for the placement of the kernels, but a change in object appearance may make these kernels unstable. Also, if the number of these kernels is small, the tracking accuracy is more vulnerable to occlusion or abrupt motion. Other additions are higher order statistics [16] or temporal information [23], and using feature selection [27, 65, 109]. DFs capture the rich and robust information contained in a histogram while preserving the spatial structure of the object by having

a distribution at each pixel and can be viewed as a soft combination of template-based and histogram-based descriptors.

DFs are related to many previous image representations used for different purposes. These previous descriptors can be viewed as special cases of DFs, and they have many of the desired properties listed above. To our knowledge, only the general case of DFs together with the set of operators described in Section 2.3 presents all the properties.

DFs have been used for image denoising [37, 38, 54, 36, 43] and pose estimation [53], under the name of *Channel Representations*. In this document we use the two names interchangeably, and choose one or the other depending on the application, to be consistent with the published literature.

In background subtraction, Elgammal et al. [33], and Stauffer and Grimson [92] use DFs for modeling the background. A pixel is classified as background or foreground depending on the probability of belonging to the background. These descriptors can adapt to changes in appearance and be robust to certain noise and illumination. However, these descriptors do not spread information in space, and thus they are sensitive to small displacements.

In object detection and recognition, descriptors like HOG [30] and SIFT [73] use histograms of gradients. These can be viewed as downsampled DFs with gradient as the feature space. The large "support" of each histogram yields a smooth descriptor and a smooth objective function. However, since the size of this "support" is fixed, the basin of attraction is much more limited; if the search for the patch starts at a position that does not overlap with the actual location, it is not possible to converge to the correct location. Another example of histogram-based descriptors used for object recognition are *shape contexts* [13]. In this descriptor, a point is represented as the histogram of the image edges over the spatial location relative to the point. A shape context can be viewed as a DF at each point of the edge, with two differences: the

histogram of a shape context uses polar coordinates instead of Euclidean coordinates, and the shape context contains only one *layer* of the DF, because they take as input an edge image. As a result, shape contexts are subject to the quality of the edge detector. These distributions were later generalized to contain the orientation of the edges [79]. In this generalization, each bin contains the dominant orientation of the edges in the bin. This descriptor can be seen as a DF over edge space, where the coordinates are again different, and each column of the DF is used to produce the expected value of the edge orientation. Geometric blur [14] is another histogram-based descriptor used for object recognition. It can be viewed as a DF where each layer is the output of orientation tuned edge detectors. The main difference is that the convolution kernel is space variant, wider at the periphery than in the center.

Another use of DFs was demonstrated by Learned-Miller [63] in developing the congealing framework for joint image alignment. In this work, the likelihood of each image is maximized with respect to the DF defined by the set of images. Congealing [63] creates a large basin of attraction for alignment, since combining a collection of images that slightly differ from each other can smooth the optimization landscape.

Finally, DFs are related to the bilateral filter [101], which is a way of smoothing an image such that both proximity in space and feature value are taken into account to preserve image detail. While this filtering is useful for image denoising and image filtering, in general it does not smooth the energy function. As a toy example, suppose an image of a white circle over a black background. If we use a bilateral filter, the image will remain the same (unless the width of the kernel used for the feature dimension is very large, which is not typically the case). Thus, the energy function will not be smoothed, and the basin of attraction will not be wider.

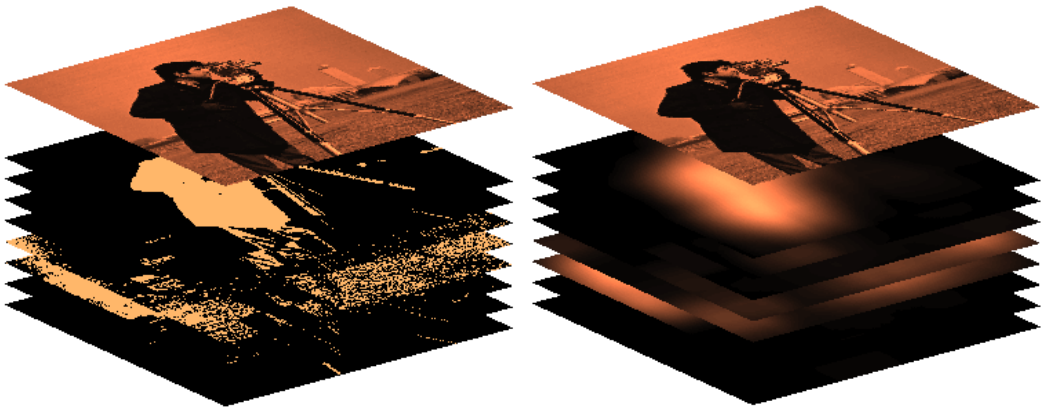


Figure 2.3: **Example of Distribution Fields. Left:**. DF after exploding the "cameraman" image. (The original image is shown superimposed on the DF for clarity.) The number of brightness levels (or layers) has been quantized to 8. **Right:** The same DF after smoothing in the dimensions of the original image.

## 2.3 Description of Distribution Fields

A DF is an array of probability distributions, one at each pixel of a "field" that is the size of the image. This distribution defines the probability of a pixel of taking each feature value. For example, if the feature space is gray-scale intensity, then at each pixel there is a probability distribution over the values 0-255.

### 2.3.1 Representation

A DF is represented as a matrix $d$ with $(2 + N)$ dimensions, where the first two dimensions are the width and height of the image, and the other $N$ dimensions index the feature space that we choose. For example, if the feature space is intensity, then an image of size $m \times n$ yields a 3D DF of size $m \times n \times b$, where $b$ is the number of intensity feature values, or bins. For a higher dimensional feature space, such as gradient, we can build a 2D distribution at each pixel location, yielding a DF of four dimensions.

### 2.3.2 Construction

*Exploding an image* into a DF results in a Kronecker delta function at each pixel location. In particular, exploding an image $I$ into $d$ with as many bins as features values is defined by

$$d(i,j,k) = \begin{cases} 1 \text{ if } I(i,j) == k \\ 0 \text{ otherwise,} \end{cases} \tag{2.1}$$

where $i$ and $j$ index the row and column of the image, and $k$ indexes the possible values of the pixel. We call the collection of bins at a fixed depth $k$ a *layer*. This produces a degenerate probability distribution at each pixel since the sum of the components of each column is 1. The left side of Figure 2.3 shows the results of computing this DF for the well-known "cameraman" image.

At this point, the DF representation contains exactly the same information as the original representation, albeit in a larger representation. We now show how to "spread" the information in the image without destroying the brightness values as occurs with traditional blurring.

The right side of Figure 2.3 shows a smoothed version of the DF on the left. The 3D DF has simply been convolved with a 2D Gaussian filter which spreads out in the $x$ and $y$ dimensions, but not in the feature dimension. That is, each layer $k$ of the smoothed DF $d_s$ is computed as

$$d_s(k) = d(k) * h_{\sigma_s}, \tag{2.2}$$

where $h$ is a 2D Gaussian kernel of standard deviation $\sigma_s$, and $*$ is the convolution operator.

Prior to convolution, we could interpret any value of 1 in layer $L$ of a DF to mean "there is a pixel of value $L$ at this location in the original image." After convolution, the semantics of the smoothed DF is, for any non-zero value in a layer $L$, "there is a pixel of value $L$ somewhere near this location in the original image." Thus, the

convolution has introduced positional uncertainty into the representation. *A critical point is that no information has been lost about the* **value** *of pixels in the original image, only about their position.* This is because there has been no mixing of pixel values during the convolution process.

It is easy to show that if the convolution kernel in Equation 2.2 is itself a probability distribution, then the smoothed $d_s$ maintains the property that each column of pixels integrates to 1,[1] and hence is still properly called a DF.

The previous discussion describes smoothing a DF in the $x$ and $y$ image dimensions. Smoothing can also take place in feature space. This allows the model to explain small changes due to subpixel motion, shadows, and changes in brightness. In a grayscale image, this smoothing is a 1D Gaussian filter over the third dimension. Each of the columns of $d_s$ can be smoothed to produce $d_{ss}$ as

$$d_{ss}(i,j) = d_s(i,j) * h_{\sigma_f}, \tag{2.3}$$

where $h_z$ is a 1D Gaussian kernel of standard deviation $\sigma_f$.

In summary, exploding an image into a DF and smoothing it can be viewed as introducing uncertainty about the object appearance. A DF is then a compact representation of the image itself and a set of its "neighboring" images. These images are the result of transforming the original image with small changes in appearance and in location. These are weighted according to the simple assumption that the most likely event is that the image will stay the same, and larger changes are less likely.

---

[1]This property breaks down at the boundaries of images. In order to avoid this problem, the missing information outside the boundaries is filled with uniform distributions.

### 2.3.3 Comparison

The comparison between DFs that different images yield can be done with any distance function. In this chapter we use the $L_1$ distance between the two arrays $d_1$ and $d_2$ as:

$$L_1(d_1, d_2) = \sum_{i,j,k} |d_1(i,j,k) - d_2(i,j,k)|. \tag{2.4}$$

### 2.3.4 Combination

Combining the information of several DFs can also be useful. In tracking we combine the DF of initial model and the DFs of new observations using a component-wise convex combination of them, which also yields a DF:

$$d_{t+1}(i,j,k) = \lambda d_t(i,j,k) + (1-\lambda) d_{t-1}(i,j,k) \tag{2.5}$$

By combining DFs of different instances of the same object we build a non-parametric data-driven model of the distribution at each pixel. This is useful for learning the statistics of the appearance of the object during tracking.

Further details on the operations over DFs and for the probabilistic interpretation can be found in our earlier technical report [85].

## 2.4 Tracking Algorithm Details

DFs can be used in a simple tracking algorithm. A model of the target is created by exploding the image that contains the target into a DF and smoothing it. Searching for the target in a new frame consists of building a new DF by also exploding and smoothing the new frame, and following the direction of the gradient of the $L_1$ difference between the DF of the model and the underlying part of the bigger DF representing the new frame. Once a local minimum is reached, the model of the target is updated, using a linear combination of the model and the new observation, as in Equation 2.5.

For better performance, we use a hierarchical approach. Instead of using a single DF to represent the target, we use a small set of DFs, where each of them is built using an increasing value of the parameter $\sigma_s$, which regulates the amount of spatial blur. These DFs contain information at different frequencies. At each frame, we use a coarse-to-fine strategy. The most smoothed DF is used to start the search, until it reaches a local minimum. This position is the start for the search in the second DF.

The method for choosing the value of the parameters $\lambda$ and $\sigma_f$ is using leave-one-out cross validation. The result of picking $\sigma_f$ separately for each video happens to coincide for all of them to be $\sigma_f = 10$. This is also the case for $\lambda = 0.95$.

Parameter $b$ corresponding to the number of bins was chosen, for speed, as the smallest power of two that does not hurt the performance of tracking algorithm on the videos. This is $b = 16$. The schedule of $\sigma_s$ for each video was chosen also using cross validation but conditioned on the size of the target. For each video, we choose the schedule of $\sigma_s$ that performs best for the video whose target is closest in size.The only motion model present in the algorithm is the assumption of constant velocity for computing the start of the search.

We summarize the procedure in pseudo-code in Algorithm 1

Here $h_s$ and $h_f$ are a 2D and a 1D Gaussian filters built with $\sigma_s$ and $\sigma_f$ respectively.

The computational cost of our algorithm is dependent on the number of bins used. We used $b = 16$. In a naive implementation, the running time is then 16 times that of template-matching with a Gaussian pyramid. However, the step that requires extra computation is the convolution of each layer of the DF. It is important to notice that this step can be completely parallelized. We have implemented this algorithm in a GPU in real-time at 70 frames per second on a PAL video ($768 \times 576$ pixels).

**Algorithm 1** Tracking with distribution fields

**Require:** $V$ = video sequence.
$I$ = patch containing target in frame 1.
$\boldsymbol{\sigma}_s$ = set of spatial smoothing parameters.
$\sigma_f$ = brightness smoothing parameter .
$b$ = number of brightness bins ($b = 16$).
$\lambda$ = mixing parameter ($\lambda = 0.95$).
**Ensure:** $(x, y)_f$ {Positions of target at each video frame $f$ in $V$}

Initialize $d^i_{model} = explode(I) * h_{s(i)} * h_f, i \in 1, ..|\boldsymbol{\sigma}_s|$
Initialize target location $(x, y)$ to center of patch $I$.
**for** $f = 2 \rightarrow |V|$ **do**
  **for** $i = 1 \rightarrow |\boldsymbol{\sigma}_s|$ **do**
    $d^i_f = explode(f) * h_{s(i)} * h_f$
    $(x', y') = \underset{(x,y)}{\text{argmin}}\ L_1(d^i_f(x, y), d^i_{model})$
    $(x, y) = (x', y')$
  $d_{model} = \lambda d_{model} + (1 - \lambda) d_f(x, y)$

## 2.5 Experiments

One of the main advantages of DFs is the width of the capture range around the position of the target. To illustrate this, we first compare the width of the basin of attraction of DFs to other descriptors. Second, we show that the tracking algorithm that uses DFs is able to outperform other state-of-the-art methods in standard benchmarks.

### 2.5.1 Experiments on Basin of Attraction of DFs

Here we evaluate the improvement in the basin of attraction achieved by using DFs compared to other descriptors.

#### 2.5.1.1 Experiment Description

For an objective function $f(x, y)$ and a point $p$, the *basin of attraction* is the region around the point $p$ from which descending the gradient of $f(x, y)$ leads to $p$. The size of this region is crucial for tracking algorithms that follow gradient descent to avoid exhaustive search. In this experiment, given an image, a patch is randomly selected and displaced in the horizontal direction. The task is to find, using gradient

descent, the true position of the patch. This procedure is illustrated in Figure 2.4. Since there is no noise or distortion, this task isolates the problem of creating a spatially smooth objective function to that of creating an objective function robust to changes in appearance. 289 images from the UWA data set[2] are used. The size of the patches is 30×30, and they were displaced from 1 to 30 pixels in each direction of the horizontal axis. We compare DFs to six other related or commonly used descriptors. These are the three traditional techniques: sum of squared distances (ssd), normalized crossed correlation (ncc), sum of squared distances of the blurred image (blur), meanshift using Bhattacharyya distance (ms-bhatt), meanshift using $L_1$ (ms) and multiple kernel tracking descriptor (mkt). We use both $L_1$ (df) and Bhattacharyya (df-bhatt) for the DF descriptor. The size of the kernel used for blurring is the same for all descriptors that use blur. Figure 2.4 shows an example of the result of evaluating each objective function around the true location of a patch for one particular image.

#### 2.5.1.2 Results

Figure 2.4 is the cumulative histogram of the size of the basin of attraction for each image. It shows how often each descriptor is able to successfully follow a gradient back to its original position as a function of the original displacement. For example, the DF descriptor (cyan and magenta lines) is able to follow a gradient back to its original position from a displacement of 15 pixels more than 90% of the time, while normalized cross correlation can only do this about 20% of the time.

There are two important points to take from this graph. The first is that the basin of attraction of DFs is larger than for any other descriptor. The reason for the superiority over traditional blur is that blurring the DF descriptor, as described in Equation 2.2, does not mix the values of different pixels. The reason for the superiority

[2] `http://www.cs.washington.edu/research/imagedatabase/groundtruth/`

over the other kernel-based descriptors is that the size of the signature is increased and therefore there is more specificity among patches. Kernel-based descriptors are a special case of DF, where the value of $\sigma$ is fixed and only the distribution of the center pixel is used. The second point is that the basin of attraction of a DF is consistently superior despite the distance metric used, this is, $L_1$ and Bhattacharyya. Thus the descriptor in this experiment is more influential than the distance metric.

### 2.5.2 Comparative Tracking Experiments

#### 2.5.2.1 Data Set

To evaluate the performance of the tracking algorithm we use the list of videos compiled by Babenko et al. [4] that is publicly available.[3] Since there is no standard benchmark for tracking, we choose these videos because they are the ones that most authors have compared to and they represent a valuable *de facto* standard for evaluation. They exhibit a wide range of phenomena from occlusion, object deformation, significant change in object appearance (subject turns 360 degrees out of plane), moving complex backgrounds (surfing). They also show a variety of objects being tracked such as faces, toys, and soda cans. Some examples of the frames can be seen in Figure 2.5.

#### 2.5.2.2 Algorithms for Comparison

We compared the performance of our algorithm to three other algorithms. We used the trackers that, to the best of our knowledge, performed best at the time this work was done on most videos of this data set: MIL [4] and PROST [83]. MIL uses unsupervised online learning over Haar features to train a discriminative model over the two classes background and foreground. In PROST, three different trackers are used and combined using a cascade. PROST has only been run on a subset of six of

[3]The videos can be found at `http://vision.ucsd.edu/~bbabenko/project_miltrack.shtml` One of the links to the videos was broken.

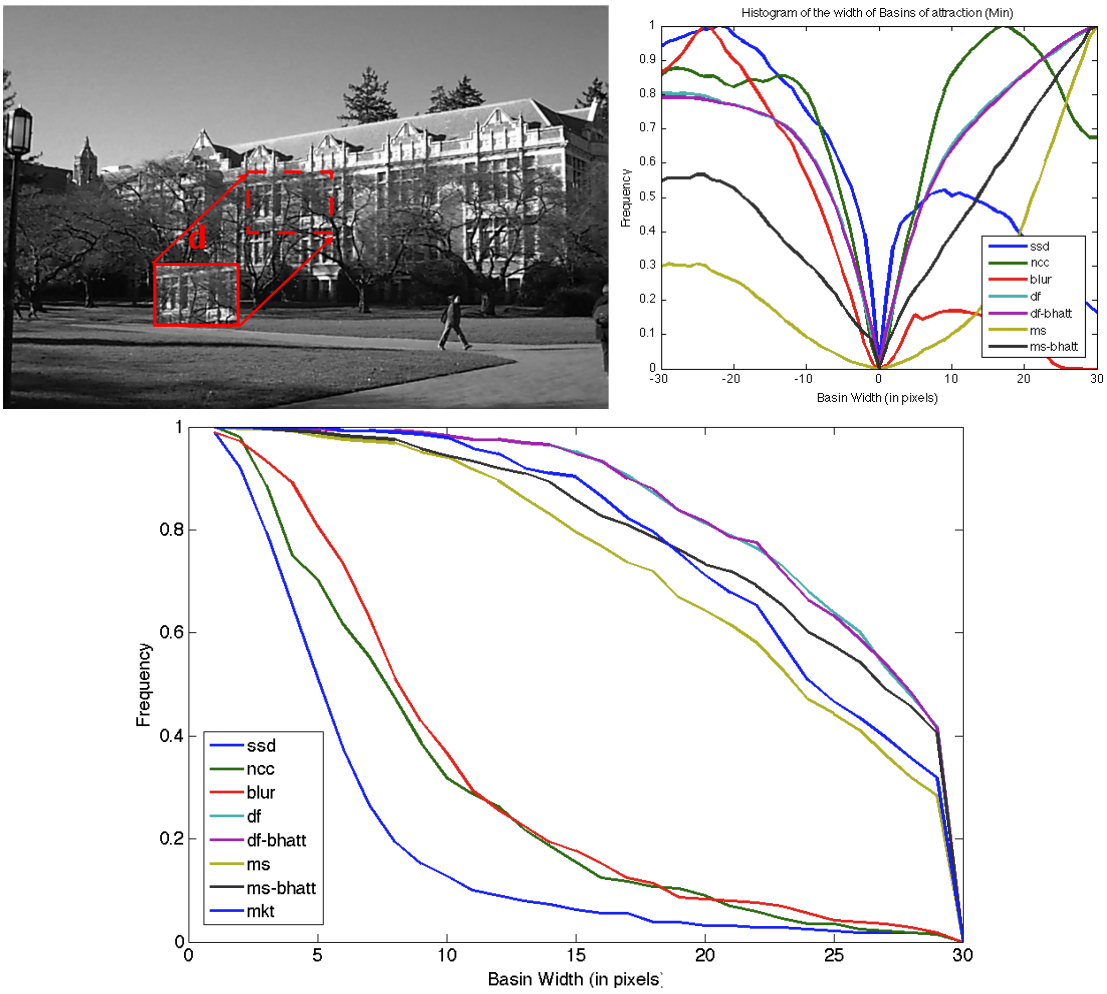



Figure 2.4: **Experiment on the basin of attraction of different similarity metrics and descriptors. Top left:** The dashed rectangle is the true position of the patch, and it is displaced by a distance $d = 30$. The basin of attraction tests show how often a patch is able to find its original position at this displacement. **Top right:** One instance of the different distance metrics evaluated translating a patch 1-30 pixels in both directions horizontally. **Bottom:** This plot shows, for a variety of descriptors, the distribution of the size of their basin of attraction. For a particular metric and a particular value of the basin, the graph shows what percentage of the images have a basin of at least that value. For example, using the similarity metric SSD, 50% of the images have a basin of attraction of 5 pixels or larger.

| Video | DF | PROST | MIL | MKT |
|---|---|---|---|---|
| tiger1 | **88.57** | 79 | 43.14 | 12.86 |
| david | **100.00** | 80 | 58.91 | 30.43 |
| sylvester | 66.79 | **74** | 73.88 | 20.90 |
| girl | 73.00 | **89** | 55.20 | 7.00 |
| faceocc | **100.00** | **100** | 77.28 | 6.21 |
| faceocc2 | **98.76** | 82 | 78.13 | 27.33 |
| coke11 | **75.86** | - | 17.93 | 27.59 |
| surfer | **94.67** | - | 56.00 | 62.67 |
| dollar | **100.00** | - | 90.76 | 15.38 |
| tiger2 | **81.94** | - | 46.39 | 8.33 |
| cliffbar | **87.69** | - | 72.31 | - |

Table 2.1: **Percentage of frames correctly tracked**.

| DF | PROST | MIL | MKT |
|---|---|---|---|
| **6.49** | 7.20 | 17.60 | 79.13 |
| **9.97** | 15.30 | 23.45 | 98.62 |
| 15.92 | **10.60** | 10.62 | 49.24 |
| 21.57 | **19.00** | 32.76 | 105.05 |
| **5.00** | 7.00 | 27.28 | 102.47 |
| **11.25** | 17.20 | 21.06 | 87.68 |
| **7.19** | - | 20.85 | 20.33 |
| **5.20** | - | 12.06 | 17.54 |
| **5.26** | - | 15.15 | 81.26 |
| **6.75** | - | 18.97 | 69.48 |
| **7.77** | - | 12.23 | - |

Table 2.2: **Distance between track and ground truth**.

the videos, and therefore we can only compare directly to it on these. In addition, our tracker is most similar in spirit to the Multiple Kernel Tracker [46] (MKT), and so we implemented and ran it on the same suite. The kernels used in our implementation are those described in their experiments, which are an Epanechnikov kernel and a roof kernel. The MKT descriptor is updated with the same scheme as used for DFs.

#### 2.5.2.3 Quantitative Analysis

We use two different metrics for the analysis of the tracking results. These are the mean distance to the ground truth (Table 2.2) and the percentage of frames correctly tracked. A frame is correctly tracked if the track and the ground truth have an overlap that is larger than half the union of their areas. This is, if the estimated location is rectangle $A$, and the ground truth is rectangle $B$, then a frame is correctly tracked if $(A \cap B)/(A \cup B) > 0.5$. This is shown in Table 2.1. We consider this metric to be much more informative than the distance, since once the track is lost the distance to the ground truth is somewhat arbitrary, and might bias the average distance. However, we report both for consistency with existing literature. Since MIL is stochastic, we show the average of the five runs that they report.

The tables show superiority of DF with respect to MIL and MKT in all 11 videos. Compared to PROST, DF is better in 4 out of the 6 videos that they use for comparison. In the video "faceocc" where both algorithms track correctly 100% of the frames, the accuracy of DF is better.

#### 2.5.2.4 Qualitative Analysis

The tracking algorithm is able to overcome different challenges like moderate occlusion and moderate changes in appearances due to pose and illumination change. Occlusions can cause drift in systems that are updated online without supervision. When the DF that represents the model of the target is updated with an occluder, this is represented in the distributions with small weight, as an unlikely event. If the occluder is not present during many frames, the model DF will not be polluted with the occluder, and will successfully avoid drift. This is the case of many of the videos of the benchmark, as depicted in Figure 2.5. This is also the case for other changes in appearance, like changes in illumination and pose. In the two videos where DFs perform worse ("girl" and "sylvester") the track drifts due to very drastic changes in appearance. In the first one, the girl whose face is being tracked turns around twice before the track drifts. In the second, the object undertakes large pose and illumination changes simultaneously. All parameters in the algorithm are chosen automatically using cross validation.

### 2.5.3 Parameter Analysis

All parameters in the algorithm are chosen using cross validation. In this section we further study the effect of each of them in tracking performance. The most important parameters are $\sigma_s$ and $\sigma_f$, which are the standard deviation of the Gaussian filters in image space and feature space respectively. In the algorithm we use a coarse-to-fine strategy for $\sigma_s$, where we convolve the DF first with a large, flatter filter, and we progressively sharpen it. The initial large filters smooth the optimization land-

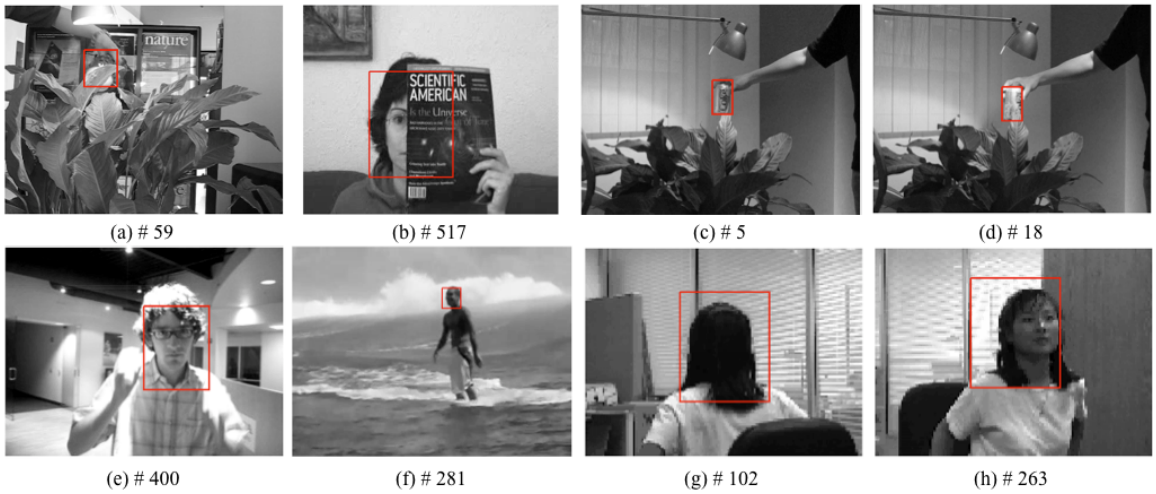



Figure 2.5: **Sample frames of the benchmark.** These frames correspond to the following sequences: (a) sylvester, (b) faceocc, (c, d) coke, (e) david, (f) surf, (g, h) girl. Our algorithm overcomes limited occlusions (a, b), moderate changes in illumination (c, d, e), and it is robust to low resolution outdoors sequences (f). Drift occurs when the changes in appearance are prolonged and very drastic (g, h).

scape, allowing the recovery of long displacements. The final smaller filters increase the specificity of the descriptor, allowing the optimization function to be more discriminative. If the value of $\sigma_s$ is too large, all patches might be too similar, and the track might be lost. If it is too small, the tracker might get stuck in a local minimum. Figure 2.6 shows the result of running the tracking algorithm with different configurations of $\sigma_s$. The bars for a given video show the sizes of $\sigma_s$ in ascending order. Although improvements in accuracy are not very smooth (since a single occlusion can cause the track to be lost forever), often larger targets are better tracked with larger values of $\sigma_s$. For example, "girl" and "faceocc" seem to perform better with larger sizes of $\sigma_s$. This seems reasonable, because descriptors that represent small patches can become indistinguishable from each other more easily when blurred, since they have lower dimensionality. This suggests that the parameter is somewhat consistent and what is most important, is generalizable. That is, the best value for a particular

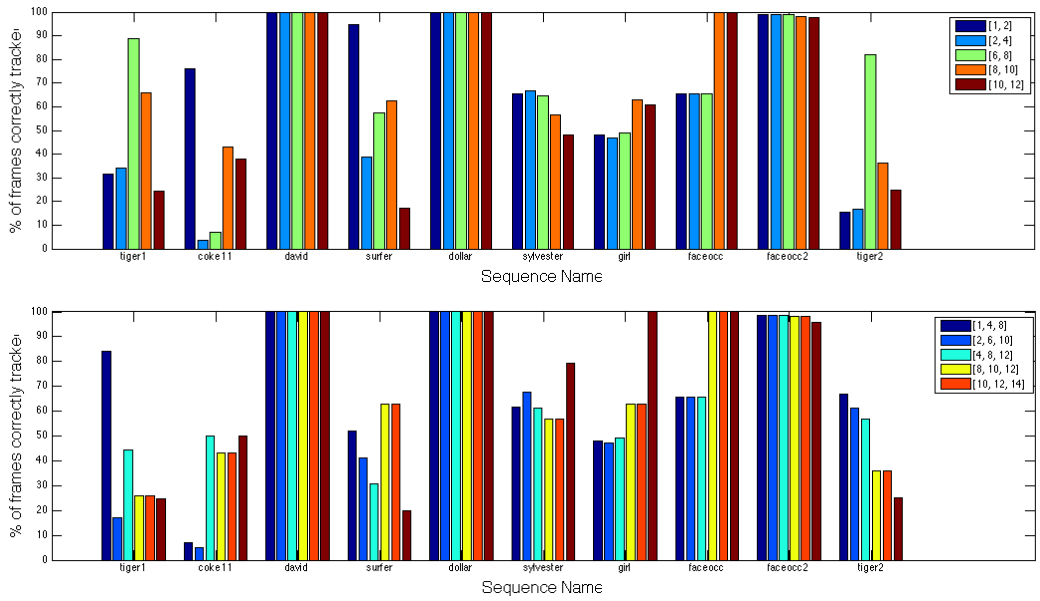



Figure 2.6: **Percent of frames correctly tracked across different $\sigma$ configurations**. **Top.** Using a two-level pyramid. **Bottom.** Using a three-level pyramid.

target will be very good for a different target in a different video if they have similar sizes.

The case of the value of $\sigma_f$ in feature space is similar to $\sigma_s$ in space, and in general smaller targets are better tracked with smaller kernels. However, there are other factors that also influence the best value of $\sigma_f$, such as the similarity between background and foreground and the variance in the appearance of the model. If $\sigma_f$ is very small, large targets tend to perform worse and videos with small targets tend to perform better. If $\sigma_f$ is very large, the opposite is true. This explains the large variation of the performance in small and large values of the parameter in Figure 2.7. Potentially, both $\sigma_s$ and $\sigma_f$ could be learned for the particular characteristics of the video. However, this particular suite of videos is too small and diverse for this purpose.

Parameter $\lambda$ controls the rate at which the model is updated as described in Equation 2.5. If the model is updated very fast, small errors accumulate quickly and

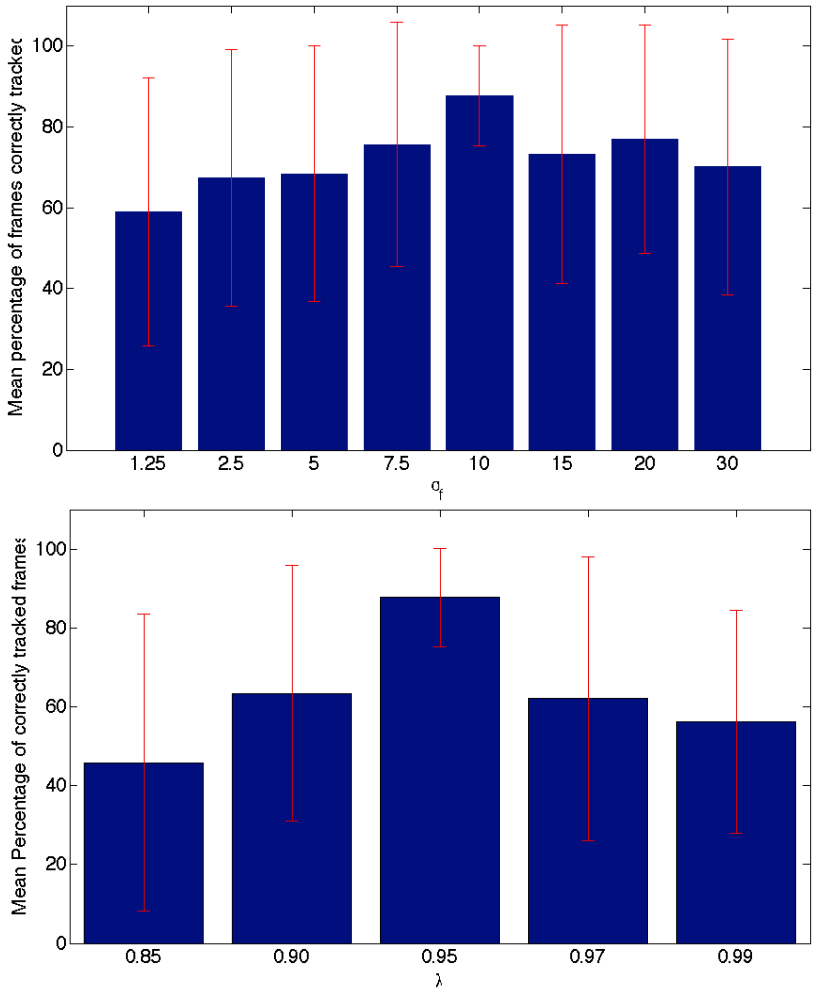



Figure 2.7: **Mean percentage of frames correctly tracked vs. value of $\sigma_f$ and $\lambda$.**

cause the track to drift. This trade-off is shown in Figure 2.7. If the model is updated very slowly, it may become outdated since it may be unable to reflect changes in the appearance.

### 2.5.4 Drift Avoidance over Long Sequences

Tracking algorithms that are updated online tend to drift over long sequences [76]. Once the model starts including a part of the background, errors will accumulate and the track will drift away from the target. We argue that our algorithm for tracking

with DFs is able to avoid this problem naturally by keeping a model of the target that is flexible enough to account for changes in appearance but allows a certain memory of the appearance model. We ran our algorithm with the parameters chosen as in Section 2.4 on a sequence of 19000 frames. Although during some frames there is some part of the background included (frames 3000 - 4000), the tracker successfully recovers as shown in Figure 2.8.

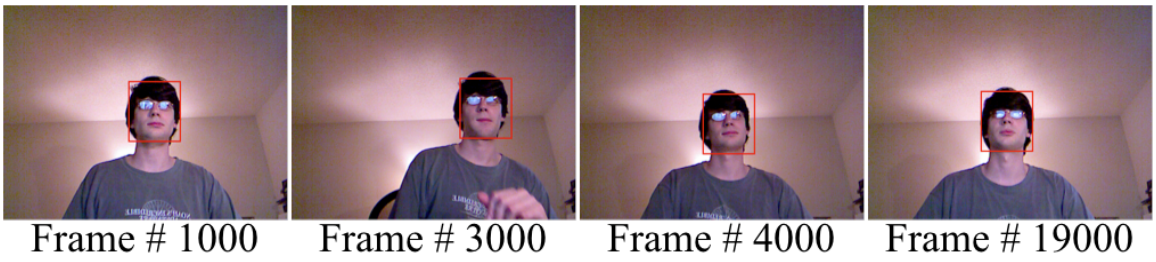



Figure 2.8: **Sample frames of a sequence with 19000 frames.** Although in some frames (3000 to 4000) there is a certain amount of background included in the model, this does not lead to drift. The tracker successfully recovers the correct position of the head even after 19000 frames.

## 2.6 Other Directions Explored

In this section we report extensions of this work that did not yield improved results, with the goal of casting some light into what may and what may not be promising future work.

### 2.6.1 Object Tracking using Higher Dimensionality

In this work we have estimated the displacement of the object along an image sequence. Many of the algorithms in the object tracking area also estimate only displacement, probably as a result of the ground truth being annotated only for displacements. However, it would be useful to estimate a higher dimensional transformation along an image. For example, this would allow large changes in scale or pose to not hurt the tracking. Also, a tighter bounding box could help a cleaner tem-

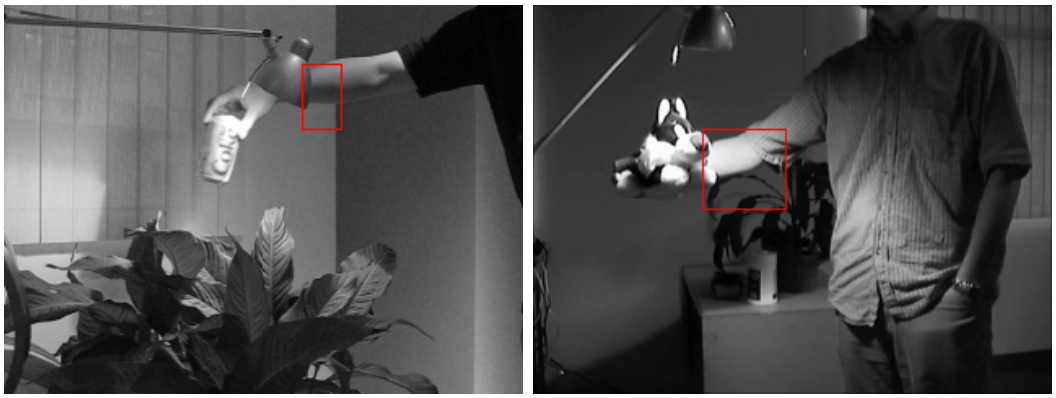


Figure 2.9: **Sample frames using brightness changes**. Frames 291 (left) and 1041 (right) of the two videos with strongest illumination changes.

plate over time, since presumably it should contain a smaller portion of background. To this end, we included different parameters in the optimization, such as scale and brightness correction. Neither of these yielded improved results. In the case of scale, the track tended over time to become smaller. Intuitively, this would make sense: it is easier to find a good match when two patches are smaller than it is when they are bigger. This is not just a matter of normalizing the similarity measure by the number of pixels; there exist more possible combinations of pixel values in a bigger patch. Even using a regularizer the track tended to become smaller overtime. A possible alternative could be to smooth the energy function directly, as done in synthetic examples in previous work [78]. In the case of illumination correction we found no improvement either. Although the performance did not drop dramatically, overall results were worse. In this case, occlusions or misalignments tended to be explained as changes in illumination, and the template would get degraded overtime. Figure 2.9 shows samples of the videos in the suite with the strongest changes in illumination.

Thus, integrating scale changes and other transformations into this framework remains an open problem

### 2.6.2 Sharpening Match

Since DFs contain a probability distribution at each pixel, one could draw a probabilistic interpretation of what a DF represents. For example, a DF could be seen as a process that can generate images by sampling from the distribution at each pixel. The comparison between DFs would be the likelihood of one being sampled from the other. More formally, given two images $I_1$ and $I_2$, the likelihood that $I_2$ came from $I_1$ is

$$P(I_2|I_1) = \prod_{x \in I_2} P(x|I_1), \tag{2.6}$$

which in the DF framework can be computed as

$$P(I_2|I_1) = \prod_{x \in I_2} (d_1(x) * \boldsymbol{\sigma}) d_2(x), \tag{2.7}$$

where both $d_1$ and $d_2$ are binary DFs and $\boldsymbol{\sigma}$ is a 3-dimensional Gaussian filter.

The advantage of this formulation is that the choice of parameter $\boldsymbol{\sigma}$ can also be chosen with probabilistic meaning by maximizing its likelihood as

$$L(\boldsymbol{\sigma}|I_1, I_2) = \prod_{x \in I_2} (d_1(x) * \boldsymbol{\sigma}) d_2(x) \tag{2.8}$$

which can be expanded to

$$L(\boldsymbol{\sigma}|I_1, I_2) = \prod_{i \in I_2} \sum_{j \in I_1} \frac{1}{(2\pi)^{(3/2)}} \frac{1}{\sigma_s^2 \sigma_f} e^{-\frac{1}{2}\left( \frac{(x_j - \mu_{x_i})^2 + (y_j - \mu_{y_i})^2}{\sigma_s^2} + \frac{(z_j - \mu_{z_i})^2}{{\sigma_f}^2} \right)}, \tag{2.9}$$

where $\sigma_f$ and $\sigma_s$ are the standard deviations of the Gaussians filter used to convolve in feature space and image space respectively. Changing the product for the sum of the logs and rearranging the terms, the expression for the log-likelihood is

$$\log L(\sigma_s|I, t) = |t| \log \left( \frac{1}{(2\pi)^{(3/2)} \sigma_s^2 \sigma_f} \right) + \sum_{i \in t} \log \left( \sum_{j \in I} e^{-\frac{1}{2}\left( \frac{(x_j - \mu_{x_i})^2 + (y_j - \mu_{y_i})^2}{\sigma_s^2} + \frac{(z_j - \mu_{z_i})^2}{{\sigma_f}^2} \right)} \right). \tag{2.10}$$

The automatic choice of $\boldsymbol{\sigma}$ has multiple advantages. First, it is difficult to set manually, since there exists a trade off. If $\boldsymbol{\sigma}$ is very large, all DFs will be very similar, because the distributions will be close to uniform. If $\boldsymbol{\sigma}$ is too small, small spatial shifts or changes in illumination might produce a much worse similarity value between the images than it should. Traditionally, blurring parameters are chosen empirically, and so an analytic solution would avoid the parameter tuning stage. Second, different images might need different parameters, and this process would then be adaptive. Finally, if the value of $\boldsymbol{\sigma}$ can be chosen at each step in the optimization, one would obtain an "automatic Gaussian pyramid", which would be very useful for many applications in computer vision. We called this process of updating the value of $\boldsymbol{\sigma}$ at each step in the optimization *Sharpening Match*. It is easy to be enamored with it.

We test the basin of attraction of this new similarity metric in the same experiment described in section 2.5.1. We test using a fixed value and an adaptative value of parameter $\boldsymbol{\sigma}$. Results are shown in Figure 2.10. Using the new similarity metric (*fixed-lk* in the figure) produces worse results than $L1$. We suspect this might be due to the product operator being more sensitive to outliers; a single pixel with a very low probability can drive the score down. However, when the value of $\boldsymbol{\sigma}$ is chosen automatically at each step of the optimization using Eq. 2.10, the performance actually improves (*step1-lk* in the figure). Figure 2.11 shows some examples of results in images.

We used this new similarity metric, together with the automatic choice of $\boldsymbol{\sigma}$ in our tracking algorithm. This yielded poor results, where the track would drift away from the object very quickly. The fact that in synthetic data the *sharpening match* helps performance, but in real data it does not, suggests that indeed it is very sensitive to outliers. These outliers do not occur in a synthetic setting where there are no changes in appearance.

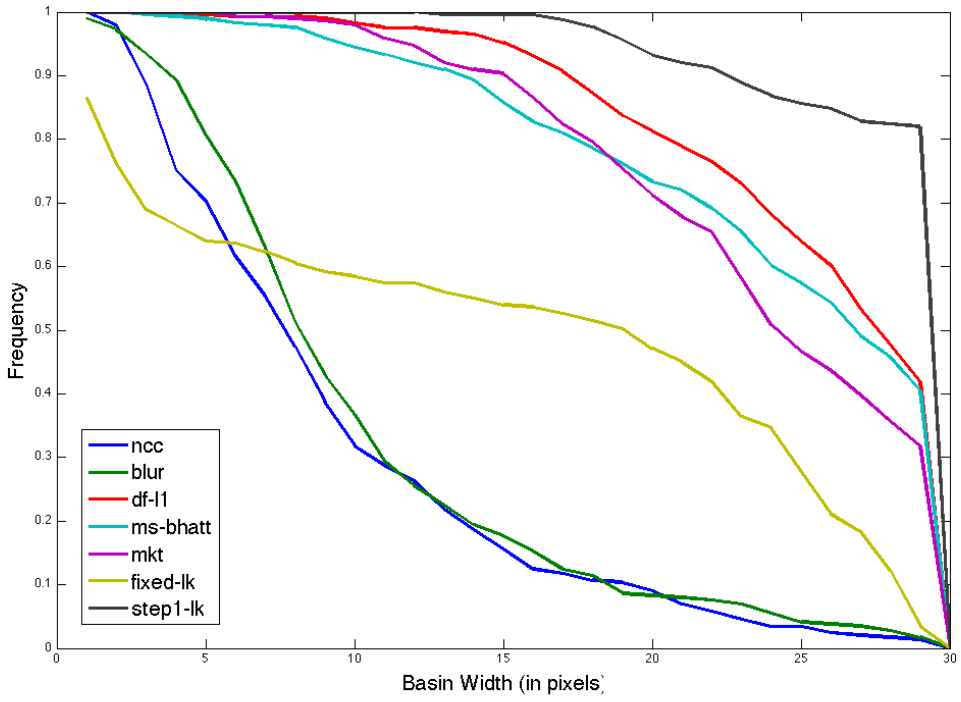



Figure 2.10: **Cumulative distribution of the width of the basin of attraction for different similarity metrics**.

### 2.6.3 Multi-view tracking

Probably the biggest drawback of the tracking algorithm presented is that when the object changes appearance too much over time, the model becomes degraded. This problem stems from the fact that DF representation does not capture the joint statistics of the observations, as it is described in Figure 2.12. When the object appearance changes drastically, the implicit assumption of independence between pairs of pixels leads to a corrupted model. For two pixels $x$ and $y$, their joint probability $P(x, y)$ is not modeled, and therefore it is not possible to know what combinations of $x$ and $y$ have high probability. For example, in Figure 2.12, $(x = 0, y = 128)$ might have high probability under the object model, but in reality it may not correspond to

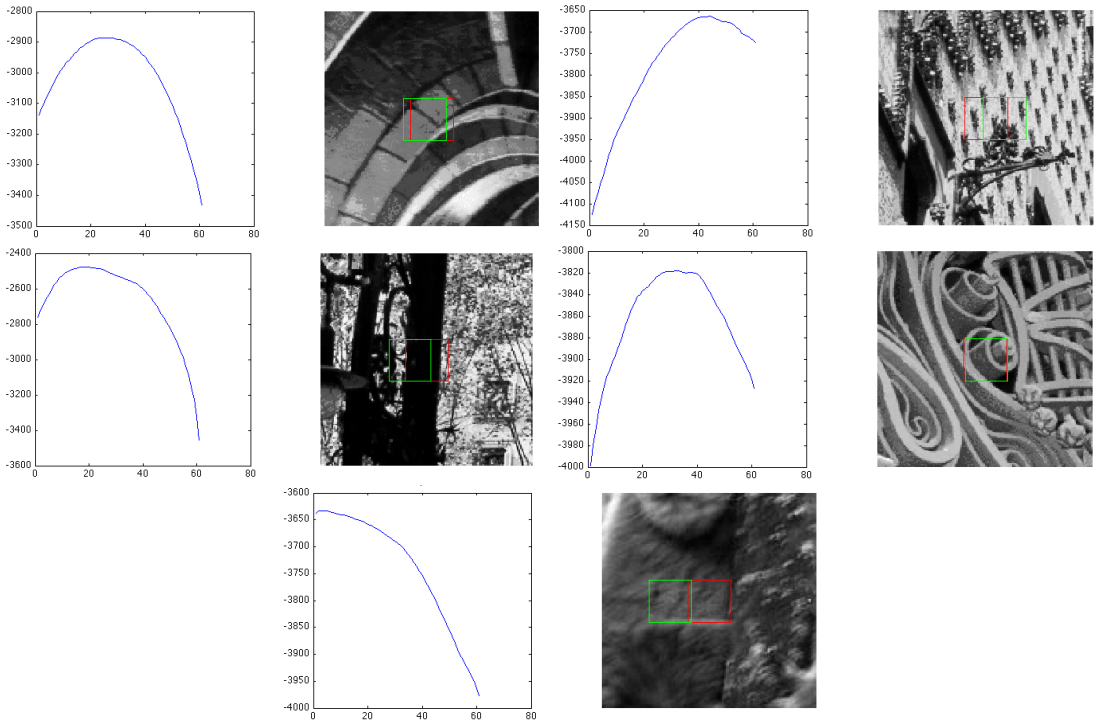


Figure 2.11: **Basins of attraction and local optima reached for different sample images**. Each graph shows the likelihood along the $x$ axis, the true position is at 31. Each image shows the true position of the patch in red and the local optimum reached in green. It seems reasonable that patches with rich texture (like the one in second row and second column) will be easier to find than those more homogeneous (like that one in the third row). Repeating patterns are likely to be confusing (as in the first row, second column), and some mistakes are surprising (like the one in second row first column). A typical example is the very first one, where the true position is missed by 5-6 pixels. One possible explanation is that the highest frequency information is lost when the DF is convolved with a very large value of $\boldsymbol{\sigma}$.

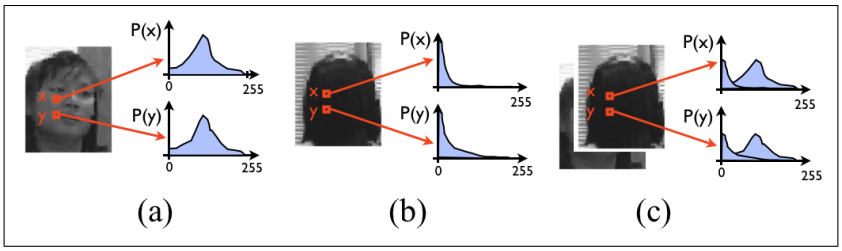



Figure 2.12: **DFs do not capture the joint statistics of the observations.** Representing an image with a DF provides a probability distribution at each pixel over the feature vales. In image **(a)** pixels $x$ and $y$ have high probability around value 128. In image **(b)** the pose of the target has changed drastically, and the observations are no longer coming from the same distribution. Pixels $x$ and $y$ have high probability of values close to 0. Combining the DFs in (a) and (b) results in the DF in **(c)**. Since $P(x, y)$ is not modeled, it is not possible to know what combinations of $x$ and $y$ have high probability. For example, $(x = 0, y = 128)$ would have high probability under this model, but in reality it does not correspond to any of the observations of the target.

any of the observations of the target. Modeling joint distributions in images becomes intractable quickly, since the number of variables, which are the pixels, is large.

We explored an approximation where we keep separate models of the object, one for each appearance or view. Each of these models is built with those observations that are likely to come from the same cluster. Two observations can be considered to come from the same cluster when their appearances are similar. This means that at each pixel $p$ the values of the different observations $i = 1, ..N$ will be quite similar, and therefore its distribution will be unimodal. The advantage of unimodal distributions is that there is less ambiguity about their joint probability, as explained in Figure 2.13.

A natural method to identify structure in a data set is to use a mixture model of several components. Clustering can be performed over the observations of the object in order to discover the different appearances. In particular, Dirichlet process mixture models (DPMM) are well suited for the problem of clustering for tracking because the number of necessary components is unknown a priori and also changes over time.

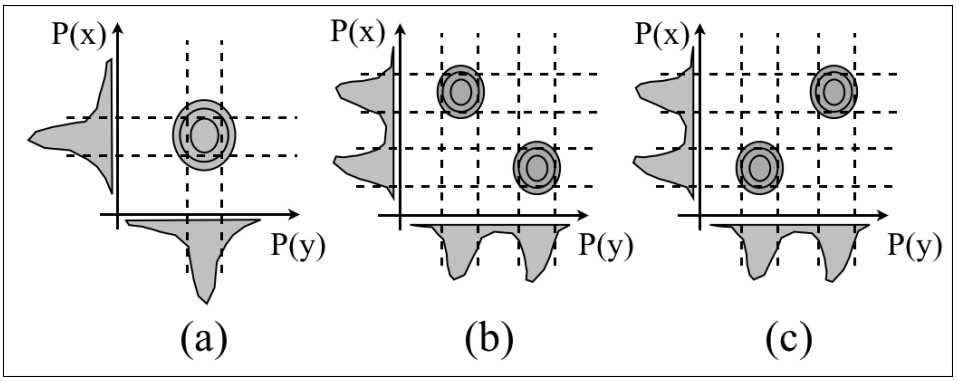



Figure 2.13: **DFs and bimodal distributions.** Unimodal probability distributions are less difficult to recover when only the marginal probabilities are available. When two distributions $p(x)$ and $p(y)$ are unimodal as they are in **(a)**, the region of high probability is easy to approximate. When two distributions are bimodal, as they are in **(b)** and **(c)**, different joint probability distributions could produce them. This ambiguity cannot be resolved.

We use DPMM to cluster the observations of the object, and then we build a DF for each of the clusters. The algorithm in pseudocode is described in Alg. 2.

In practice it is not necessary to cluster the observed images at every single frame. In our experiments, we cluster only after the first 100 frames. Also, for speed, the clustering is only performed every 10 frames after the first 100. The clustering is not performed directly over the raw pixel values. Instead, we compute the PCA representation of the images and use the first 10 principal components. This representation improves the clustering accuracy and performance time. We used a multivariate Gaussian with unknown mean and covariance for the cluster conditional distribution in the DPMM. The model parameters were learned using a collapsed Gibbs sampler [95]. The clustering is initialized using the output of running K-means with 10 clusters. We ran the sampler for 500 iterations, ignoring the first 100 for the burn-in phase, and then keeping every 10th sample.

To measure the tracking performance we use the percentage of frames correctly tracked.

**Algorithm 2** Tracking with Multiple Views

**Require:** $V$ = video sequence.
$I$ = patch containing target in frame 1.
$\sigma_s$ = spatial smoothing parameters.
$\sigma_f$ = brightness smoothing parameter .
$b$ = number of brightness bins ($b = 16$).
$\lambda$ = mixing parameter ($\lambda = 0.95$).
**Ensure:** $(x, y)_f$ {Positions of target at each video frame $f$ in $V$}
Initialize ${d_1}^1 = explode(I) * h_s * h_f$
Initialize target location $(x, y)$ to center of patch $I$.
**for** $f = 2 \rightarrow |V|$ **do**
  **for** each object view $i$ **do**
    Track object using view $i$ to optimum $(x'_i, y'_i)$
    Compute similarity score at optimum reached $s(i) = L_1({d_i}^f(x'_i, y'_i), {d_i}^{f-1})$
  Compute best view $\hat{j} = \arg\min_j s(j)$
  Reset starting point to $(x, y) = (x'_{\hat{j}}, y'_{\hat{j}})$
  Update best view with ${d_{\hat{j}}}^f = \lambda {d_{\hat{j}}}^{f-1} + (1 - \lambda){d_{\hat{j}}}^f(x'_{\hat{j}}, y'_{\hat{j}})$
  Compute clusters from best views $I_1(x'_{\hat{j}}, y'_{\hat{j}}), ..I_f(x'_{\hat{j}}, y'_{\hat{j}})$
  Update object views $d_j$ from clusters indices $c_j$

| Video | K-means(k=2) | K-means(k=5) | K-means(k=8) | Single-view | DPMM |
|---|---|---|---|---|---|
| tiger1 | **90.00** | 85.71 | 84.29 | 88.57 | 88.57 |
| david | **100.00** | **100.00** | **100.00** | **100.00** | **100.00** |
| sylvester | 89.92 | 88.43 | **90.30** | 66 .79 | 88.43 |
| girl | 70.00 | **86.00** | 83.00 | 73.00 | 77.00 |
| faceocc | **100.00** | **100.00** | **100.00** | **100.00** | **100.00** |
| faceocc2 | 98.14 | **98.76** | **98.76** | **98.76** | **98.76** |
| coke11 | 81.03 | 77.57 | 77.57 | 75.86 | **82.76** |
| surfer | 93.33 | 93.33 | 93.33 | **94.67** | **94.67** |
| dollar | **100.00** | 40.00 | 40.00 | **100.00** | **100.00** |
| tiger2 | 75.00 | 68.06 | 52.78 | **81.94** | 73.61 |
| cliffbar | 49.23 | 69.23 | 73.84 | **87.69** | 70.76 |

Table 2.3: **Percentage of correctly tracked frames**

We compare the results of this algorithm to two other algorithms. First, we use the single-view algorithm presented before. Also, we compare to the same algorithm described in Alg. 2 but substituting the non-parametric clustering by k-means clustering, with parameter $k$ fixed. We run this algorithm with different values of the parameter $k$. Our algorithm has 4 parameters, which correspond to the initialization of the hyperparameters of the DP. All the parameter values are chosen using cross validation. The results of these three algorithms on the 11 data sets are shown in Table 2.3.

There are two key points to take from the results table:

1. In all but two of the videos, tracking with multiple views performs at least as well as using a single view. It is important to note that the video *cliffbar*, which is the one that shows a big loss in accuracy, presents an object whose size changes throughout the sequence, to test the ability of an algorithm to adapt to scale. However, none of the algorithms that we study optimize for the scale changes of the object, and they are not expected to perform well.

2. Using a non-parametric model for clustering improves the object representation, and consequently the tracking. The experiments across different values of $k$ for KMeans show that for each video the best value of $k$ is different. For example, *coke11*, *tiger1* and *tiger2* perform better with smaller values of $k$, since the target appearance has many sudden outliers (occlusions, non-rigid deformations, and strong changes in illumination) that can hurt in the multi-view setting. However, others like *girl* and *sylvester* undergo large slow changes in appearance, which is the ideal scenario for having many different views, and therefore perform better with higher values of $k$. A bad choice of $k$ can hurt the results in the *dollar* video up to 60%. The use of a non-parametric model for clustering allows avoiding the choice of $k$, adapting the number of clusters to the complexity of the data. This has two main advantages. First the number of

clusters is automatically tailored to the nature of the data set in a data-driven fashion. Second, this number of clusters can be adapted over time, as more data appears along the tracking sequence. In practice, we observe that as more views of the object have appeared, there are more clusters discovered in the data set.

Having multiple views is particularly useful when the videos are long and the object appearance changes significantly. In the benchmark, the longest video is *sylvester*, where the target undergoes strong changes in illumination and pose. In this video, having multiple views allows the track to not drift away as a result of the accumulation of different views. An example of this improvement is shown in Figures 2.14 and 2.15 .

## 2.7 Conclusion

In this chapter we have used a descriptor called DF for the tracking of general objects in image sequences. Tracking with DFs has two contributions. First, they have a larger basin of attraction than other similar descriptors, which prevents the search from getting stuck in local minima. Second, DFs present a variety of advantages for tracking, they include spatial information in the kernel-based framework, which resolves ambiguity and overcomes the undersensitivity to spatial structure. They also resolve the oversensitivity that other descriptors have to the geometric structure of the target, and they are able to model slow changes in appearance and pose and be robust to minor occlusions.

We believe that DFs are a fertile framework for image comparison and there are many improvements to our algorithm that could be explored, such as modeling occlusion, or combining information over multiple feature spaces.

Figure 2.14: **Tracking with multiple views avoids drift.** First column: Sample of the tracking results with single view, at seconds 1, 36, 54 and 55 of the video sequence *sylvester*. As the target appearance varies a lot (first two samples), eventually the track drifts and cannot recover. Second column: Tracking results at the same frames with multiple views, showing no drift. Third column: Sample of the tracking results with single view, at seconds 1, 7, 17 and 27 of the video sequence *girl*. After self occlusion, the track slowly drifts. Fourth column: Tracking results at the same frames with multiple views, showing no drift.

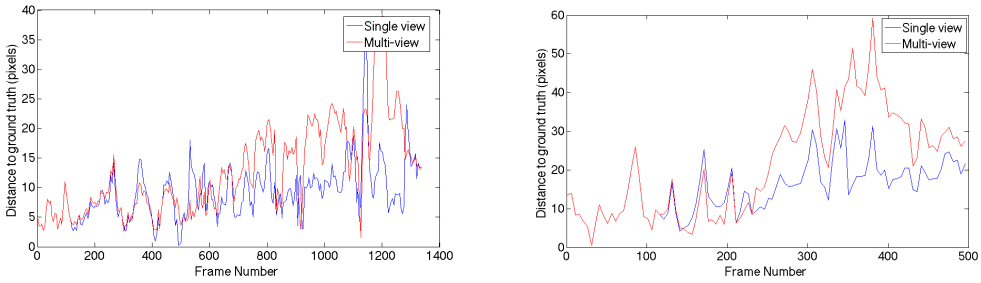



Figure 2.15: **Tracking with multiple views avoids drift.** Distance to ground truth as time evolves for *sylvester* (left) and for *girl* (right). On average the multi-view algorithm is closer and when the distance gets large it shows the ability to recover (for example around frame 1150, where it recovers much faster than the single-view).

# CHAPTER 3

# LONG RANGE OPTICAL FLOW

## 3.1 Introduction

In this chapter we address the problem of recovering large motions of small objects in optical flow. Small, fast moving objects are easy for humans to see and track. Their motion is important for biological tasks such as obstacle avoidance, catching, and predator detection. Figure 3.1(a) shows an example in which a small animated character is viewed from above, running through a bamboo forest [22]. In contrast to biological vision, current optical flow algorithms perform badly in such cases (Figure 3.1(e)). We find that this is particularly true for small or thin regions.

The issue stems from the basic assumptions of most current flow methods. Most techniques estimate dense optical flow using two constraints: brightness constancy and spatial smoothness of the flow field [98]. Brightness constancy assumes that the intensity value of a small region remains constant despite its change in location. The brightness term is a non-linear function of the flow and, in gradient-based formulations, is typically linearized for optimization. This linearization is valid only in the case that the displacement is small. In order to capture longer range motion, a coarse-to-fine method is employed [17, 20], typically using a Gaussian pyramid [21]. The pyramid is built by successively smoothing and downsampling the images. The problem with this approach is that, for scenes with multiple moving objects, this blurs the pixel values across object boundaries. For small or thin objects this means that at coarse (high) levels of the pyramid, the object may completely disappear; see Fig-

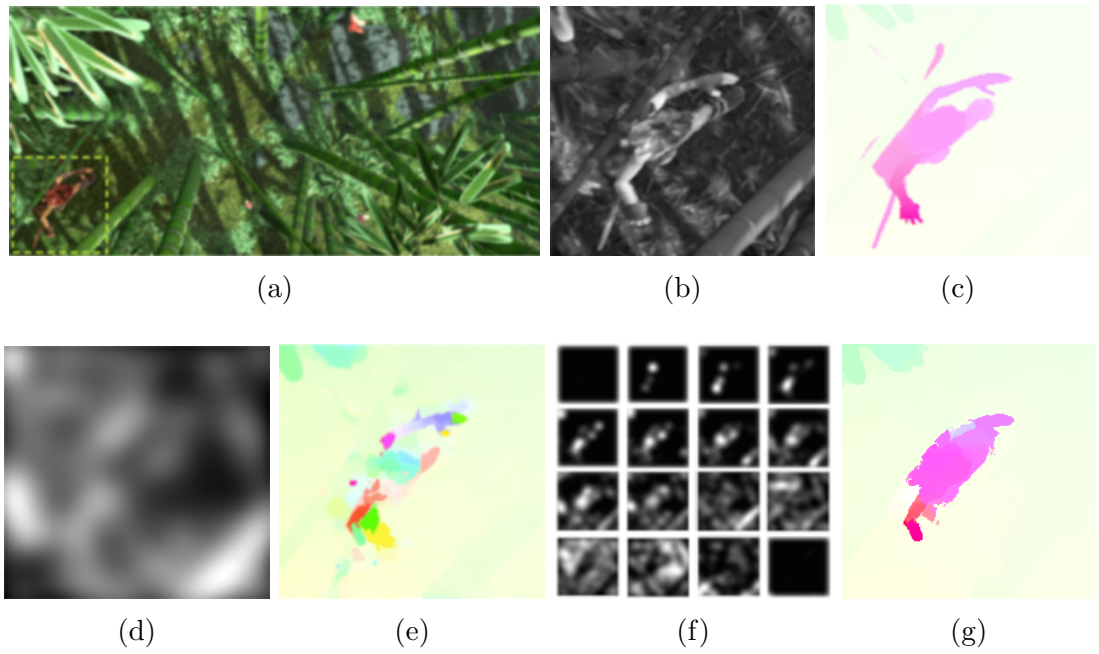
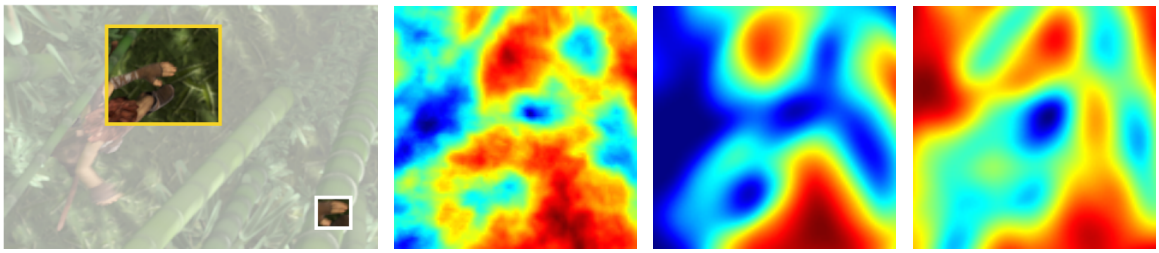



Figure 3.1: **Problems with Gaussian pyramids:** (a) Image from [22] (b) Detail of a small, fast object (c) Ground truth (d) Blurred image patch from high pyramid level (e) Flow using a Gaussian pyramid [96] (f) CR of the same patch blurred with same kernel size (g) Flow using a CR pyramid and our method.

ure 3.1(d-e) for an example of a blurred image region and the flow of the Classic+NL algorithm, which uses a pyramid [96].

Instead, if one could segment the scene into objects, then the objects could be matched across large displacements. But since object segmentation is itself an unsolved problem, we need an alternative. In this work we replace the brightness constancy assumption with a descriptor constancy assumption. For this we represent an image using a channel representation (CR), or Distribution Field (DF) described in Section 2.3. We use these two terms interchangeably in this document, to be consistent with the published work [87, 86]. This representation contains a descriptor at each pixel location. This descriptor is a locally weighted histogram.

The advantage of this representation, as described before, is that performing blurring in CR space does not introduce mixing of the brightness values of the pixels.



(a) (b) (c) (d)

Figure 3.2: **Pyramid versus CR pyramid.** Computing the sum of squared difference between the hand in the white region in (a) and pixels in the yellow region gives the error surface in (b) with many local minima. The global optimum is near the center. Using a Gaussian pyramid oversmooths the energy (c) and the minimum is less clear. Decomposing the image into a CR and blurring each channel using the same kernel as in (c) yields the energy in (d), which smooths the surface but maintains the global optimum. The error increases from cold (blue) to warm (red) color.

Instead, the image is decomposed into several different channels, according to the pixel intensities (or other image property). Each of these channels is then blurred separately (Figure 3.1(f)). This process allows spreading the information about the pixel values spatially. This smooths the optimization landscape, but prevents the averaging of pixel values that one sees in a Gaussian pyramid. An example of the effect of blurring using CRs in the optimization landscape is shown in Figure 3.2. This descriptor constancy assumption is also linearized, to fit in the traditional approach of flow estimation. We then apply the standard coarse-to-fine framework, creating a CR-pyramid by blurring each CR and downsampling it. This prevents losing some small objects at the coarse levels and oversmoothing motion boundaries.

In this work we start with the original Classic+NL technique. Classic+NL is a method that optimizes the two constraints common to most methods (smoothness and brightness constancy) using a coarse-to-fine approach. The parameters and pre and post processing steps are optimized for improved performance, and thus it is widely used as a baseline. From this method we simply replace the data term by our descriptor-constancy data term, leaving the rest of the system, including the param-

eter values, untouched. We call this new data term channel constancy or descriptor constancy. The only parameter that we change is the weight of the smoothness term, since the statistics of the values of an image and a CR are slightly different. We compare the performance of both systems in a synthetic setting as well as on the standard benchmark for large displacements, which is the *MPI-Sintel* dataset [22]. We find that this simple change improves results overall, especially in long range motions and at motion boundaries. This suggests that replacing pixel value constancy with channel constancy may also improve other optical flow algorithms.

## 3.2 Previous Work

The problem of recovering long range motion has previously been addressed [19, 93, 105], but results on the *MPI-Sintel* dataset [22] show that current methods still fail to capture really large motions, especially of small objects.

Many of the top performing algorithms on the standard datasets are based on the traditional approach of Horn and Schunck [50]. This method minimizes an energy function that is the sum of two terms: the smoothness term that encourages neighboring flow vectors to be similar, and the data term that encourages corresponding pixels to have the same brightness. These classical methods often use a coarse-to-fine approach [21] for computing optical flow. These methods smooth the images and, as a consequence, the optimization landscape, so that motions larger than 1 pixel can be estimated. However, smoothing the images enough to capture large motions makes it nearly impossible to recover the motion of small objects with large displacements.

In principle, this approach fails because each pixel is not discriminative enough. If we consider estimating optical flow as finding correspondences between image pixels, we would like each pixel to be uniquely identifiable so that the correspondence can be found easily. In the traditional approach, these correspondences cannot be found because the blurring makes pixels indistinguishable from each other. One way of

avoiding this is adding additional features at each pixel. Estimating the flow becomes finding correspondences where not only the pixel value matches, but also other pixel features like linear filter responses, edges and information about the neighboring pixels [71, 93, 94, 103, 105]. This is typically done by including additional feature matching terms in the original energy function.

However, including these terms in the classical approach is not trivial, because it makes the optimization of the energy function difficult. For these terms to be integrated in the framework, they need to be differentiable and increase away from the global optimum, which is often not the case. Therefore, this family of methods that try to incorporate additional features are difficult to optimize, and suffer from increased computational cost, or in some cases they restrict the solutions to a discretized space that does not reach sub-pixel accuracy.

Brox and Malik [19] take an important step towards incorporating additional features in the energy function. They precompute a descriptor at each pixel, and find the best match at the next frame for each pixel. Then they include a term in the energy function that encourages the estimated flow to be similar to the precomputed best feature match. While this is a step in the right direction, it does not directly include the similarity between descriptors in the global optimization. Our method fully integrates the descriptor matching in the global optimization of the energy function.

## 3.3 Methods

In this section we first explain the proposed energy function and its relationship to the traditional approach. We then describe how to compute the image descriptor used in this energy function.

### 3.3.1 Energy Function

Most optical flow formulations make assumptions about brightness constancy and spatial smoothness, in one form or another. In this formulation, we minimize the following energy function

$$E(u,v) = E_{\text{brightness}}(u,v) + \lambda E_{\text{smooth}}(u,v), \tag{3.1}$$

where $u$ and $v$ represent the horizontal and vertical flow fields from the first image $I_1$ to the second image $I_2$ respectively.

The first (brightness) term assumes that the brightness of a pixel persists over time and is typically formulated as

$$E_{\text{brightness}}(u,v) = \sum_{x,y} \rho\left(I_1(x,y) - I_2(x+u_{(x,y)}, y+v_{(x,y)})\right). \tag{3.2}$$

where $u_{(x,y)}$ and $v_{(x,y)}$ represent the flow at a pixel $(x,y)$ and where $\rho(\cdot)$ is a robust penalty function that downweights the influence of outliers (i.e. violations of the brightness constancy assumption) [17].

The brightness constancy term is a non-linear function of the unknown flow fields. The traditional approach linearizes the brightness constancy term for optimization and requires the smoothing of the optimization landscape. This is typically implemented by blurring the image as part of a coarse-to-fine strategy. Depending on the size of certain objects in the scene and the amount of blur, some details may be lost. To preserve the details, we need to robustly smooth the object boundaries without mixing pixel values.

### 3.3.2 Channel Constancy Assumption

In our approach, the data term in the energy function enforces descriptor constancy, which matches the descriptor at each pixel instead of the pixel intensity. The

energy function becomes

$$E(u,v) = E_{CR}(u,v) + \lambda E_{\text{smooth}}(u,v), \tag{3.3}$$

where the descriptor constancy term enforces that each of the components in the descriptors should match. Let the descriptor at each pixel have $K$ components, and $d_1(x,y,k)$ be the $k$th component (level) of the descriptor at pixel $(x,y)$ in image $I_1$. Then the data term becomes

$$E_{\text{CR}}(u,v) = \sum_{x,y} \sum_{k=1}^{K} \rho\left(d_1(x,y,k) - d_2(x+u_{(x,y)}, y+v_{(x,y)}, k)\right). \tag{3.4}$$

Two CRs can be compared by comparing each of their corresponding components, as in Eq. 3.4. In this work we use two different metrics: the $L2$ distance ($\rho(x) = x^2$) and the generalized Charbonnier ($\rho(x) = (x^2 + \epsilon^2)^\alpha$) [24]. These choices were made based on previous studies [96] that illustrate their advantages. Note that we apply the function to each component of the CRs, not to the pixel values.

To understand what it means to compute the distance between two CRs, we can consider a simple case in which each input image has one non-zero pixel. We then consider the distance as a function of the difference in pixel values. The CRs blur the pixel value across several levels and the error is computed by summing the robust error across all $K$ levels in Eq. 3.4. In Figure 3.3 we plot this error as a function of pixel-value difference and compare the CR case with the standard intensity case for both the quadratic and Charbonnier penalty functions. The effective shape of the error function is quite different in the case of the CR error. In fact, the error function has an interesting property of saturating like a robust error function. This saturation happens even when the penalty function is quadratic.

The intuition here is simple. If there is no blurring across layers, then the difference between two CRs will be zero only when the input pixels have the same value and will

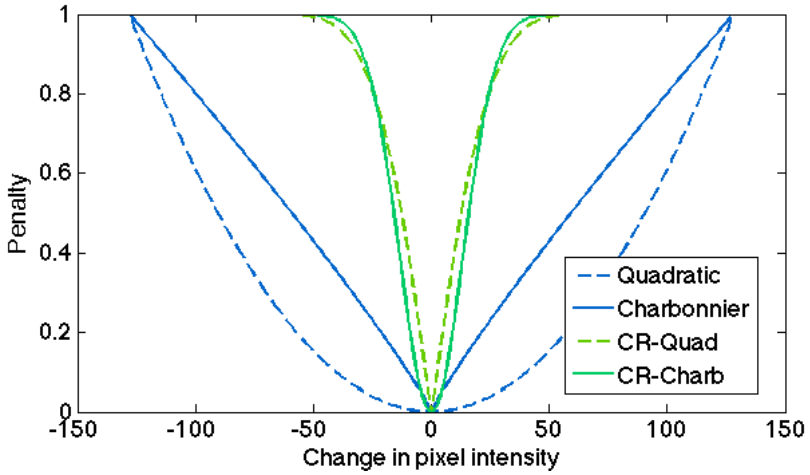



Figure 3.3: **Shape of the different penalty functions:** The generalized Charbonnier and quadratic distance functions change shape when applied to a pair of distributions instead of to a pair of pixels. This new shape is more robust to outliers but it is still convex in CR space like the quadratic function.

be constant for any difference in pixel values; that is, like an inverted delta function. Blur across layers allows different pixel values to be compared and the more smoothing there is the wider the convex region around zero.

## 3.4 Optimization

Our goal is to isolate and evaluate the effect of channel constancy versus brightness constancy. To that end we use the existing Classic+NL framework for flow estimation and simply replace the data term while keeping all other elements of the method the same.

### 3.4.1 Integration in Traditional Approach

Optimization in Classic+NL involves linearizing the brightness term as part of an incremental warping strategy that gradually warps the second image towards the first image.

By replacing the images with the CR, each incremental warping step linearizes the descriptor constancy term of the energy function (Eq. 3.4) around the current

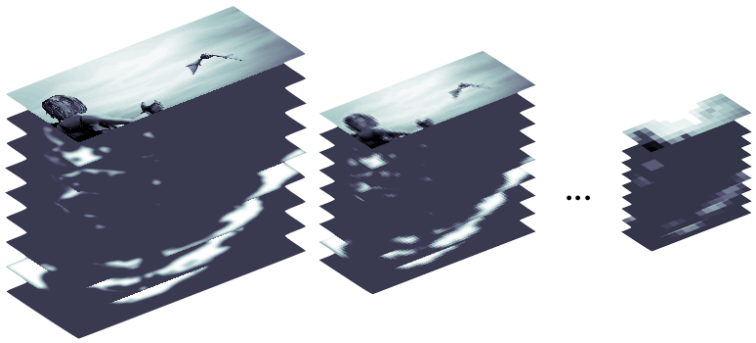


Figure 3.4: **Channel representation pyramid.** Each level of the pyramid is a CR, created by smoothing and downsampling the previous level. The original image is also shown here but is not part of the CR.

flow estimate $(u_0, v_0)$, differentiates Eq. 3.3 w.r.t. the flow increment $(du, dv)$, and sets the derivatives to be zero. The resultant linear equation system to solve for the flow increment are

$$\begin{bmatrix} \sum_k \tilde{\rho}(d) d_x(k)^2 + \lambda L & \sum_k \tilde{\rho}(d) d_x(k) d_y(k) \\ \sum_k \tilde{\rho}(d) d_x(k) d_y(k) & \sum_k \tilde{\rho}(d) d_y(k)^2 + \lambda L \end{bmatrix} \begin{bmatrix} du \\ dv \end{bmatrix} = - \begin{bmatrix} \sum_k d_x(k) d_t(k) + \lambda L u_0 \\ \sum_k d_y(k) d_t(k) + \lambda L v_0 \end{bmatrix},$$

where the weighting function $\tilde{\rho}(x) = 2(x^2 + \epsilon^2)^{a-1}$ for the generalized Charbonnier penalty $\rho(x) = (x^2 + \epsilon^2)^a$, $L$ is the Laplacian operator, $d_x$ and $d_y$ are derivatives of the CR w.r.t. $x$ and $y$ and $d_t$ is the derivative of the CR w.r.t. $t$. We compute the derivatives by taking the derivative of each layer of the CR, as if they were images.

### 3.4.2 Optical Flow Estimation

We use a coarse-to-fine warping-based approach to optimize the proposed energy function. Instead of using the traditional image pyramid [20], we use a pyramid of CRs, as shown in Figure 3.4. The first level is computed by exploding the image according to Eq. 2.1. Each successive level is computed from the previous one by smoothing it as in Eq. 2.2 and downsampling. We downsample each layer of a CR as if the layer were an image and interpolate using bicubic interpolation.

We compute the flow at the coarsest level of the two pyramids and use the flow to warp the second CR toward the first CR. Each layer of a CR is warped separately. At the next level, the computation of the flow starts from the position of the previous level, interpolating for the points where there is no estimation yet. We use a 3-stage graduated non-convexity (GNC) scheme for the optimization [98]. The first one uses a quadratic penalty, the last one uses a generalized Charbonnier penalty and the middle one uses a linear combination of the two (as in [96]). In practice, since the CRs need to be differentiated, the bottom level of the pyramid is also smoothed with a small Gaussian filter. We call the resulting algorithm *Channel Flow (CFlow)*.

### 3.4.3 Modeling the Change in Illumination

If there were no smoothing across layers in the $k$ (vertical) direction, the CR would be sensitive to slight changes in image intensity. Despite smoothing across layers, we find that the descriptor constancy is still sensitive to brightness variations. Classic+NL does not actually use brightness constancy but rather uses a texture decomposition for the data term, which reduces the effects of illumination change. In the case of the CR, we take a different approach and explicitly model illumination change.

Previous work [47, 106] has shown the advantage of using a model of the change in illumination for techniques that assume brightness constancy. The physics of the natural world make the changes in illumination in a scene multiplicative. However, many images have gamma-correction applied to them, which makes the changes in illumination have an additive effect on the image. Therefore, a plausible model for changes in illumination is: $I(x + u_{(x,y)}, y + v_{(x,y)}, t + 1) = I(x, y, t) + b(x, y)$.

What we need, however, is the effect of brightness changes on the CR. If a pixel changes brightness, this changes the *level* at which the pixel appears in the CR. Thus, to compensate for the brightness change we want to *warp* the CR in the direction

that undoes this change. This warping is analogous to the warping we do in space using the optical flow except it happens in the vertical direction of the CR.

To better understand how brightness varies, we used the Sintel training sequences. We warped adjacent frames together using the ground truth flow and computed the brightness difference $b$ at every pixel. We found that the distribution of $b$ values is tightly peaked at zero with heavy tails.

This leads us to a simple method to compute the brightness change. Given the current flow estimate, we warp the input images and compute their difference. We then apply a median filter of $21 \times 21$ to obtain a robust estimate of $b$ at every pixel. We can use the flow fields and the brightness change to apply a 3D warp (in space and level) to the CRs as follows:

1. Compute $d_1$ and $d_2$ from input images $I_1$ and $I_2$
2. Compute optical flow $(u, v)$ using $d_1$ and $d_2$ as described in Section 3.4.2
3. Compute $I_2^w$ by warping $I_2$ according to the flow $(u, v)$
4. Compute the change in illumination $b$ at each pixel as: $b = I_1 - I_2^w$
5. Filter this field $b$ with a median filter
6. Warp $d_2$ according to the field of 3D vectors: $(u, v, b)$ .

We also experimented with including the variable $b$ in the main energy function and optimizing it as we optimize $u$ and $v$. While this should be the optimal choice in principle, in practice we find that it is computationally considerably more expensive while presenting similar results to our approximation.

## 3.5 Experiments

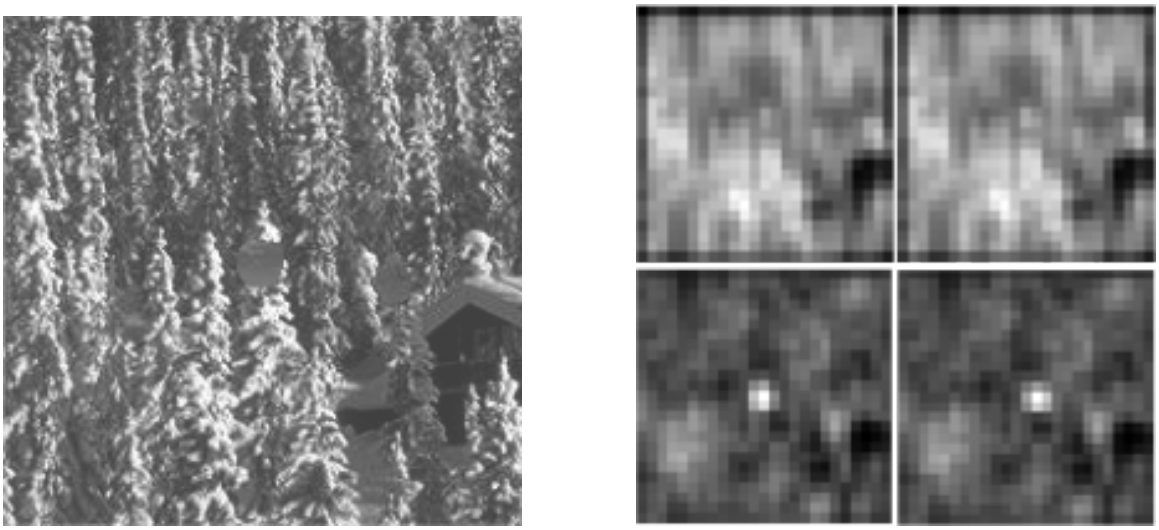
We have built Channel-Flow by replacing the brightness term in Classic+NL with the intent of isolating the contribution of this new formulation. Consequently, here

we provide a detailed comparison with Classic+NL. We focus our analysis on the *MPI-Sintel* dataset (both training and testing) because it contains many fast motions of small or thin objects. Below we use mean endpoint error (MEPE) as a measure of accuracy [8].

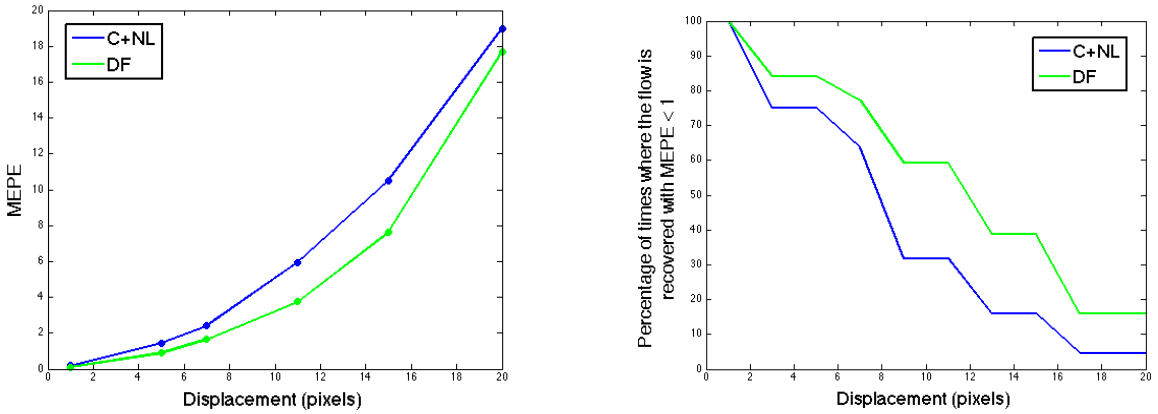
### 3.5.1 Synthetic Experiment

First we evaluate experimentally the core contribution of our technique, which is recovering large motions of small objects. Since it is difficult to create a real dataset of small objects moving fast with ground truth optical flow, we chose to create a synthetic dataset with the properties we want to test. To that end, we take natural images and create a sequence with a small (20 pixels) circular-shaped region in the foreground (Figure 3.5(a)). Then for a range of foreground displacements (0-20 pixels), we compute the optical flow and we measure the error. We repeat this for several images.

We compute the optical flow with 4 levels of the pyramid, using Classic+NL and also using our method. Figure 3.5(b) shows the third level of the Gaussian pyramids for two consecutive frames (top). Note that the foreground object is hard to be distinguished from the background. Even though the texture of the foreground and background were originally different, the blurring of the pyramid has eliminated much of this difference. In the bottom half of Figure 3.5(b) we show one of the channels of the CR-pyramid where the foreground was represented, also at the third level. Note that the object is visible as a bright spot near the center of the image in both frames. Clearly this is a trackable feature in the CR pyramid. This illustrates our central hypothesis, that the channel representation pyramid keeps information about different objects separate, allowing us to smooth within a layer and estimate long-range motions.



(a) (b)




(c) (d)

Figure 3.5: **Image pyramid versus CR-pyramid.** The foreground object in (a) is lost at the third level of the image pyramid (b, top), while still distinguishable at the third level of the CR-pyramid (b, bottom). This results in more accurate flow estimation for longer displacements (c). The histogram in (d) shows that in our set of images, the 20-pixel foreground object can be recovered.

Of course, this effect depends on the actual image values. We generated 44 such examples with different natural images, each with the same range of displacements. Figure 3.5(c) plots the MEPE as a function of displacement of the foreground (the background is stationary). As expected, the error of Channel-Flow is below that of Classic+NL, and this is particularly pronounced for motions above 10 pixels; that is, more than half the diameter of the foreground region.

When the methods fail, the errors are large, and these obscure what is going on. Consequently in Figure 3.5(d) we report the percentage of times that the flow is accurately estimated (with an MEPE $< 1$ pixel). When the object moves a distance half its size (10 pixels), we recover the true flow twice as many times as does Classic+NL.

### 3.5.2 Constant Albedo Sequences

#### 3.5.2.1 Experiment description

In this experiment we take the traditional Classic+NL algorithm and simply substitute the traditional brightness constancy by our descriptor constancy term. The **hypothesis** is that using a CR pyramid preserves more information at high levels of the pyramid, thus making the recovery of large motions and motion boundaries more accurate. In other words, we wish to test whether the basin of attraction around the true flow is wider using the CR data term than using the brightness constancy term. **Dataset:** We want to isolate this question, that concerns the optimization landscape, from that of which data term is more accurate or robust to certain phenomena. For this reason we use the albedo sequence of the *MPI-Sintel* training dataset, where pixel values do not change from one frame to the next.[1] Here we did not use the full training set but, rather, sampled a subset at random.

[1]Here we must use the training set because the test set does not include constant albedo sequences.

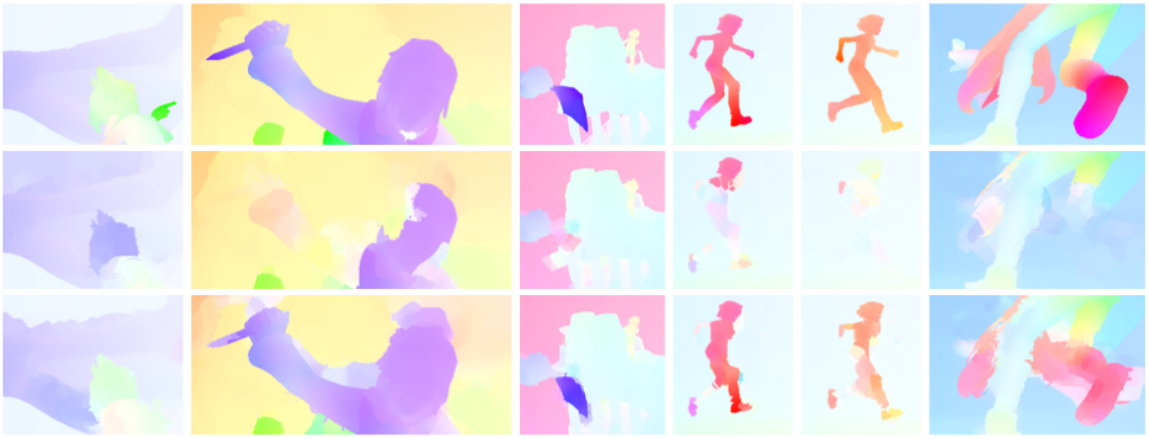


Figure 3.6: **Details of results on the albedo training sequences**. **Top row:** Ground truth. **Middle row:** Flow estimation with the traditional approach often fails to capture large motions, especially of smaller objects. **Bottom row:** Using CR's to represent the image improves the accuracy of the flow in such difficult regions. Some examples are (from left to right): Sintel's hair, the arm and knife, the bat's wing, Sintel's limbs, Sintel's body, Sintel's foot.

#### 3.5.2.2 Quantitative results

Numerical results are shown in Tables 3.1 and 3.2. In addition to the traditional mean end point error (MEPE), we report other statistics for further analysis, following the *MPI-Sintel* standard. Categories *matched* and *unmatched* group pixels according to whether they exist in both frames or not, the $s$- and $d$- categories group pixels based on their speed and distance to a motion boundary respectively. Further details can be found in the original dataset publication [22]. The table shows that that Channel-Flow produces overall better results than Classic+NL. The two columns where Channel-Flow outperforms Classic+NL by wider margins are: closer to the boundaries ($d10$) and in the large motions ($s40$). These are areas near motion boundaries and fast moving regions. This provides some confirmation, on complex sequences, of our original hypothesis that the CR should be better for these cases.

| Method | **MEPE all** | MEPE matched | MEPE unmatched |
|---|---|---|---|
| Classic+NL | 4.530 | 2.645 | **28.857** |
| Channel-Flow | **4.218** | **2.148** | 30.923 |

Table 3.1: **Albedo Sequence, Part 1.** Results on 398 non-consecutive, randomly chosen, image pairs of the albedo sequence of *MPI-Sintel*. Bold letters show the best results in the category.

| Method | d0-10 | d10-60 | d60-140 | s10 | s10-40 | s40 |
|---|---|---|---|---|---|---|
| Classic+NL | 5.701 | 3.238 | 2.038 | 0.770 | 3.117 | 18.710 |
| Channel-Flow | **4.779** | **2.529** | **1.618** | **0.694** | **2.211** | **15.403** |

Table 3.2: **Albedo Sequence, Part 2.** Results on 398 non-consecutive, randomly chosen, image pairs of the albedo sequence of *MPI-Sintel*. Bold letters show the best results in the category.

#### 3.5.2.3 Qualitative results

Figure 3.6 shows some examples of the performance of the two methods. Channel-Flow is able to recover certain details or fast motions where Classic+NL fails. On the other hand, Classic+NL produces very nice and smooth flow fields, while Channel-Flow has a stronger tendency towards piece-wise constant fields, presumably due to the data term being very robust to outliers (see Figure 3.3).

### 3.5.3 Experiments on "Final" Pass of the Sintel Training Set

#### 3.5.3.1 Experiment description

In this section we test our method in the same set of frames used in the previous section, but this time the frames contain complex phenomena such as changes in illumination, motion blur, fog, etc. The purpose of this experiment is to test the new data term under these additional phenomena. Then, for each of the problems that we identify we propose a solution. The main disadvantage of the CR formulation is that the penalty function is similar for pixel differences that differ by a little or a lot.

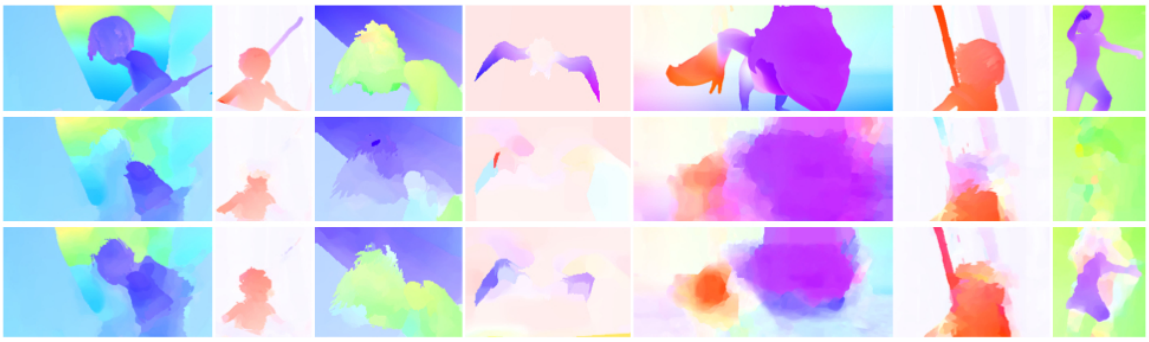


Figure 3.7: **Results on the "final" training sequences**. **Top row:** Ground truth. **Middle row:** Results with Classic+NL. **Bottom row:** Results with Channel-Flow and the two additions (C-Flow+I+C)

This problem can be seen in Figure 3.3, where the penalty is similar for a change in intensity of 50 and 100. This lack of gradient makes optimization unlikely to converge to the correct solution when there are brightness changes in the scene. To address this we use the illumination model described in Section 3.4.3.

We call the method with the illumination model C-Flow+I in Tables 3.3 and 3.4. In addition, in order to have valid derivatives, the finest level of the CR pyramid needs to be spatially smoothed. Therefore, the two frames are never compared without spatial blur. To refine the output, we use a 1-level pyramid of Classic+NL, and we call this method C-Flow+I+C in Tables 3.3 and 3.4.

#### 3.5.3.2 Results

Quantitative results are shown in Tables 3.3 and 3.4. We observe that both the illumination model and the 1-level pyramid steps improve results in all categories. Visualizations of the recovered change in illumination at each pixel are shown in Figure 3.9. We observe that our technique often recovers successfully the low frequency changes in illumination produced by fog, specular reflections, etc. Higher frequency changes are not recovered due to the wide median filter used. Qualitatively, in Figure 3.7 we see roughly the same behavior as in the albedo case. Some of the best

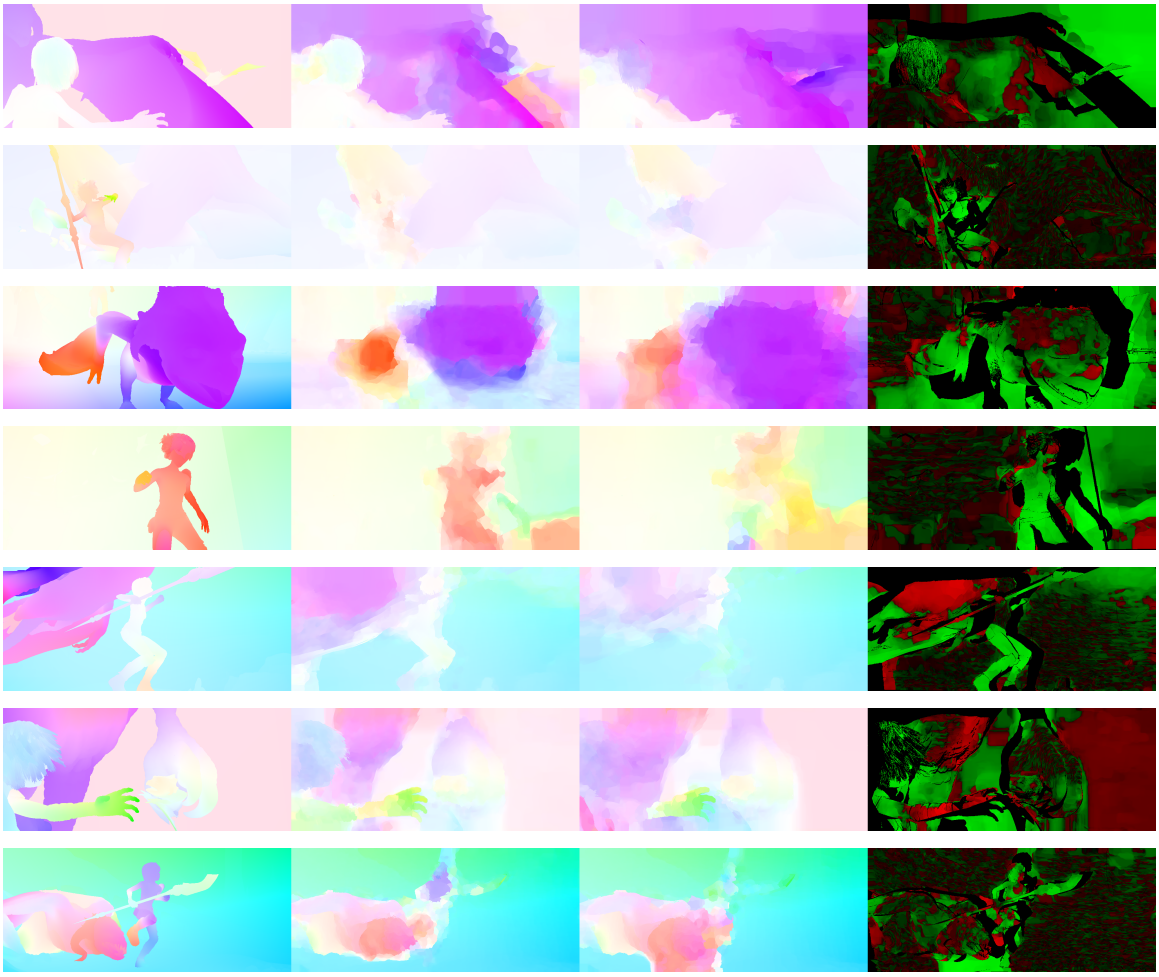


Figure 3.8: **Pixel by pixel comparison**. From left to right: Ground truth, Channel-Flow, Classic+NL, pixel EPE comparison.

performing full frames of Channel-Flow are shown in Figure 3.8. For each frame, we show the ground truth flow field, the result of Classic+NL, the result of Channel-Flow and a per-pixel comparison. This last image will have, at each pixel, red or green, depending on whether Classic+NL or Channel-Flow had a smaller EPE, respectively. The brightness of the color will be proportional to the difference in EPE. In long displacements and small or thin regions moving fast, Channel-Flow has an advantage over the traditional Classic+NL.

The runtime of the Matlab code for a $1024 \times 436$ *MPI-Sintel* image pair is about 5 hours on a standard Linux desktop. Half of the computational time is spent on

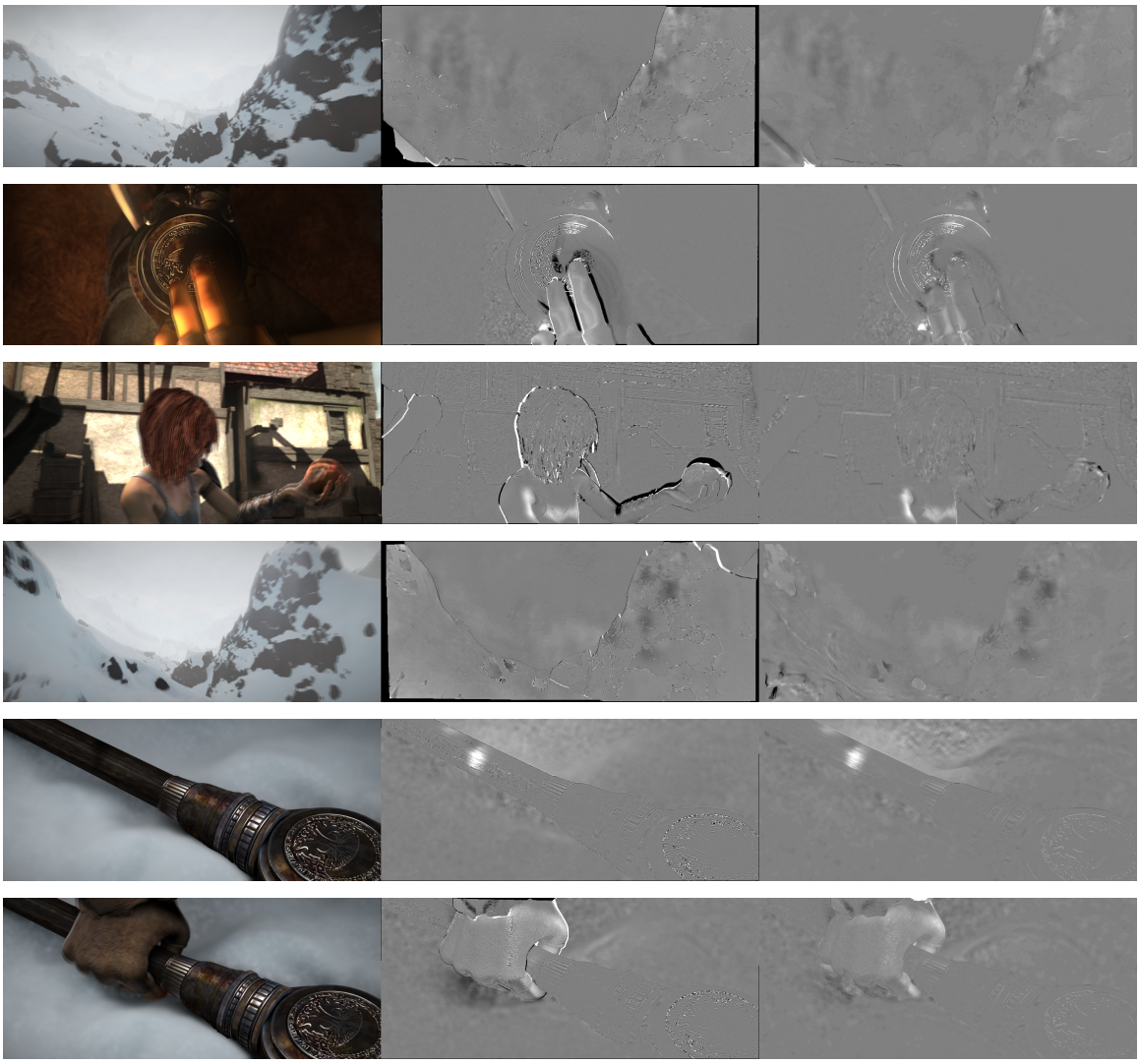


Figure 3.9: **Visualization of recovered change in illumination**. **Left:** Average of the pair of input frames. **Middle:** Ground truth change in illumination. **Right:** Estimated illumination change. The ground truth change in illumination is estimated by warping the second frame according to the ground truth flow field, and subtracting this from the first frame. If brightness constancy held, the result would be a constant image of zeros. However, changes in illumination and other complex phenomena (motion blur, smoke, fog, etc) violate this.

| Method | **MEPE all** | MEPE matched | MEPE unmatched |
|---|---|---|---|
| Channel-Flow | 8.0147 | 6.1496 | 34.418 |
| C-Flow+I | 7.6456 | 5.8759 | 30.823 |
| C-Flow+I+C | **7.3330** | **5.5448** | **30.736** |

Table 3.3: **Training Final Sequence, Part 1.** Results on the same 398 non-consecutive, randomly chosen, image pairs of the final sequence of *MPI-Sintel* . C-Flow+I is Channel-Flow with illumination model. C-Flow+I+C is the same, followed by a refinement with a 1-level pyramid Classic+NL.

| Method | d0-10 | d10-60 | d60-140 | s10 | s10-40 | s40 |
|---|---|---|---|---|---|---|
| Channel-Flow | 9.4287 | 7.0531 | 5.4954 | 1.5578 | 7.3484 | 42.2794 |
| C-Flow+I | 9.3090 | 6.7750 | 5.2896 | 1.4564 | 6.8913 | 40.9962 |
| C-Flow+I+C | **8.9132** | **6.3343** | **4.9935** | **1.3041** | **6.1424** | **40.1855** |

Table 3.4: **Training Final Sequence, Part 2.** Results on the same 398 non-consecutive, randomly chosen, image pairs of the final sequence of *MPI-Sintel* . C-Flow+I is Channel-Flow with illumination model. C-Flow+I+C is the same, followed by a refinement with a 1-level pyramid Classic+NL.

solving the linear equation system using the Matlab built-in backslash function. The parameter values used are: $\sigma_{sp} = 1$ and $\sigma_f = 1.2$, $\lambda = 100$, number of bins = 32, $\alpha = 0.45$, $\epsilon = 0.001$.

### 3.5.4 Experiments on the Sintel Test Set

We evaluate our method on the test set of *MPI-Sintel* and the results are shown in Tables 3.5 and 3.6 for the clean pass and Tables 3.7 and 3.8 for the final pass. Numbers in parentheses indicate the ranking on the Sintel site at the time of submission. These tables show comparison to a few methods; see the MPI-Sintel website for the comparison to all the methods submitted to the benchmark, and images of the results. We see a consistent improvement over the baseline Classic+NL, both in the clean and final sets. This is consistent with the experiments on the training set. In addition we compare with LDOF [19], a popular method for dealing with large displacements.

| Method | **MEPE all** | MEPE matched | MEPE unmatched |
|---|---|---|---|
| DeepFlow (6) | 5.377 | 1.771 | 34.751 |
| Channel-Flow (13) | 7.023 | 3.086 | 39.084 |
| LDOF (15) | 7.563 | 3.432 | 41.170 |
| Classic+NL (16) | 7.961 | 3.770 | 42.079 |

Table 3.5: **Select results on the *MPI-Sintel* test set for the clean pass, Part 1**. The simple change in data term improves results over the Classic+NL baseline. See the Sintel website for the full table.

| Method | d0-10 | d10-60 | d60-140 | s10 | s10-40 | s40 |
|---|---|---|---|---|---|---|
| DeepFlow (6) | 4.519 | 1.534 | 0.837 | 0.960 | 2.730 | 33.701 |
| Channel-Flow (13) | 5.411 | 3.236 | 1.918 | 0.624 | 2.791 | 49.021 |
| LDOF (15) | 5.353 | 3.284 | 2.454 | 0.936 | 2.908 | 51.696 |
| Classic+NL (16) | 6.191 | 3.911 | 2.509 | 0.573 | 2.694 | 57.374 |

Table 3.6: **Select results on the *MPI-Sintel* test set for the clean pass, Part 2**. The simple change in data term improves results over the Classic+NL baseline. See the Sintel website for the full table.

The classical formulation with the CR data term largely outperforms LDOF for large displacements without the use of an external matching process. Since our method is based on Classic+NL, it does not benefit from the latest ideas in optical flow. Other methods like DeepFlow [105] are significantly more accurate, even for large motions. Our results suggest, however, that switching from classical brightness constancy to some sort of descriptor constancy may be valuable and we hypothesize that this idea will apply to other methods as well.

### 3.5.5 Experiments on the Middlebury dataset

In order to provide a more complete comparison with existing methods, we also test our method in the training set of the *Middlebury* dataset [8]. The *Middlebury* dataset has been the standard benchmark for optical flow methods for years, but

| Method | **MEPE all** | MEPE matched | MEPE unmatched |
|---|---|---|---|
| DeepFlow (5) | 7.212 | 3.336 | 38.781 |
| Channel-Flow (13) | 8.835 | 4.754 | 42.064 |
| LDOF (15) | 9.116 | 5.037 | 42.344 |
| Classic+NL (16) | 9.153 | 4.814 | 44.509 |

Table 3.7: **Select results on the *MPI-Sintel*, Part 1** test set for the final pass. Channel-Flow again improves over the baseline. See the Sintel website for the full table.

| Method | d0-10 | d10-60 | d60-140 | s10 | s10-40 | s40 |
|---|---|---|---|---|---|---|
| DeepFlow (5) | 5.650 | 3.144 | 2.208 | 1.284 | 4.107 | 44.118 |
| Channel-Flow (13) | 6.757 | 4.566 | 3.657 | 1.292 | 5.349 | 54.648 |
| LDOF (15) | 6.849 | 4.928 | 4.003 | 1.485 | 4.839 | 57.296 |
| Classic+NL (16) | 7.215 | 4.822 | 3.427 | 1.113 | 4.496 | 60.291 |

Table 3.8: **Select results on the *MPI-Sintel*, Part 2** test set for the final pass. Channel-Flow again improves over the baseline. See the Sintel website for the full table.

it only contains small displacements and thus it is not a central experiment for our method.

As we do in the previous experiments, we use the parameter configuration reported by Classic+NL. The MEPE of CR-Flow is 0.287 and the MEPE of Classic+NL is 0.257. We test the statistical significance of these two results and we find them not to be significant. Note that Middlebury motions are small and the value of the CR term for dealing with large motions is not evident here. Figure 3.10 shows the flow fields.

## 3.6 Conclusion

One of the dilemmas of optical flow is that there is a trade-off between the size of the objects and the magnitude of motions that can be estimated. The large motions and complexity of the *MPI-Sintel* optical flow database demand that such trade-

Figure 3.10: **Flow fields from the Middlebury dataset.** The first and third columns show the original images from the *Middlebury* dataset. The second and fourth columns show the optical flow fields computed with our method.

offs be addressed. We have shown how to at least partially address this issue by introducing a channel representation to replace the images used in standard methods. This representation maintains more of the image information under significant blurs.

Our paradigm works with the Classic+NL framework and changes only the data term. This allows us to isolate the effects of this term from other properties of a flow method. We have demonstrated quantitative improvement over the baseline method for a controlled experiment of small regions moving quickly. We have also demonstrated improvement over baseline for the *MPI-Sintel* albedo sequences where brightness constancy holds (except at occlusion boundaries). Given that many of the top-performing methods are based on the variational approach, the channel representation may be potentially very useful for many other flow algorithms as well.

Finally we introduced a simple method to deal with changing brightness, which extends the Channel-Flow method to more complex sequences. This simple method could also be used for other flow algorithms. On the difficult *MPI-Sintel* final test set

we show improvement over the baseline, especially in areas near motion boundaries and in fast moving regions.

# CHAPTER 4

# APPLICATION TO VIDEO STITCHING FOR CREATING LOOPY VIDEOS

## 4.1 Introduction

Finding correspondences between images is also a very useful component for many high-level computer vision tasks. Correspondences help characterizing objects and motion, and so they have been a fundamental part of many different applications like object detection [30, 69] and recognition [72, 77, 44], 3D scene understanding [64, 51, 48], or activity recognition [108, 25, 29]. Good correspondences are also very useful in computer graphics. Applications include view interpolation [111], temporal interpolation (or slow motion), morphing [89], and super-resolution [70].

In this chapter we explore an application of finding correspondences to video stitching for the creation of loopy videos. Loopy videos and animated GIFs have gained tremendous popularity in the last few years with the ease of video capture, and the introduction of video sharing services like Vine.co and Instagram.com. More than 100 million people watch Vine videos every month, and over one billion loops are played daily on Vine alone. [1] The typical length of these videos is surprisingly short – up to six seconds on Vine and 15 on Instagram. This is much shorter than typical YouTube videos, that have a median length of two minutes. [2] These videos are popular in social networks, blogs, digital marketing, music clips and art. Even Youtube has

[1] http://expandedramblings.com/index.php/vine-statistics/

[2] http://www.journalism.org/2012/07/16/video-length/

made a tool[3] to create short loops for some videos. Short loopy videos capture key scene dynamics and can convey a richer meaning than a single photograph, but are more concise, portable, and sharable than long videos. Most such videos are created by cutting a short clip from a longer video. This frequently leads to abrupt changes from the last to the first frame, resulting in an uncomfortable experience when watching them played as a loop. One popular "trick" to avoid this, is to play the video back-and-forth (by concatenating a copy of the video in reverse order). While this alleviates abruptness due to change in location, the change in motion is still abrupt, and it often creates unrealistic motions due to time-reversal.

In contrast, artists, animators and professional photographers, create strikingly hypnotizing micro-videos by perfectly "closing the loop" to seamlessly transition from the last frame back to the first frame. [4] Creating perfectly loopy clips from casual video footage can be highly tedious or even impossible for some videos. It typically involves manually finding the right cut locations in the video clip, and aligning the two ends with professional video editing tools. The goal of our work is to automate these two steps, and make the process of creating compelling loopy video clips significantly easier.

The seminal "Video Textures" work by Schödl et al. [84] proposed an elegant framework to automate this process for specific types of content. They showed that videos with dynamic texture-like characteristics (such as the flame of a candle) often contain multiple moments with similar appearance and dynamics that can be used as transition points for creating infinite loopy videos. The camera and background in these videos are static.

---

[3] http://www.bbc.co.uk/newsbeat/30449728

[4] http://erdalinci.tumblr.com/
http://milosrajkovic.tumblr.com/
http://lillicarre.tumblr.com/

In this work, we generalize the Video Textures framework to handle videos "in the wild". These are typically captured by hand-held devices and contain arbitrary camera motion (including translation and zoom) and complex, non-rigid scene dynamics (including human motion). We focus on one popular type of content: videos of a dominant moving foreground, such as a moving person, animal or an object, in front of a roughly static background (small motion in the background is usually fine) and a moving camera. Motivated by research on visual attention [39] that shows that people have higher tolerance to inaccuracies in the periphery of the point of attention, our key observation is that, in many cases, finding moments where only the *foreground* is similar, is sufficient to produce pleasant loopy videos.

In order to handle such challenging videos we replace both the *analysis* and *synthesis* components of the Video Textures framework with new algorithms. During analysis, we find moments in the input video where the dominant foreground is similar both in appearance and dynamics. We start with a rough segmentation of the foreground in a scene, computed using an automatic method [80] (or in some hard cases with user interaction [6]). We develop a similarity metric based on this segmentation and Non-Rigid Dense Correspondences [45] to robustly assess similarity in the motion and appearance of the foreground between two sets of video frames. In the synthesis step, we propose a method based on Regenerative Morphing [89], a patch-based synthesis method that was designed to morph different images. We augment it to morph between two video clips using linear motion constraints for the background region, second-order motion constraints for the foreground and automatic temporal gap estimation based on the dynamics of the scene. We show compelling results on several challenging videos downloaded from the internet, as well as comparisons to previous methods.

## 4.2 Related Work

In this section we review prior work that is related to ours in two different ways. We describe related work in the application to video loops, and more generally, related video editing tools. We also give a brief review of the family of correspondence methods based on nearest neighbors, since they will be used in our system.

### 4.2.1 Motion Synthesis

The analysis of scene dynamics in videos is a critical component of many video editing tasks, and has been studied extensively in graphics and vision literature. We focus on the techniques that are particularly relevant to our work.

#### 4.2.1.1 Video Transitions

Combining multiple video clips is one of the most common video editing operations, and film editors have developed a taxonomy of the different kinds of transitions (or "cuts") used to achieve this (see Goldman [41] for an overview). While general video editing requires a significant amount of skill and time, it can be (semi-) automated in specific instances. Zheng et al. [110] leverage information in a light field to automatically author cinematic effects from photographs. Kemelmacher-Shlizerman et al. [56] generate smooth animations from personal photo albums by aligning and transitioning between the faces in the photographs. Berthouzoz et al. [15] focus on editing interview footage, and propose a method that uses audio-visual features to automatically find good cut locations.

Most of this previous work is applicable only to specific classes of data (for e.g., faces) and cannot be trivially extended to general video sequences. In contrast, our technique does not make strong assumptions about the content of the input video sequences, and, to our knowledge, is the first general purpose approach for synthesizing realistic video transitions automatically in the presence of camera motion and complex foreground dynamics.

#### 4.2.1.2 Video Morphing

Transitioning between two shots might require morphing between the two (especially when the content is not properly aligned, or has significant differences). There is a significant amount of literature on image morphing [42], and most techniques compute correspondences between images, and construct motion trajectories from these correspondences. Both of these are challenging problems that become especially harder in the presence of complex camera motion and scene dynamics. Previous work has tackled this by relying on user input to specify the region of interest [5, 82], or correspondences between pairs of videos [67]. These methods cannot handle regions without correspondences, as often happens with the backgrounds in our examples. In addition, the synthesized motion trajectories need to be consistent with the motion in the original footage for the morphed result to look natural. Many morphing techniques (for e.g., Shechtman et al. [89]) do not account for this, and produce non-realistic results for general video sequences.

In our work, we compute correspondences between video frames using the technique of HaCohen et al. [45]. We morph the background and the foreground separately to account for the fact that they might move in different ways. In addition, we synthesize background motion trajectories using linear interpolation (or use linear motion constraints when background correspondences do not exist), while using parabolic constraints to synthesize foreground motion trajectories. Unlike previous work, this allows us to handle both moving cameras and fairly general scene dynamics.

#### 4.2.1.3 Video Textures and Cinemagraphs

Video Textures [84, 62, 31, 1] create infinitely looping videos by finding similar frames in a video clip (based on image features), and transitioning between them using morphing. However, these methods were designed to work on videos that are shot by a static camera (or a smoothly moving camera), and where the dynamics are

either local (e.g., a swinging candle flame or flapping flags) or stochastic in nature (e.g., flowing water, fire flames). More recent efforts use spatially varying dynamics to handle multiple motions [68] and to include manual interaction [102], but they require videos captured with static cameras.

Cinemagraphs are a related form of media that lie between video and photographs; the salient objects in the scene remain animated while the surrounding objects are held still. Recent work has proposed interactive tools for their creation [102, 5, 55]. These methods work by using one of the video frames for the background and pasting the moving foreground on top. The inputs to these methods have to be captured using a static camera, the motion is often localized or repetitive in nature, and the methods require some user interaction.

Unlike this previous work, our technique can handle both camera motion and non-stochastic scene dynamics (including highly structured motions like human movement). We are able to achieve this by considering the background and foreground separately while finding good transition points, and aligning and morphing them.

### 4.2.2 Nearest Neighbors Correspondence Methods

Many high-level editing tools in computer graphics are based on finding correspondences between two images or even within the same single image. Some examples are image and video refilling [107], image retargeting or texture synthesis [32, 61]. Correspondences here are used to sample small patches (around $7 \times 7$) of the image and use them to synthesize new parts. This family of methods is loosely known as *non-parametric patch sampling*. Traditional vision techniques like optical flow or sparse correspondences could theoretically be used here. However, the characteristics of the input and the output might differ slightly depending on the application. For example, traditional optical flow methods take as input images of the same scene,

which might not be the case in graphics applications where input images only share part of the image content.

The core element of nonparametric patch sampling methods is a repeated search to find the nearest neighbor (according to some similarity metric) of each patch in one image in the second image or region [11]. This process builds a nearest neighbor field. This is an expensive process and there have been multiple efforts to make it more efficient, involving tree-structures [104, 49, 60] or dimensionality reduction [49, 59]. One of the most successful efforts was the work of PatchMatch [11], which is a randomized search algorithm where the best matches are propagated. This method has led to a variety of applications and extensions, including optical flow itself [10], stereo [18] and many computer graphics problems like denoising, or image editing [11, 12]. In our work, we will also use PatchMatch as one of the components.

## 4.3 Overview

Given an input video $V$, the goal of our method is to use a subset of the original frames, and produce a shorter video, $V_{out}$, whose last frame seamlessly transitions to the first frame, to produce a realistic and compelling video loop. In other words, any differences between the foreground and background appearance (illumination, location, pose) or scene dynamics (velocities and higher location derivatives) between these frames should change as smoothly as possible so that the transition is not noticeable.

We achieve this using a two step process. In the first step – the analysis stage – we analyze the input video to automatically choose the short subset clip that we consider to be most likely to produce a loop with a smooth transition of the foreground. We do this by finding frames in the video sequence that have the most similar foreground regions, up to a global rigid alignment. This results in a frame pair $(a, b)$ that denotes the start and end of the subset clip that we use to synthesize the video loop. While

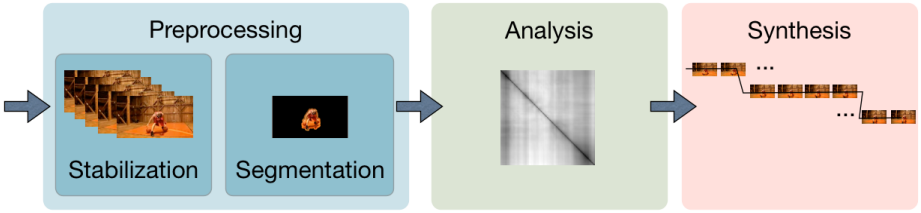



Figure 4.1: **Overview of the algorithm**

the appearance of the foreground in these frames is similar, it needs to be aligned properly for the loop to be seamless. In the second step – the synthesis stage – we morph the frames around this transition point, i.e., frames $V(a+k)$ to $V(a)$ and $V(b)$ to $V(b-k)$, to handle the differences in appearance and motion, and synthesize a smooth transition of both the foreground and the background from frame $b$ back to frame $a$. This results in the output loopy video $V_{out}$. This overview is illustrated in Figure 4.1.

### 4.3.1 Pre-processing

Our system is meant to work on casual videos. These include videos captured with handheld cameras that contain high frequency camera motion due to unintended camera jitter. This makes it difficult to make assumptions about foreground motion. Instead, we use the Warp Stabilization tool in Adobe After Effects (set to smooth motion) to remove the high frequency component of the camera motion. Previous methods [84, 66, 68] focus on subtle foreground motion, and do not handle camera motion. They stabilize the input videos to eliminate background motion entirely or also some parts of the foreground with user interaction [5]. Our method is designed to handle low-frequency camera and foreground motions, and therefore, we stabilize the input video to remove only the high frequency components of the camera motion.

We assume that the input video contains a single dominant foreground motion, and our goal is to create a smooth transition of this motion. A key part is to identify

this region of interest explicitly, and we use the motion segmentation algorithm of Papazoglou and Ferrari [80] to achieve this. We choose this method because it is automatic and is designed with minimal assumptions on the input videos. It consists of an initial estimate based on motion boundaries followed by refinement based on a spatio-temporal extension of GrabCut [81]. The per-frame masks computed by this technique may contain several non-connected components, and may miss some portions of the foreground. We smooth the mask with a median filter, fill in holes by dilating the mask, and choose the main connected component as the foreground. For each frame $V(i)$, we denote the binary mask that we obtain as $M(i)$. For future computation, we also construct a bounding box around it, and call this bounded portion of the image $B(i)$. This notation is demonstrated in Figure 4.2.

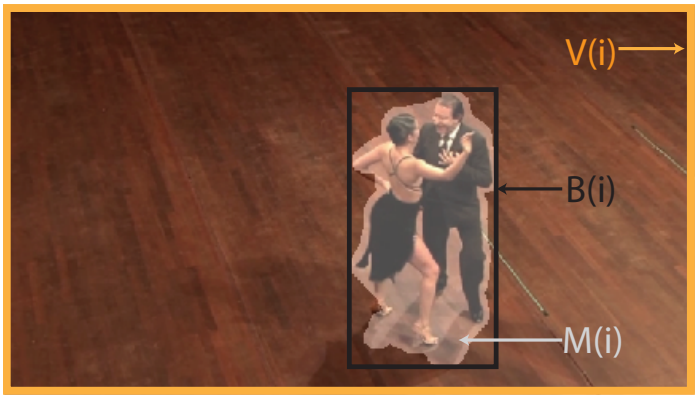



Figure 4.2: **Illustration of notation.** At each frame $V(i)$ we compute a foreground mask $M(i)$ and draw a bounding box $B(i)$ around it.

## 4.4 Analysis – Choosing Transition Points

We analyze the input video $V$ to choose the pair of frames $(a, b)$ that we will use as a transition point. This choice of this transition point plays a key role in the quality of the output. To create the most natural looking video loops, we would like to find the pair of frames where the foreground is the most similar in terms of both appearance

(a) (b) (c) (d)

Figure 4.3: **Illustration of our similarity metric.** (a) and (b) Pair of bounding boxes containing the foreground mask colored light grey used as input to compute the correspondences with NRDC. (c) Confidence mask computed using NRDC coloured in light grey. (d) Foreground mask and confidence mask overlapped. Only the pixels contained in both masks (coloured in the brightest grey) are used for the for computing the similarity.

and motion. Our method for choosing this pair of frames is inspired by the work of Video Textures [84]. In particular, we choose this pair of frames by maximizing

$$(a, b) = \arg\max_{(i,j)} S_{app}(i, j) + S_{motion}(i, j), \tag{4.1}$$

where the terms $S_{app}$ and $S_{motion}$ refer to the similarity in appearance and motion respectively, and are computed on the bounding box of the foreground in each frame, $B(i)$.

### 4.4.1 Appearance Similarity

There are a number of different metrics to assess the similarity of the foreground of two frames. Given the segmentation computed previously, we can use pixel-wise metrics like the sum of squared differences (SSD) of color values, like [84], but restricted to the bounding boxes. However, these pixel-wise metrics are sensitive to transformations and deformations. A small inaccuracy of the segmentation mask might lead

to a shift in the bounding box, and thus a low similarity score, even if the foreground appearance is very similar. Even if the bounding boxes are well aligned, or if we used a representation robust to small shifts like DFs, still non-rigid deformations of the foreground (due to motion) may lead to low similarity scores.

Instead, we find correspondences between the foregrounds of two video frames and use it as a proxy for appearance similarity. This is based on the observation that the more similar the foregrounds are, the more correspondences we are likely to find between the frames, and the more similar these correspondences are likely to be. While there are many methods for finding such correspondences, in our work, we use the Non-Rigid Dense Correspondence (NRDC) [45]. We choose NRDC for several reasons: it is robust to changes in illumination, it is designed for pairs of images where only part of the content is shared and unmatched regions do not hurt the estimation of the correspondence, and it provides a confidence map that indicates which pixels have a correspondence. Section 4.4.4 describes the NRDC method in detail.

Given two bounding boxes $B(i)$ and $B(j)$, we compute NRDC and obtain a confidence map, that indicates whether a pixel in $B(i)$ has found a correspondence in $B(j)$. Our proposed similarity is the ratio of foreground pixels that are shared between the bounding boxes. In other words, we use the proportion of the number of pixels with valid correspondences to the total number of pixels in the foreground. We compute this as

$$S_{pair}(i,j) = \frac{\sum_{\boldsymbol{x}} \gamma(B(\boldsymbol{x},i), B(\boldsymbol{x},j)) \odot M(\boldsymbol{x},i)}{\sum_{\boldsymbol{x}} M(\boldsymbol{x},i)}, \tag{4.2}$$

where the function $\gamma(\cdot)$ returns the correspondence confidence from the NRDC algorithm (0 indicates that there is no confident match, 1 if there is absolute certainty about the match), $\odot$ is the element-wise product and $M(\boldsymbol{x},i)$ is pixel $\boldsymbol{x}$ of mask $M(i)$. Bounding boxes are scaled to a resolution of $200 \times 200$ for this similarity measure. This computation is illustrated in Figure 4.3.

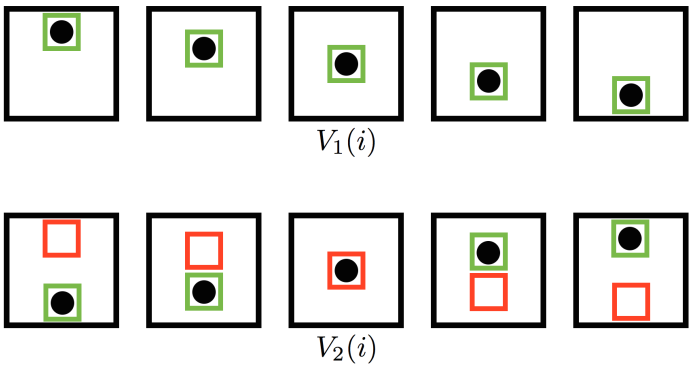



Figure 4.4: **Motion similarity**Given frames $V(i)$ and $V(j)$, if we used only the appearance similarity (Eq. 4.3) within the original bounding boxes (green boxes), we would transition between two clips with different foreground motion, leading to a semantically wrong motion. Transforming the bounding boxes in the second clip according to the motion in the first (red boxes) and using 4.4 results in a low score in this case.

To increase robustness and temporal coherence we seek transition points where a *sequence* of frames have high similarity. Similarly to [84], we capture similarity across a range of frames by summing it over a temporal window around a considered transition point:

$$S_{app}(i,j) = \sum_{-l \leq k \leq l} w(k) S_{pair}(B(i+k), B(j+k)), \tag{4.3}$$

where $w_k$ is a Gaussian weight with a standard deviation of 1, and $|l|$ is the size of the neighborhood (in our case $|l| = 5$).

### 4.4.2 Motion Similarity

NRDC finds good non-rigid correspondences, and thus the previous term ensures similar appearance within the segmented foreground across a set of frames. This is usually sufficient to ensure temporal smoothness for relatively static foregrounds with simple motions such as the ones used in [84]. However, $S_{app}$ only ensures good similarity exists across several frames, and does not take into account the global motion of the objects relative to their backgrounds, as often happens in the examples

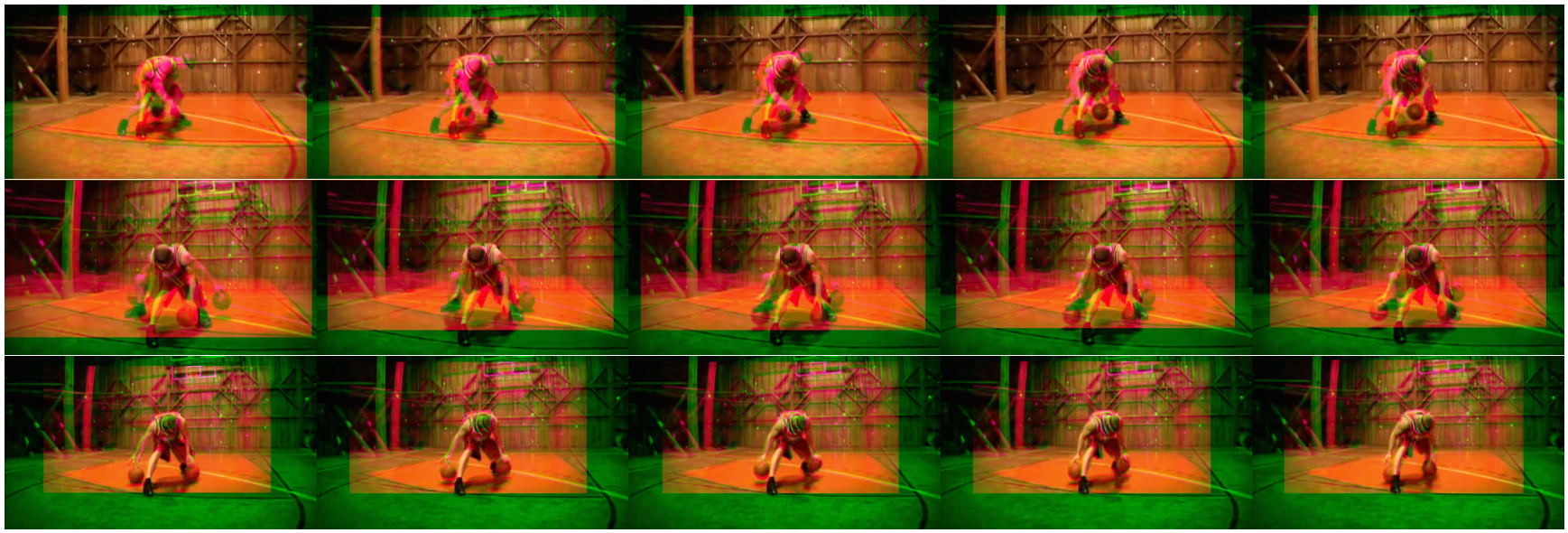


Figure 4.5: **Comparison of similarity metrics.** We compare the matched frames computed using different similarity metrics and show the chosen transition frames (center) as well two preceding and subsequent frames, after rigid alignment. To visualize the quality of the alignment, each image is generated by using the red and blue channels from one frame and the red channel from the other frame; the areas that are well aligned appear naturally colored, while the misaligned regions appear oddly colored (green or pink). Using SSD over the whole frame (top) leads to results where the foreground is completely misaligned. Using SSD only in the bounding box (middle) is an improvement. Our method (bottom) is significantly better – the foreground is almost perfectly aligned at the cost of errors in the background.

we aim to handle. For example, in a repetitive motion like a child jumping on a trampoline, a series of frames where the child is going upward could be matched to a series of frames where the child is going downward, because the correspondences subsume the differences in the motion between these two scenarios. If we were to stitch the clips, this would lead to a non-realistic change in the global motion.

To avoid this problem, we add another term to the objective function that captures the relative *rigid* motion of the foreground object with respect to the background. We do this by transforming the bounding boxes in one of the videos to follow the relative motion of the other video, and recomputing the similarity using the new bounding boxes. For a pair of frames $V(i), V(j)$ we do this by computing the transformation $\hat{T}$ that best aligns the foreground in $V(i)$ to the one in $V(j)$. We parameterize this transformation using translation and scale (more degrees of freedom led to a less robust estimation and such transformations are typically less common within the same video). The transformation $\hat{T}$ aligns the video frames $V(i)$ and $V(j)$. We then transform the location of the bounding boxes $B$ at a few frames before and after $j$, i.e., $B(j \pm k)$, to compute the transformed bounding boxes, $B'(j \pm k)$. If the global motion of the foreground is similar in both videos, then the original bounding boxes $B(j \pm k)$ and the transformed ones $B'(j \pm k)$ will be similar, and therefore, $B'(j \pm k)$ will overlap largely with the foreground. However, if the motion in both clips is not similar, then $B'(j \pm k)$ will mostly contain background pixels. This process is illustrated in Figure 4.4. We use this transformed bounding box to compute the motion similarity between the two frames as:

$$S_{motion}(i, j) = \sum_{-l \leq k \leq l} S'_{app}(i + k, j + k), \tag{4.4}$$

where $S'_{app}$ computes the appearance similarity as in Equation 4.3 but using the transformed bounding box $B'(j)$ for the second frame. We use $l = 3$ frames to compute the motion similarity.

### 4.4.3 Optimizing the Similarity Function

Optimizing Eq. 4.1 can be expensive, since it requires computing dense correspondences between every possible pair of frames. In order to make the method more efficient, we make two important approximations. First, it is important to be very precise with the choice of the transition point, and being one or two frames away from the optimal transition point can have a big effect on the final result. This makes it important to explore every possible transition point. However, most of them are actually bad candidates and the similarity behaves smoothly around the good ones. We take advantage of this by using a *coarse-to-fine* strategy to find the transition points. We evaluate $S_{app}$ at every fifth pair of frames, and then we choose the top 40 to explore at the finest temporal resolution. Second, computing $S_{motion}$ is computationally expensive because for each pair of frames it involves computing NRDC an order of $\mathcal{O}(n^3)$ number of times. To avoid such a large of computations, we use a *two-step* process: we first compute the top 20 candidate pairs using the $S_{app}$ score, and compute $S_{motion}$ and evaluate the full similarity metric only on these. Finally, we constrain the search for transition points to frames that are at least 20 frames apart (i.e., $|i - j + 1| \geq 20$) to avoid short loops.

The total computation time varies depending on the size of the input video. In our experiments the average computation time to optimize the energy is around 2 hours. Most of this time is spent computing correspondences with NRDC, and thus a faster alternative would greatly improve the efficiency of our method.

We compare the performance of our foreground similarity metric with two baseline techniques on one of our examples. These techniques are: first, SSD computed on the entire frame [84], and second, SSD computed only in the masked region. These results are shown in Figure 4.5. Our method achieves significantly better spatio-temporal alignment of the foreground. Our technique for handling residual misalignments of the foreground, as well as differences in the background are discussed next.

### 4.4.4 Non-Rigid Dense Correspondence

This method is designed to find correspondences between images that share some content but also may contain different scenes or may be taken with different cameras. Given two images, NRDC computes a geometric and a photometric transformation at each overlapping patch, a global parametric color transformation and a confidence map. These are computed in a coarse-to-fine scheme, iterating the following four steps.

1. Computing the nearest neighbor field. This is performed using the generalized PatchMatch algorithm [11]. This algorithm starts by randomly assigning correspondences between patches. The transformation of each patch is compared to its neighbors' transformations. The transformation that produces the closest correspondence is chosen. Then a random search is performed for improvements. These two steps are repeated a few times. Figure 4.6 illustrates the process.

2. Aggregating consistent regions. In this stage the goal is to compute the regions that have a reliable match. A consistency error is computed between adjacent pixels. If this consistency error is smaller than a threshold, these two pixels are considered to be consistent. Using these links, the connected components are computed. Components are discarded if they are very small or inconsistent. Figure 4.7 describes the consistency metric.

3. Fitting a global color transformation. Given the correspondences, a color transformation is fit. This transformation is a monotonic curve in each channel of RGB space, followed by a linear transform to account for saturation.

4. Adapting the search range. This stage is designed to avoid overfitting, given the large number of parameters that are optimized at each patch. To avoid this, the search range of the transformations is narrowed over time.

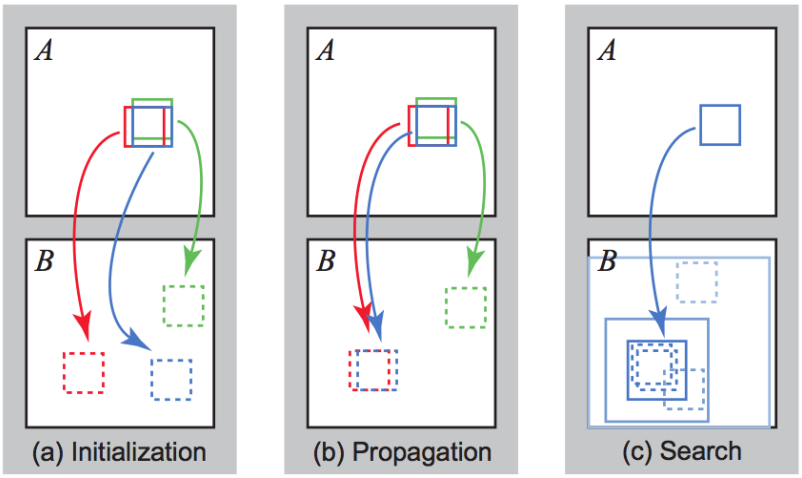



Figure 4.6: **Illustration of PatchMatch stages.** From Barnes et al. [11]. Given two images $A$ and $B$, correspondences between the two images are computed using the randomized nearest neighbor algorithm. The stages are the following. (a) Patches in $A$ initially have random assignments. In the figure, color encodes assignment: the red patch drawn with a solid line in $A$ is assigned the red patch drawn with dotted line in $B$, and so on. (b) The blue patch checks above/green and left/red neighbors to see if they will improve the blue mapping, propagating good matches. (c) The patch searches randomly for improvements in concentric neighborhoods.

Further details and comparisons to other correspondences methods like SIFT can be found in the original paper [45].

## 4.5 Synthesis – Video Morphing

During the synthesis stage our method takes the transition point $(a, b)$ ($b > a$, w.l.o.g.) resulting from optimizing Eq. 4.1, and produces an output video $V_{out}$ that closes the loop seamlessly. Our goal is to ensure that the motion and appearance foreground change smoothly, and that there is a reasonably smooth transition of the background. Ensuring a good transition of both is in general not possible, and we emphasize the smoothness of the foreground. This is based on the assumption that most of the viewer's attention is focused on the foreground, and, therefore, a smoother foreground transition will lead to a more compelling transition, as long as the transition of the background is not too abrupt. In this section we describe the

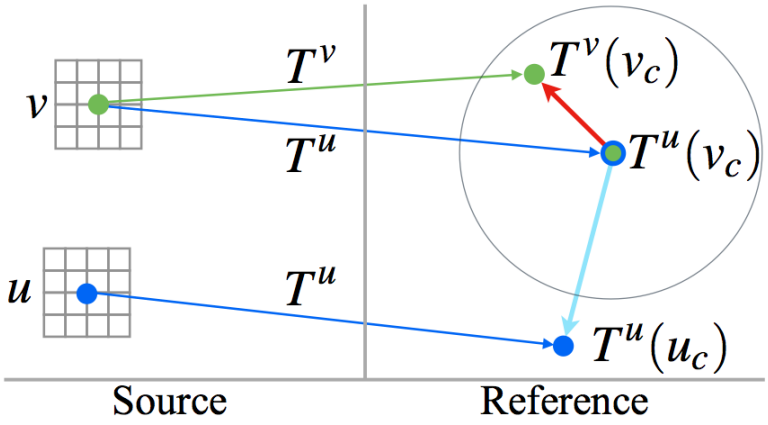



Figure 4.7: **Illustration of NRDC consistency measure**. From HaCohen et al. [45]. Given two images $S$ and $R$, NRDC computes correspondences from the patches in the source $S$ to the reference $R$ image, and a transformation for each of these correspondences. Consider a pair of patches $u, v \in S$ where the transformations for each of the correspondences are $T_u, T_v$. Let $v_c$ denote the coordinates of the center of patch $v$. If the two patches are matched consistently we expect the distance between $T_v(v_c)$ and $T_u(v_c)$ to be small. However, for this measure to be meaningful, the distance should be normalized. This leads to the consistency measure: $C = \frac{||T^v(v_c) - T^u(v_c)||^2}{||T^u(u_c) - T^u(v_c)||^2}$

video morphing step of our method: first, we align the foregrounds using a global, coarse, and rigid alignment of the videos, and second, we use a local, detailed, non-rigid alignment of the foreground and background using correspondence guided patch-based synthesis.

Throughout this process we use NRDC to obtain correspondences for alignment, although there are other options like optical flow. There are two reasons for our choice. First, we used NRDC to evaluate the similarity of frames and choose the optimal transition point. As a result, the transition point is guaranteed to have enough good correspondences from NRDC, making alignment and morphing more likely to succeed. Second, we tested different optical flow methods in our dataset, and they produced significantly worse results. These experiments and a discussion on when to use each method are described in Section 4.8.

### 4.5.1 Global Rigid Alignment via Camera Planning

Since our similarity measure is based on correspondences, it does not guarantee the alignment of the foregrounds in frames $V(a)$ and $V(b)$. We first handle gross global differences in the alignment by constructing a virtual camera path. As in Eq. 4.2, we compute a transformation $\hat{T}$, consisting of a translation and a scale, that best aligns these frames. To ensure that the transition is smooth, we spread this transformation over a window of frames after $V(a)$ and before $V(b)$. For this we fit a cubic spline $f(t)$, of length $L$ and with zero first and second derivatives at the end points, for each of the three transformation parameters (2-D translation and 1-D scale), sample these splines to compute interpolated transformations, and apply them to the frames.

### 4.5.2 Local Non-Rigid Alignment via Morphing

Applying the transformations to the start and end of the sub-clip gives us two sets of frames around the transition point that are rigidly aligned, but there might still be mis-alignments because of small changes in appearance or non-rigid motion in the scene. As a result, naïve approaches like concatenation or cross-fading would create artifacts like abrupt cuts or ghosting, respectively. Traditional morphing (TM) techniques use a combination of optical flow and cross-fading to alleviate this problem. However, our input videos are often noisy, and have large displacements, and this leads to poor flow results that, as illustrated in the results section, manifest in poor synthesis results.

Instead, we use a patch-based morphing technique based on the Regenerative Morphing work of Shechtman et al. [89]. Intermediate frames are synthesized using patches from two source frames while preserving local similarity to the sources and to consecutive frames for temporal coherence. Regenerative Morphing has demonstrated good performance on scenarios ranging from interpolating nearby views to morphing

entirely different images, and we build on it to smoothly morph our sub-clip around the transition point. Section 4.5.5 describes this method in detail.

We could use Regenerative Morphing in the most straightforward way, i.e., given the transition point $(a, b)$, interpolate between frames $V(a+k)$ and $V(b-k)$, where $2k$ is the size of the window we consider to be wide enough to create a smooth transition. This is a purely image-based approach that does not leverage the fact that we have multiple frames ($V(a : a + k)$ and $V(b - k : b)$) near the transition point that are in close alignment. There are two main problems with this approach, and we propose two generalizations of the algorithm to tackle them:

### 4.5.3 Parabolic Motion

One useful property of Regenerative Morphing is the ability to incorporate correspondences between frames as constraints. When correspondences exist, pixels in one source can be constrained to move on a linear path towards their corresponding location in the second source. Regions without correspondence are reconstructed in a plausible way that satisfies the correspondence constraints. However Regenerative Morphing has been demonstrated on image interpolation of relatively nearby views, where linear motion of corresponding pixels is a plausible assumption. In our case, the input is a pair of image sequences with complex camera motions and non-rigid scene dynamics (for e.g., moving people). The linear motion assumption does not hold in these cases and leads to unrealistic dynamics during the transition. To tackle this problem, we generalize this method to use a *parabolic motion* at each pixel. The parabola is computed by using NRDC correspondences between 4 frames: $V(a)$-to-$V(b)$, $V(a)$-to-$V(a+k)$ and $V(b-k)$-to-$V(b)$. We fit a parabola to these correspondences and sample along it, to generate the motion at each constrained pixel, and constrain the pixels using random sampling. As shown in Figure 4.8, this leads

to more realistic dynamics of the foreground in the morphed frames, compared to the linear constraints originally used in Regenerative Morphing.

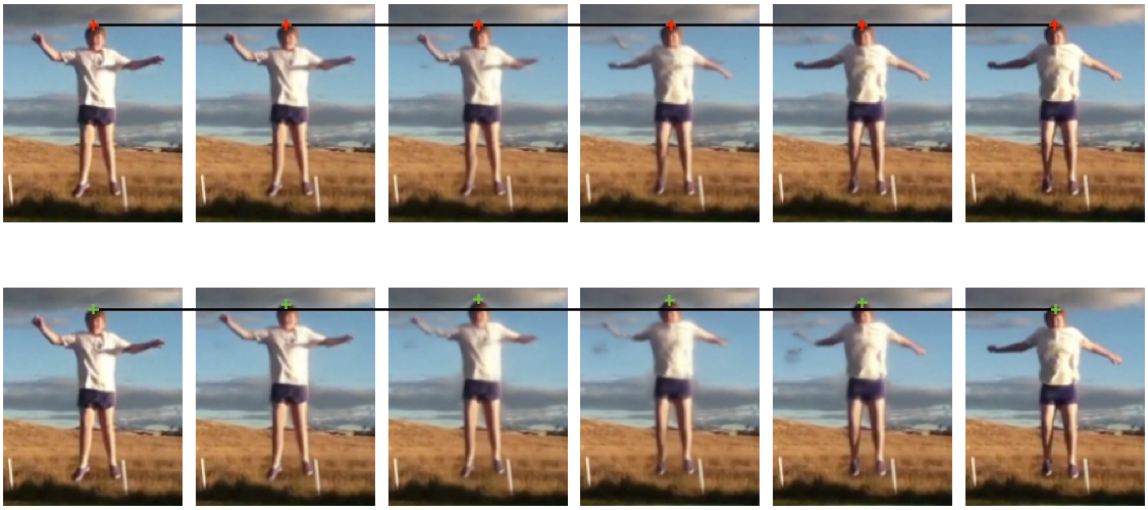


Figure 4.8: **Linear (top) vs. parabolic (bottom) constraints.** The transition happens around the peak point during the jump. The parabolic constraints capture the correct dynamics and lead to a natural motion during the transition, while the linear constraints make the kid "freeze" weirdly in the air during the transition.

### 4.5.4 Separate Foreground and Background Morphing

Regenerative Morphing uses a fixed temporal window size (the number of frames to be interpolated) for the entire frame. In our case, the foreground is very well aligned due to the choice of the optimal transition point and the virtual camera path, and thus a small number of frames is enough to produce a smooth transition of the foreground. However, the background can be very different between frames $V(a)$ and $V(b)$. Regenerative Morphing has been shown to perform well at morphing significantly different content when the transition window is sufficiently long. However, if we used the same short temporal window for the background, the foreground would move naturally but the background would change abruptly, producing an undesired transition. We address this problem by morphing the foreground in a first stage, and the background in a second stage with a different window size.

The size of the window for morphing the *foreground* is 4 frames for all our examples, i.e., we morph between frames $V(a+2)$ and $V(b-2)$. However, recall that in order to morph the foreground we need good quality correspondences. Sometimes the foreground moves very quickly (for e.g., see the **trampoline** video), and there are no correspondences between neighboring in the foreground. In this case it is not possible to morph, and we use a simple concatenation of the foreground. Since NRDC provides a confidence map for each correspondence, we use this as a proxy to decide whether to morph or not. In practice it is a rare case, since most often neighboring frames do contain correspondences.

The size of the window for morphing the *background* is computed automatically for each video. The criterion to choose the starting and ending frames is that the velocity of the camera (both direction and magnitude) should change as little as possible when introducing the morph. More specifically, we explore pairs of frames $V(c)$ and $V(d)$, around the transition point where $c < a, d > b$. For each pair of possible frames, we compute the camera motion at those frames in the original video. We also compute the camera motion that morphing would induce. We then choose the pair of frames where the induced motion is most similar to that in the original video. As in the case of the foreground, it is possible that there are not enough correspondences to compute a camera motion between frames after $V(a)$ and before $V(b)$. In this case we cannot assure that the morph will look realistic. This scenario is more common than in the case of the foreground, because $V(a)$ and $V(b)$ were not chosen to match the background, and typically occurs because the camera is moving quickly. In this case, we impose *motion constraints* on the background, so that it moves at the same velocity as before and after the transition. As in the case of the foreground we use the NRDC confidence map to decide whether to use motion or correspondences constraints.

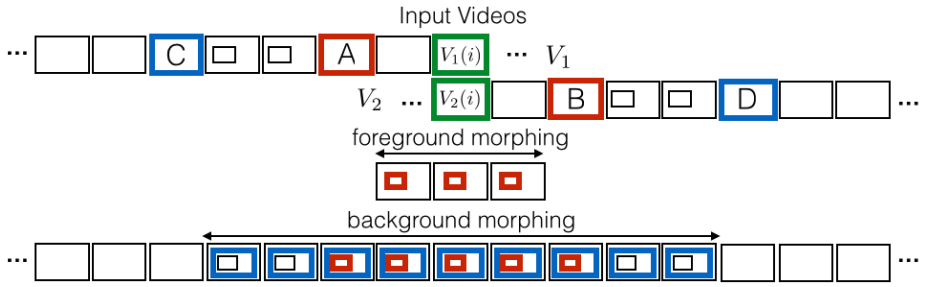



Figure 4.9: **Overview of video morphing**. To illustrate our morphing algorithm, we show the same video sub-clip as two separate clips that are aligned at the transition point; the start of $V_1$ and the end of $V_2$ are the same, and we want to make the loop seamless at the transition point. We use frames $A$ and $B$ around the transition point to synthesize the foreground with parabolic motion constraints . We then use a longer window – frames $C$ and $D$ – to synthesize the background using linear motion, using the previously computed foreground as a constraint. This is enforced by pasting in the foreground during the optimization, and is signified with the small red boxes. For frames where we synthesize background but not the foreground (i.e.: frames between $C - A$ and between $B - D$), the foreground constraint is simply the original segmented foreground from the video. The background is synthesized with linear constraints, based on correspondences if there are enough of them or using motion based constraints otherwise.

The full process of integrating foreground and background morphing is described in Figure 4.9. Once the foreground is morphed, we paste it during the background morph at each synthesis iteration of the coarse-to-fine process used by Regenerative Morphing, ensuring that the foreground is used as a constraint for the synthesis of the background.

We compare the performance of our morphing technique to traditional morphing and to the morphing part of video textures [84]. Traditional morphing is designed to morph two still images. We chose the two frames around the transition (i.e., $V(a)$ and $V(b)$) and warp them one towards the other using optical flow [96] followed by a gradual cross-fading. Video textures can morph two (or more) overlapping sequences. In case of two sequences, it warps the second clips to the first (again, using [96]) during the first half of the transition, and warps the first to the second, followed by gradual cross-fading. Figure 4.10 shows how our method overcomes the residual

Figure 4.10: **Comparison of morphing techniques.** Top: Video Textures morphing. Middle: Traditional morphing. Bottom: Proposed method. Note that the Video Textures morphing is presented for the transition that was chosen by their method, while the other two show a morph for our transition.

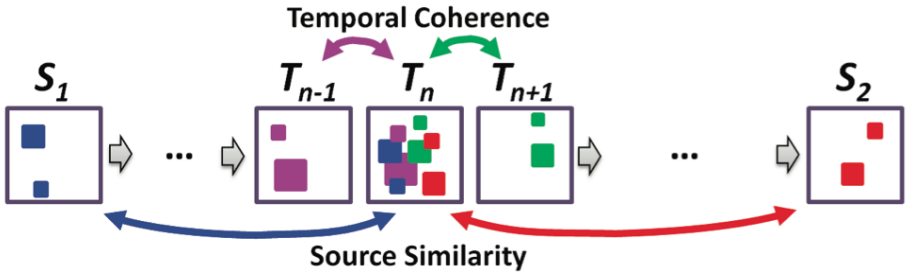



Figure 4.11: **Notations and schematic representation of the regenerative morphing objective.** From Shechtman et al. [88]. The objective function encourages local similarities between each frame $T_n$ and its two neighboring frames $T_{n-1}$ and $T_{n+1}$ (Temporal Coherence), as well as to the two sources $S_1$ and $S_2$ (Source Similarity)

misalignments and results in a clean morph while the other two methods suffer from "ghosting" artifacts due to limitations of traditional optical flow and warping methods in this challenging case.

### 4.5.5 Regenerative Morphing

In this work, the morphing is posed as the optimization of an energy function that contains two terms: *source similarity* and *temporal coherence*. More formally, given two input image sources $S_1$ and $S_2$, they compute $N-1$ intermediate frames ${\{T_n\}_1}^{N-1}$, by optimizing

$$E_{morph}(T_1, ..., T_{N-1}) = E_{SourceSim}(T_1, ..., T_{N-1}) + \beta E_{TempCoher}(T_1, ..., T_{N-1}). \quad (4.5)$$

The first term, encourages the synthesized frames to be similar to the sources. The intuition is that reusing actual patches will generate realistic images. The second term encourages synthesized frames to be similar to their temporal neighbors, to achieve smoothness of appearance and motion. This process is illustrated in Figure 4.11.

The similarity metric between frames is based on the existing work called *Bidirectional Similarity* [91], which has been used for image and video summarization.

This metric has two components: the completeness term that ensures that the output image contains as much visual information from the input as possible, and the coherence term that ensures that the output is coherent with respect to the input and that new visual structures (artifacts) are penalized.

Formally, the distance measure is defined simply as the sum of the average distance of all image patches (e.g., 7x7 pixels) in S to their most similar (nearest-neighbor) patches in T and vice versa:

$$\begin{aligned} d_{BDS}(S,T) &= d_{Complete}(S,T) + d_{Coher}(S,T) \\ &= \frac{1}{N_S}\sum_{s\in S}\min_{t\in T} D(s,t) + \frac{1}{N_T}\sum_{t\in T}\min_{s\in S} D(s,t), \end{aligned} \tag{4.6}$$

where distance $D$ is the sum of square differences in LAB space of patch pixel values. This distance measure captures the distance between two images. However, in the case of morphing, the goal is to use similarity between an image and to two different sources. There are a few different generalizations of the BDS metric that are useful in the morphing case:

#### 4.5.5.1 $\alpha$-Blended Bidirectional Similarity

A straightforward way of combining the similarity to two different images is using a convex combination of the two:

$$d_{\alpha BlendBDS}(T,S_1,S_2,\alpha) = \alpha d_{BDS}(S_1,T) + (1-\alpha) d_{BDS}(S_2,T). \tag{4.7}$$

This can also be applied to each of the terms in Eq. 4.6:

$$d_{\alpha BlendCoher}(T,S_1,S_2,\alpha) = \alpha d_{Coher}(S_1,T) + (1-\alpha) d_{Coher}(S_2,T). \tag{4.8}$$

and

$$d_{\alpha BlendComplete}(T, S_1, S_2, \alpha) = \alpha d_{Complete}(S_1, T) + (1 - \alpha) d_{Complete}(S_2, T). \quad (4.9)$$

Rewriting Eq. 4.8:

$$d_{\alpha BlendCoher}(T, S_1, S_2, \alpha) = \frac{1}{N_T} \sum_{t \in T} \left( \alpha \min_{s1 \in S1} D(s_1, t) + (1 - \alpha) \min_{s_2 \in S_2} D(s_2, t) \right). \quad (4.10)$$

What this would mean is that at each pixel $t$, the method would find the most similar patch in $S_1$ and in $S_2$ and do a weighted average of the two. Since the two sources are slightly different, the result of this will be blurry due to misalignments and changes in illumination. To avoid this they introduce a new term, that encourages patches to belong to one source or the other.

#### 4.5.5.2 $\alpha$-Disjoint Coherence

Adding this new term, the equation for coherence is:

$$d_{\alpha DisjCoher}(T, S_1, S_2, \alpha) = \frac{1}{N_T} \sum_{t \in T} \min \left( \min_{s_1 \in S_1} D(s_1, t), \min_{s_2 \in S_2} D(s_2, t) + D_{bias}(\alpha) \right), \quad (4.11)$$

where $D_{bias}$ is such that the portion of patches taken from $S_1$ will be $\alpha$ times the area of $T$.

Using these similarity metrics, we can now expand on the terms of the morphing objective function, which are:

$$E_{TempCoher}(\{T_n\}_1^{N-1}) = \sum_{n=1}^{N-1} d_{\alpha BlendBDS}(T_n, T_{n-1}, T_{n+1}, 0.5) \tag{4.12}$$

$$\begin{aligned} E_{SourceSim}(\{T_n\}_1^{N-1}) = \sum_{n=1}^{N-1} & d_{\alpha BlendComplete}(T_n, T_{n-1}, T_{n+1}, \frac{n}{N}) \\ & + d_{\alpha DisjCoher}(T_n, T_{n-1}, T_{n+1}, \frac{n}{N}) \end{aligned} \tag{4.13}$$

## 4.6 Experiments

We test our algorithm on a set of videos collected from the Internet, and compare our results to previous work. Figures 4.12 and 4.13 show results on each of these videos.

- **Basketball:** This example contains large changes in scale, which illustrates the value of using the virtual camera path because morphing produces much better results when the foreground is aligned. It also contains motions that are not linear at all, such as the bouncing balls, which illustrates the need for parabolic motions

- **Trampoline:** In this video, we show the importance of our similarity metric focusing on the foreground. Since the foreground occupies a fairly small portion of the frame, a similarity metric that does not take into account separation between foreground and background would be dominated by the background. In addition, in this video morphing the background is particularly challenging because the scene moves quickly. In this case motion constraints are used to maintain the same motion during the morph.

- **Tango:** This example contains a very slow detailed motion of the foreground, where a linear morph produces unlikely motions. Also, the floor moves very slowly with respect to the camera. In such cases it is very important to use a

large window size for the background, that allows the camera motion to stay slow and smooth. In our case, the method uses a window size of 17 frames.

- **Skydiving:** In this example it is again very valuable to use a metric that separates between foreground and background, because otherwise the energy is dominated by the horizon in the background. During synthesis, traditional morphing fails because flow estimation is unreliable due to the low texture of the sequence.

- **Violin:** This video contains an almost fixed camera, with detailed and structured foreground motions. In this case the parabolic motion produces a more realistic result than traditional morphing.

- **Dog:** In this video the camera is fixed, but the foreground object moves from one side of the frame to the other. Because of the small foreground overlap across well-separated frames, VT picks a poor set of transition frames. This video also illustrates the importance of the virtual camera path, because in the case of TM, where the clips that are not aligned, the foreground is morphed and displaced simultaneously, creating an unrealistic effect.

- **Ski-slalom:** The skier in the foreground of this video moves from side to side, and the camera undergoes a significant amount of motion. The frame is dominated by the background, and VT will not find good transition points that align the skier well. Our method is able to produce a nice loop of the skier using parabolic motion, while interpolating the mountains in the background using linear motion constraints.

- **Back-flip:** This video depicts a gymnast performing back-flips in front of a background that changes significantly. While our method produces a natural

look of the gymnast's motion, the synthesized background has artifacts because of the large differences between the start and end of the clip.

We compare to two different methods. First, we compare to Video Textures (VT) because it is the most similar end-to-end method. To make the task easier for this method, we stabilize the input to have no motion at all whenever the input video allows it. Their analysis technique often produces poor transition points (because it use full-frame similarity) and this conditions the result of the synthesis stage, making it very difficult to evaluate the contribution of our synthesis method. For this, we compare to a second method, where we use the transition point from our similarity metric, and then use traditional morphing (TM), i.e., warping using optical flow and blending using cross-fading, instead of our own morphing algorithm. Results for some of the videos are shown in Figs 4.14, 4.15, 4.16 and 4.17.

## 4.7 Limitations

Our method has a few limitations. First, we assume the foreground (whether a person, group of people, etc.) is located in one contiguous regions and has roughly consistent motion which is different from the background. Second, although our method can operate fully automatically in many real-world cases, it relies on a few computer vision components that don't always work ([45, 80, 89]). The most fragile component is the automatic foreground segmentation [80] which we replace with an interactive method in a few examples where the foreground is not distinct enough. We expect segmentation methods to continue improving in future, leading to a higher success rates of our method. Third, we assume the background is roughly static and has similar visual properties in the two clips or else Regenerative Morphing [89] produces noticeable artifacts; the **back-flip** result (Figure 4.13 bottom) is our worst result in that respect.

## 4.8 Optical Flow and Object Tracking on Casual Videos

Our method contains several methods as building blocks, and they could be replaced by different methods. In this section we relate the methods chosen to those presented in Chapters 2 and 3.

### 4.8.1 Object Tracking on Casual Videos

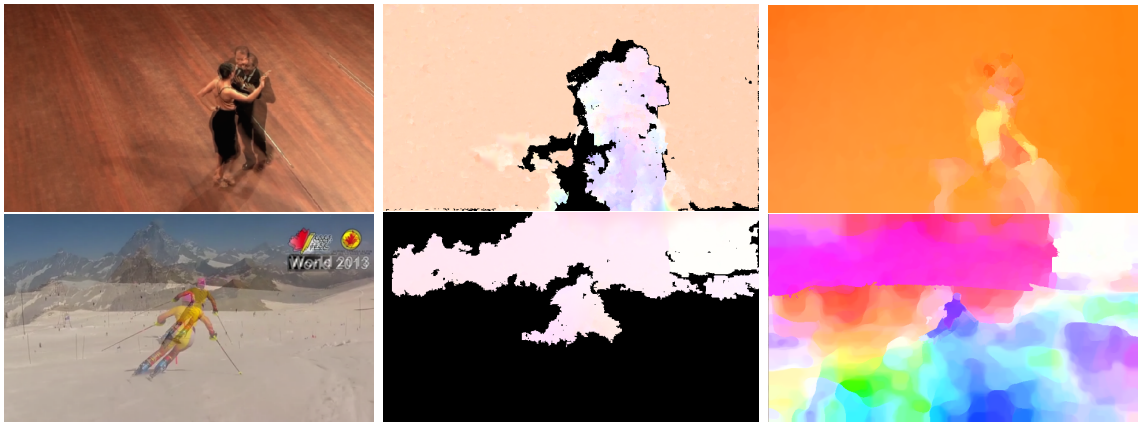
Since the goal of our system is smooth foreground activity, a key part is to identify this region of interest explicitly. Our initial approach was to use our object tracker for this stage. Then we would compare the content within bounding boxes to assess the similarity between frames. Our videos are well suited for a general object tracker: they rarely contain occlusions, they are fairly short, which avoids drift, and there is a single foreground motion.

However, this approach has three important problems: initialization of the tracker, changes in scale and tightness of the bounding box. One of the most interesting goals of our method is to be fully automatic, and so we want to avoid manual initialization of the region of interest which is a requirement for object tracking. Naive approaches to initialization, like simple clustering of motion vectors, are too approximate and often lead to incorrect bounding boxes. Changes in scale occur often in our input videos, for example as a result of zooming in and out of the scene. Since our method does not model scale, it is not expected to work well in these cases. Finally, the foreground object is often a human, which is poorly approximated as a bounding box, resulting in a template that is not tight to the object and contains too much background.

### 4.8.2 Optical Flow on Casual Videos

Dense, non-rigid correspondences for morphing are often computed using optical flow, like in Video Textures [84]. However, our suite of videos shows to be too challenging for current optical flow methods. They contain large changes in illumination,

compression artifacts, and very long displacements (larger than in the case of Sintel). We compare our optical flow method to NRDC in a few videos. Results are shown in Fig. 4.18. Although there is no ground truth, one can notice that optical flow produces poor results in the **tango** sequence, where the foreground people is estimated to move together with the background, presumably due to the regularizer. In the **ski** sequence a large part of the foreground is missed, for example the limbs of the skier.




(a) (b) (c)

Figure 4.18: **Comparison between optical flow and NRDC in casual videos. (a):** Pair of input images overlaid, from the sequences **tango** (above) and **ski** (below). **(b):** Correspondences computed with NRDC. Pixels in black do not have correspondences. **(c):** Correspondences computed with our optical flow method.

Given these experiments, it is interesting to consider when is it better to use NRDC, optical flow or tracking. All methods have strengths and weaknesses that make them better suited for different kinds of applications. Based on our experience, we summarize what tools to use depending on the scenario.

- **Size of displacement:** Short displacements are best captured by optical flow. Results provide sub-pixel accuracy, and in displacements under 10 pixels they produce errors of average 1 pixel. Very large displacements are still a challenge for optical flow methods. Patch-based algorithms like NRDC use randomized

search, which does not have such strong bias towards small displacements. Thus, long displacements can be better captured, although results are not as precise.

- **Content of the scene:** If the input images only share part of the content, or there are strong changes in the lighting of the scene then it is better to use NRDC, because it is more robust to these changes and it is not affected by wrong guesses. However, if most of the scene content is shared between the images, then optical flow will again provide smoother, and more accurate results. We do not show results using NRDC on optical flow benchmarks, like *Middlebury* or *Sintel*, which would be interesting. However, given the correspondences from NRDC, our guess is that they would not score very high. Since there is no regularizer, unmatched regions would have a high error (their motion would be arbitrary).

- **Motion boundaries** In general, patches at boundaries are different between the images. Therefore, no matches would be found in these regions, and the error near the boundary would be expected to be high. Our optical flow method produces low errors at boundaries, due to the smoothing in DF space and the weighted median filtering.

- **Characteristics of the deformation:** If we expect highly non-rigid deformations then it is necessary to use NRDC or optical flow. However, rigid or close to rigid deformations are better approximated by using tracking. For example, in the tracking footage we would not expect optical flow methods to work very well, since it is noisy and has low quality. For rigid transformations like we use in the camera planning, the best technique we found was to use correspondences (sparse or dense) and fit a parametric transformation to these correspondences. In general, we have found that the lower the dimensionality of the transformation, the more robust it will be to error in the correspondences.

## 4.9 Conclusion

In conclusion we have described an automatic method for creating infinitely loopy clips from casual videos. Our method is capable of handling complex camera and foreground motions, and we have demonstrated this on a variety of unconstrained videos downloaded from the internet.

Our method can be easily generalized to stitch two input videos instead of one, if the two clips capture the same scene under a similar viewpoint. This could be used for many applications, like video editing (a generalization of [15] from interview footage to more general videos), view interpolation [9], or video summarization (e.g., summaries from multiple videos of the same event [2]). In addition, different components of our pipeline could be changed for specific applications. For e.g., the similarity metric could be modified to evaluate only motion similarity, instead of motion and appearance. This would allow stitching videos with similar dynamics but different appearances, such as different people doing the same dance or practicing the same sport.

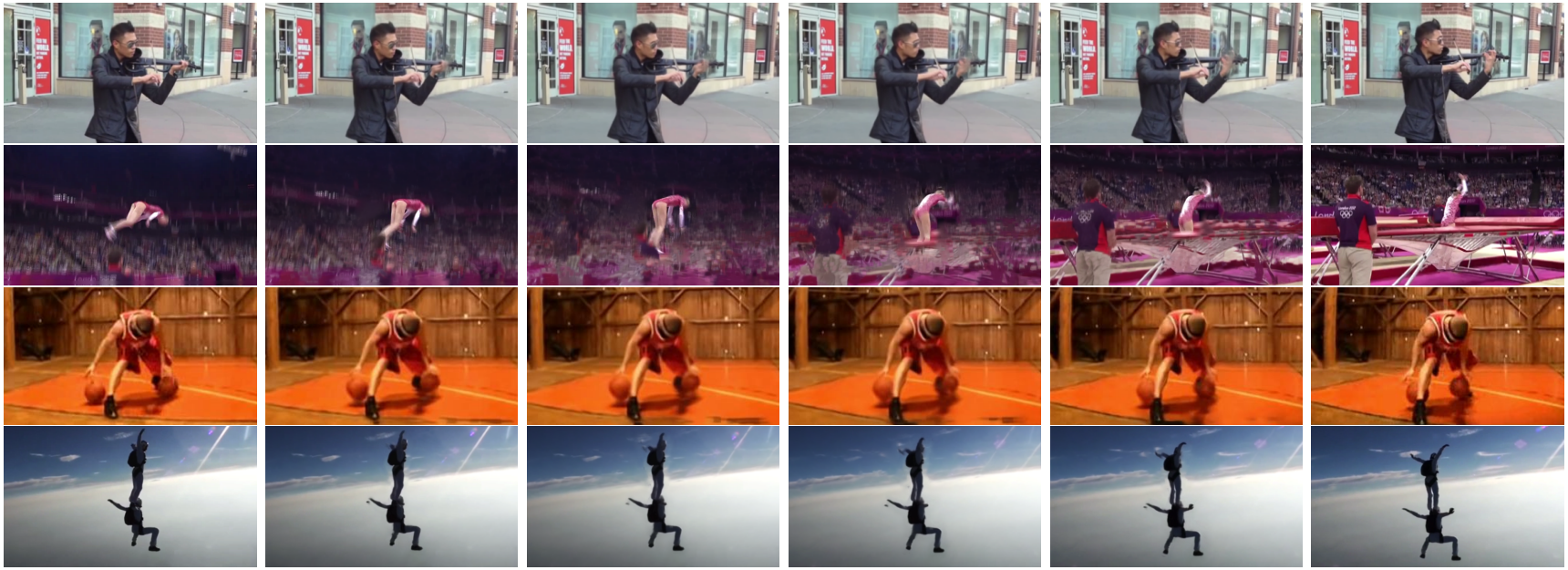


Figure 4.12: **Morphed results.** Results of our technique on a wide variety of examples. From top to bottom, **violin**, **trampoline**, **basketball**, and **skydiving**.


Figure 4.13: **Morphed results.** Results of our technique on a wide variety of examples. From top to bottom, **tango**, **dog**, **ski-slalom**, and **back-flip**.

Figure 4.14: **Comparison of end-to-end systems in the Tango sequence** Top: Video Textures. Middle: Our similarity metric with Traditional morphing. Bottom: Proposed method.


Figure 4.15: **Comparison of end-to-end systems in the Ski sequence.** Top: Video Textures-style morphing. Middle: Traditional morphing. Bottom: Proposed method.

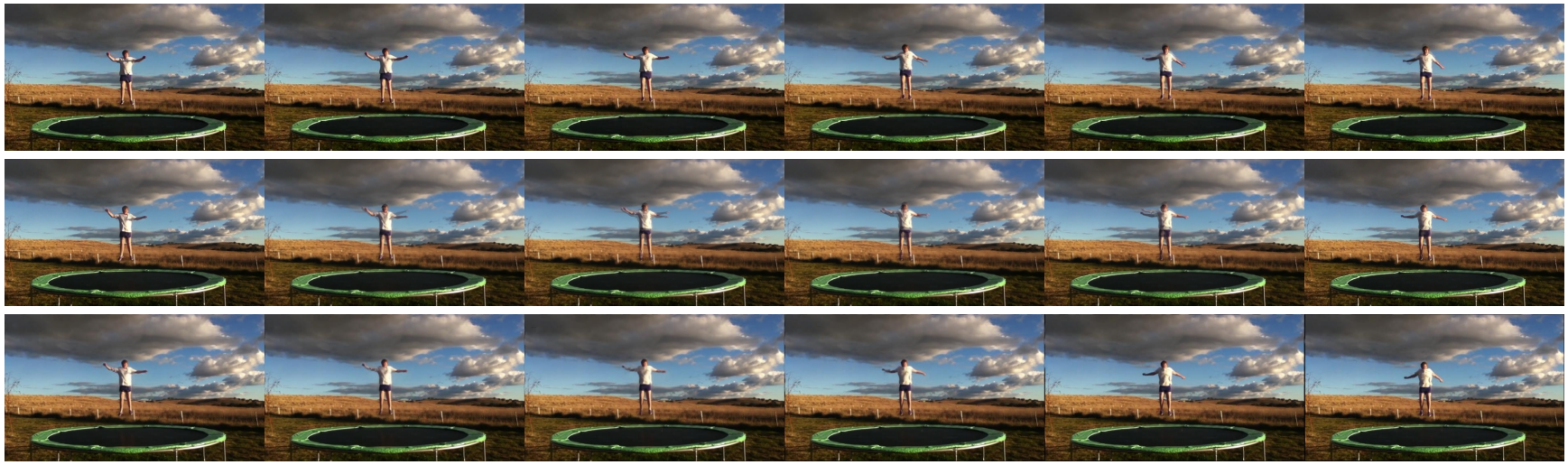


Figure 4.16: **Comparison of end-to-end systems in the Trampoline sequence.** Top: Video Textures-style morphing. Middle: Traditional morphing. Bottom: Proposed method.

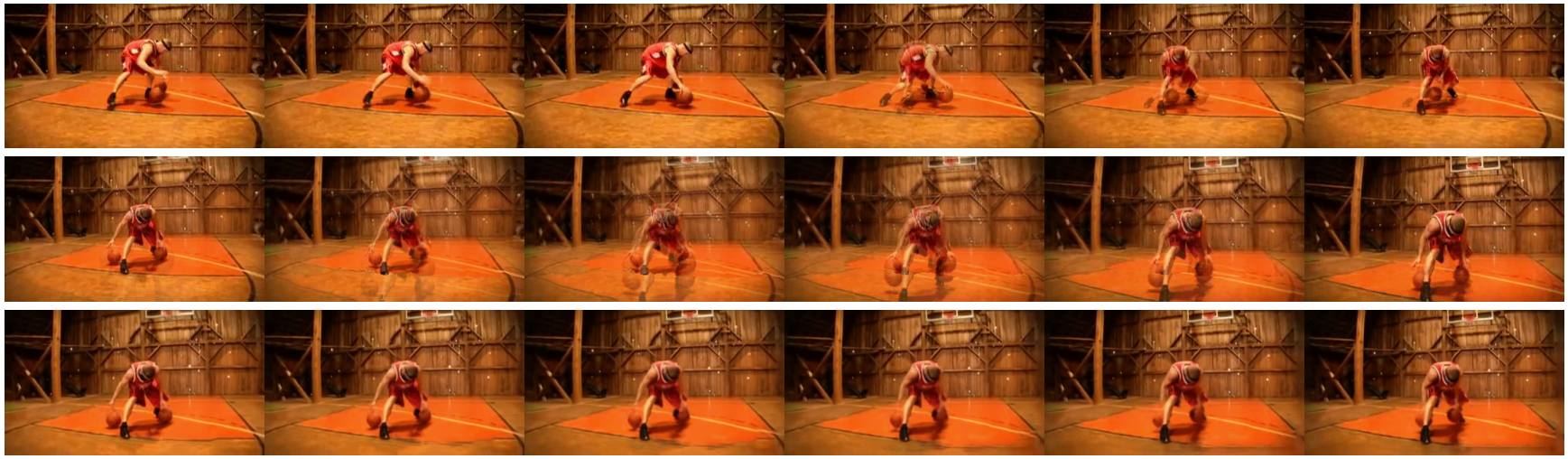


Figure 4.17: **Comparison of end-to-end systems in the Basketball sequence.** Top: Video Textures-style morphing. Middle: Traditional morphing. Bottom: Proposed method.

# CHAPTER 5

# CONCLUSIONS AND FUTURE WORK

In this thesis we have presented a technique to improve image correspondences by widening the basin of attraction of the objective function. We have demonstrated its usefulness in two different problems: object tracking and optical flow, and we have shown performance improvement in both problems. This seems like a promising tool for other methods that also find correspondences through gradient descent, like stereo vision, video registration, or structure from motion.

Our experiments have also cast some light into what may be interesting and useful future directions for the computer vision tools we used. Although there is great value in keeping basic problems simple, we consider that there is also a lot to be gained by shaping these basic problems according to the needs of high level systems. A few examples of these extensions are:

- **Incorporating uncertainty in the output of optical flow.** This would extend the use of optical flow to measuring similarity, the same way we used NRDC, but with higher accuracy. More generally, this would allow higher level systems to decide explicitly what to do in cases where there is little evidence for the match. Finally, this would allow the output of optical flow to be integrated into probabilistic systems.

- **Test optical flow methods more often on real videos.** Ground truth optical flow is very difficult to produce, and this has led to the use of small benchmarks for a long time, recorded in the lab under controlled conditions.

Recently, researchers have developed new datasets like *KITTI* [40] and *Sintel* that show more challenging scenarios. Still, when we look at results of optical flow methods in unconstrained video the quality is surprisingly low. It would be interesting if, in addition to the numerical results in datasets, optical flow methods were evaluated on real world videos, so that methods evolve accordingly to solve the real problem.

- **Object tracking would be very useful if it contained a mask of the object**. This would make it more usable, since most interesting objects are rarely square, nor do they deform rigidly. Also, this would improve the biggest problem of object tracking, which is drift: the template could be carefully updated based on this mask, and maybe even a 3D model could be constructed.

- **Problems should be more integrated.** This integration could come in two ways. First, most high-level computer vision problems require multiple inputs, like motion, segmentation, lighting or shape. These are often computed separately, with some notable exceptions [97, 58]. Solving for multiple problems at once could be helpful to get rid of ambiguities. Secondly, top-down integration. We have discussed that design decisions should be influenced by the needs of higher level systems. Low-level problems could be improved by including high-level knowledge, like semantics of the scene.

Along with these challenges, our experiments on applying motion estimation to graphics are a confirmation of the usefulness and potential of this computer vision problem.

# BIBLIOGRAPHY


[1] Agarwala, Aseem, Zheng, Ke Colin, Pal, Chris, Agrawala, Maneesh, Cohen, Michael, Curless, Brian, Salesin, David, and Szeliski, Richard. Panoramic video textures. In *ACM SIGGRAPH 2005 Papers* (New York, NY, USA, 2005), SIGGRAPH '05, ACM, pp. 821–827.

[2] Arev, Ido, Park, Hyun Soo, Sheikh, Yaser, Hodgins, Jessica, and Shamir, Ariel. Automatic editing of footage from multiple social cameras. *ACM Transactions on Graphics (Proc. SIGGRAPH) 33*, 4 (July 2014), 81:1–81:11.

[3] Avidan, Shai. Support vector tracking. In *PAMI* (2001).

[4] Babenko, B., Yang, Ming-Hsuan, and Belongie, S. Visual tracking with online multiple instance learning. *Proc. IEEE Conference on Computer Vision and Pattern Recognition 0* (2009), 983–990.

[5] Bai, Jiamin, Agarwala, Aseem, Agrawala, Maneesh, and Ramamoorthi, Ravi. Selectively de-animating video. *ACM Trans. Graph. 31*, 4 (July 2012), 66:1–66:10.

[6] Bai, Xue, Wang, Jue, Simons, David, and Sapiro, Guillermo. Video snapcut: Robust video object cutout using localized classifiers. *ACM Transactions on Graphics (Proc. SIGGRAPH) 28*, 3 (July 2009), 70:1–70:11.

[7] Baker, Simon, and Matthews, Iain. Lucas-Kanade 20 years on: A unifying framework. *International Journal of Computer Vision 56* (2004), 221–255.

[8] Baker, Simon, Scharstein, Daniel, Lewis, J. P., Roth, Stefan, Black, Michael J., and Szeliski, Richard. A database and evaluation methodology for optical flow. *Int. J. Comput. Vision 92*, 1 (Mar. 2011), 1–31.

[9] Ballan, Luca, Brostow, Gabriel J., Puwein, Jens, and Pollefeys, Marc. Unstructured video-based rendering: Interactive exploration of casually captured videos. *ACM Transactions on Graphics (Proc. SIGGRAPH) 29*, 4 (July 2010), 87:1–87:11.

[10] Bao, Linchao, Yang, Qingxiong, and Jin, Hailin. Fast edge-preserving patchmatch for large displacement optical flow. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2014), IEEE.

[11] Barnes, Connelly, Shechtman, Eli, Finkelstein, Adam, and Goldman, Dan B. Patchmatch: A randomized correspondence algorithm for structural image editing. *ACM Trans. Graph. 28*, 3 (July 2009), 24:1–24:11.

[12] Barnes, Connelly, Shechtman, Eli, Goldman, Dan B, and Finkelstein, Adam. The generalized patchmatch correspondence algorithm. In *Computer Vision–ECCV 2010*. Springer, 2010, pp. 29–43.

[13] Belongie, S., Malik, J., and Puzicha, J. Shape matching and object recognition using shape contexts. *IEEE Trans. Pattern Anal. Mach. Intell. 24*, 4 (Apr. 2002), 509–522.

[14] Berg, Alexander C, and Malik, Jitendra. Geometric blur for template matching. In *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on* (2001), vol. 1, IEEE, pp. I–607.

[15] Berthouzoz, Floraine, Li, Wilmot, and Agrawala, Maneesh. Tools for placing cuts and transitions in interview video. *ACM Trans. Graph. 31*, 4 (July 2012), 67:1–67:8.

[16] Birchfield, Stanley T., and Rangarajan, Sriram. Spatiograms versus histograms for region-based tracking. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition* (2005), pp. 1158–1163.

[17] Black, Michael J., and Anandan, P. The robust estimation of multiple motions: parametric and piecewise-smooth flow fields. *Computer Vision and Image Understanding 63* (January 1996), 75–104.

[18] Bleyer, Michael, Rhemann, Christoph, and Rother, Carsten. Patchmatch stereo-stereo matching with slanted support windows. In *BMVC* (2011), vol. 11, pp. 1–11.

[19] Brox, T., and Malik, J. Large displacement optical flow: descriptor matching in variational motion estimation. *IEEE Transactions on Pattern Analysis and Machine Intelligence 33*, 3 (Mar. 2011), 500–513.

[20] Bruhn, Andres, Weickert, Joachim, and Schnörr, Christoph. Lucas/Kanade meets Horn/Schunck: Combining local and global optic flow methods. *IJCV 61*, 3 (2005), 211–231.

[21] Burt, Peter J., and Adelson, Edward H. The Laplacian pyramid as a compact image code. *IEEE Transactions on Communications 31*, 4 (1983), 532–540.

[22] Butler, D. J., Wulff, J., Stanley, G. B., and Black, M. J. A naturalistic open source movie for optical flow evaluation. In *European Conf. on Computer Vision (ECCV)* (oct 2012), A. Fitzgibbon et al. (Eds.), Ed., Part IV, LNCS 7577, Springer-Verlag, pp. 611–625.

[23] Cannons, Kevin J., Gryn, Jacob M., and Wildes, Richard P. Visual tracking using a pixelwise spatiotemporal oriented energy representation. In *Proceedings of European Conference on Computer Vision: Part IV* (2010), pp. 511–524.

[24] Charbonnier, P., Blanc-Feraud, L., Aubert, G., and Barlaud, M. Two deterministic half-quadratic regularization algorithms for computed imaging. In *IEEE Int. Conf. Image Proc. (ICIP)* (1994), vol. 2, pp. 168–172.

[25] Chaudhry, Rizwan, Ravichandran, Avinash, Hager, Gregory, and Vidal, René. Histograms of oriented optical flow and binet-cauchy kernels on nonlinear dynamical systems for the recognition of human actions. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on* (2009), IEEE, pp. 1932–1939.

[26] Collins, Robert T. Mean-shift blob tracking through scale space. *Proc. IEEE Conference on Computer Vision and Pattern Recognition 2* (2003), 234.

[27] Collins, Robert T., and Liu, Yanxi. On-line selection of discriminative tracking features. *Proceedings of the International Conference on Computer Vision 1* (2003), 346.

[28] Comaniciu, Dorin, Ramesh, Visvanathan, and Meer, Peter. Real-time tracking of non-rigid objects using mean shift. *Proc. IEEE Conference on Computer Vision and Pattern Recognition 2* (2000), 2142.

[29] Cutler, Ross, and Turk, Matthew. View-based interpretation of real-time optical flow for gesture recognition. In *2013 10th IEEE International Conference and Workshops on Automatic Face and Gesture Recognition (FG)* (1998), IEEE Computer Society, pp. 416–416.

[30] Dalal, Navneet, and Triggs, Bill. Histograms of oriented gradients for human detection. *Proc. IEEE Conference on Computer Vision and Pattern Recognition 1* (2005), 886–893.

[31] Doretto, G., Chiuso, A., Wu, Y., and Soatto, S. Dynamic textures. *International Journal of Computer Vision 51*, 2 (2003), 91–109.

[32] Efros, Alexei A, and Leung, Thomas K. Texture synthesis by non-parametric sampling. In *Computer Vision, 1999. The Proceedings of the Seventh IEEE International Conference on* (1999), vol. 2, IEEE, pp. 1033–1038.

[33] Elgammal, Ahmed M., Harwood, David, and Davis, Larry S. Non-parametric model for background subtraction. In *Proceedings of the European Conference on Computer Vision-Part II* (2000), pp. 751–767.

[34] Fan, Zhimin, Yang, Ming, and Wu, Ying. Multiple collaborative kernel tracking. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition* (2005).

[35] Fan, Zhimin, Yang, Ming, Wu, Ying, Hua, Gang, and Yu, Ting. Efficient optimal kernel placement for reliable visual tracking. In *Proceedings of the 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition - Volume 1* (2006), pp. 658–665.

[36] Felsberg, Michael. Spatio-featural scale-space. In *Scale Space and Variational Methods in Computer Vision (SSVM)*, LNCS 5567. Springer-Verlag, 2009, pp. 808–819.

[37] Felsberg, Michael. Adaptive filtering using channel representations. In *Mathematical Methods for Signal and Image Analysis and Representation*. Springer-Verlag, 2012, pp. 31–48.

[38] Felsberg, Michael, Forssén, P-E, and Scharr, Hanno. Channel smoothing: Efficient robust smoothing of low-level signal features. *PAMI 28*, 2 (2006), 209–222.

[39] Freeman, Jeremy, and Simoncelli, Eero P. Metamers of the ventral stream. *Nature neuroscience 14*, 9 (2011), 1195–1201.

[40] Geiger, Andreas, Lenz, Philip, and Urtasun, Raquel. Are we ready for autonomous driving? the kitti vision benchmark suite. In *Conference on Computer Vision and Pattern Recognition (CVPR)* (2012).

[41] Goldman, Dan R. *A framework for video annotation, visualization, and interaction*. PhD thesis, University of Washington, 2007.

[42] Gomes, Jonas. *Warping and Morphing of Graphical Objects*. Morgan Kaufmann, 1999.

[43] Granlund, Gösta H. An associative perception-action structure using a localized space variant information representation. In *Proceedings of the Second International Workshop on Algebraic Frames for the Perception-Action Cycle* (London, UK, UK, 2000), AFPAC '00, Springer-Verlag, pp. 48–68.

[44] Grauman, Kristen, and Darrell, Trevor. The pyramid match kernel: Discriminative classification with sets of image features. In *Computer Vision, 2005. ICCV 2005. Tenth IEEE International Conference on* (2005), vol. 2, IEEE, pp. 1458–1465.

[45] HaCohen, Yoav, Shechtman, Eli, Goldman, Dan B, and Lischinski, Dani. Non-rigid dense correspondence with applications for image enhancement. *ACM Transactions on Graphics (Proceedings of ACM SIGGRAPH 2011) 30*, 4 (2011), 70:1–70:9.

[46] Hager, Gregory, , Hager, Gregory D., Dewan, Maneesh, and Stewart, Charles V. Multiple kernel tracking with SSD. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition* (2004), pp. 790–797.

[47] Haussecker, Horst W., and Fleet, David J. Computing optical flow with physical models of brightness variation. *IEEE Trans. Pattern Anal. Mach. Intell. 23*, 6 (June 2001), 661–673.

[48] Herman, Martin, and Kanade, Takeo. The 3d mosaic scene understanding system: Incremental reconstruction of 3d scenes from complex images.

[49] Hertzmann, Aaron, Jacobs, Charles E, Oliver, Nuria, Curless, Brian, and Salesin, David H. Image analogies. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques* (2001), ACM, pp. 327–340.

[50] Horn, Berthold K.P., and Schunck, Brian G. Determining optical flow. Tech. rep., Massachusetts Institute of Technology, Cambridge, MA, USA, 1980.

[51] Irani, Michal, and Anandan, P. Parallax geometry of pairs of points for 3d scene analysis. In *Computer VisionECCV'96*. Springer, 1996, pp. 17–30.

[52] Jepson, Allan D., Fleet, David J., and El-Maraghi, Thomas F. Robust online appearance models for visual tracking. *Proc. IEEE Conference on Computer Vision and Pattern Recognition 1* (2001), 415.

[53] Jonsson, Erik, and Felsberg, Michael. Accurate interpolation in appearance-based pose estimation. In *Image Analysis*, vol. 4522 of *Lecture Notes in Computer Science*. Springer-Verlag, 2007, pp. 1–10.

[54] Jonsson, Erik, and Felsberg, Michael. Efficient computation of channel-coded feature maps through piecewise polynomials. *Image and Vision Computing 27*, 11 (2009).

[55] Joshi, Neel, Mehta, Sisil, Drucker, Steven, Stollnitz, Eric, Hoppe, Hugues, Uyttendaele, Matt, and Cohen, Michael. Cliplets: Juxtaposing still and dynamic imagery. In *Proceedings of the 25th Annual ACM Symposium on User Interface Software and Technology* (2012), UIST '12, pp. 251–260.

[56] Kemelmacher-Shlizerman, Ira, Shechtman, Eli, Garg, Rahul, and Seitz, Steven M. Exploring photobios. *ACM Trans. Graph. 30*, 4 (July 2011), 61:1–61:10.

[57] Koenderink, Jan J, Van Doorn, Andrea J, et al. Affine structure from motion. *JOSA A 8*, 2 (1991), 377–385.

[58] Kong, Naejin, Gehler, Peter V., and Black, Michael J. Intrinsic video. In *European Conference on Computer Vision (ECCV)* (Sept. 2014).

[59] Kopf, Johannes, Fu, Chi-Wing, Cohen-Or, Daniel, Deussen, Oliver, Lischinski, Dani, and Wong, Tien-Tsin. Solid texture synthesis from 2d exemplars. *ACM Transactions on Graphics (TOG) 26*, 3 (2007), 2.

[60] Kumar, Neeraj, Zhang, Li, and Nayar, Shree. What is a good nearest neighbors algorithm for finding similar patches in images? In *Computer Vision–ECCV 2008*. Springer, 2008, pp. 364–378.

[61] Kwatra, Vivek, Essa, Irfan, Bobick, Aaron, and Kwatra, Nipun. Texture optimization for example-based synthesis. In *ACM Transactions on Graphics (TOG)* (2005), vol. 24, ACM, pp. 795–802.

[62] Kwatra, Vivek, Schödl, Arno, Essa, Irfan, Turk, Greg, and Bobick, Aaron. Graphcut textures: Image and video synthesis using graph cuts. In *ACM SIGGRAPH 2003 Papers* (New York, NY, USA, 2003), SIGGRAPH '03, ACM, pp. 277–286.

[63] Learned-Miller, Erik G. Data driven image models through continuous joint alignment. *IEEE Transactions on Pattern Analysis and Machine Intelligence 28* (2006), 2006.

[64] Leibe, Bastian, Cornelis, Nico, Cornelis, Kurt, and Van Gool, Luc. Dynamic 3d scene analysis from a moving vehicle. In *Computer Vision and Pattern Recognition, 2007. CVPR'07. IEEE Conference on* (2007), IEEE, pp. 1–8.

[65] Leung, Alex Po, and Gong, Shaogang. Mean shift tracking with random sampling. In *Proc. BMVC 2005* (2006), pp. 729–738.

[66] Levieux, Philippe, Tompkin, James, and Kautz, Jan. Interactive viewpoint video textures. In *Visual Media Production (CVMP), 2012 Conference for* (December 2012).

[67] Liao, J., Lima, R. S., Nehab, D., Hoppe, H., and Sander, P. V. Semi-automated video morphing. *Computer Graphics Forum (Proceedings of Eurographics Symposium on Rendering 2014) 33*, 4 (2014), 51–60.

[68] Liao, Zicheng, Joshi, Neel, and Hoppe, Hugues. Automated video looping with progressive dynamism. *ACM Transactions on Graphics (Proc. SIGGRAPH) 32*, 4 (July 2013), 77:1–77:10.

[69] Lienhart, Rainer, and Maydt, Jochen. An extended set of haar-like features for rapid object detection. In *Image Processing. 2002. Proceedings. 2002 International Conference on* (2002), vol. 1, IEEE, pp. I–900.

[70] Liu, Ce, and Sun, Deqing. A bayesian approach to adaptive video super resolution. In *Proceedings of the 2011 IEEE Conference on Computer Vision and Pattern Recognition* (Washington, DC, USA, 2011), CVPR '11, IEEE Computer Society, pp. 209–216.

[71] Liu, Ce, Yuen, Jenny, Torralba, Antonio, Sivic, Josef, and Freeman, William T. Sift flow: Dense correspondence across different scenes. In *Proceedings of the 10th European Conference on Computer Vision: Part III* (Berlin, Heidelberg, 2008), ECCV '08, Springer-Verlag, pp. 28–42.

[72] Lowe, David G. Object recognition from local scale-invariant features. In *Computer vision, 1999. The proceedings of the seventh IEEE international conference on* (1999), vol. 2, Ieee, pp. 1150–1157.

[73] Lowe, David G. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision 60* (November 2004), 91–110.

[74] Lucas, Bruce D, Kanade, Takeo, et al. An iterative image registration technique with an application to stereo vision. In *IJCAI* (1981), vol. 81, pp. 674–679.

[75] Marcel, Hieu Nguyen, Worring, Marcel, and Boomgaard, Rein Van Den. Occlusion robust adaptive template tracking. In *Proceedings of the International Conference on Computer Vision* (2001), pp. 678–683.

[76] Matthews, Iain, Ishikawa, Takahiro, and Baker, Simon. The template update problem. In *BMVC* (2003).

[77] Mikolajczyk, Krystian, Leibe, Bastian, and Schiele, Bernt. Local features for object class recognition. In *Computer Vision, 2005. ICCV 2005. Tenth IEEE International Conference on* (2005), vol. 2, IEEE, pp. 1792–1799.

[78] Mobahi, Hossein, Zitnick, C. Lawrence, and Ma, Yi. Seeing through the blur. In *CVPR* (2012).

[79] Mori, Greg, Belongie, Serge, and Malik, Jitendra. Efficient shape matching using shape contexts. *IEEE Transactions on Pattern Analysis and Machine Intelligence 27*, 11 (November 2005), 1832–1837.

[80] Papazoglou, Anestis, and Ferrari, Vittorio. Fast object segmentation in unconstrained video. *ICCV* (2013).

[81] Rother, Carsten, Kolmogorov, Vladimir, and Blake, Andrew. GrabCut: Interactive foreground extraction using iterated graph cuts. In *ACM Transactions on Graphics (Proc. SIGGRAPH)* (2004), pp. 309–314.

[82] Rüegg, Jan, Wang, Oliver, Smolic, Aljoscha, and Gross, Markus. Ducttake: Spatiotemporal video compositing. In *Computer Graphics Forum* (2013), vol. 32, Wiley Online Library, pp. 51–61.

[83] Santner, Jakob, Leistner, Christian, Saffari, Amir, Pock, Thomas, and Bischof, Horst. PROST: Parallel robust online simple tracking. *Proc. IEEE Conference on Computer Vision and Pattern Recognition 0* (2010), 723–730.

[84] Schödl, Arno, Szeliski, Richard, Salesin, David H, and Essa, Irfan. Video textures. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques* (2000), ACM Press/Addison-Wesley Publishing Co., pp. 489–498.

[85] Sevilla-Lara, Laura, and Learned-Miller, Erik. Distribution fields. Tech. rep., University of Massachusetts Amherst, 2011.

[86] Sevilla-Lara, Laura, and Learned-Miller, Erik. Distribution fields for tracking. *CVPR* (2012).

[87] Sevilla-Lara, Laura, Sun, Deqing, Learned-Miller, Erik G., and Black, Michael J. Optical flow estimation with channel constancy. In *European Conference on Computer Vision (ECCV)* (Sept. 2014).

[88] Shechtman, Eli, Rav-Acha, Alex, Irani, Michal, and Seitz, Steve. Regenerative morphing. In *IEEE Conference on Computer VIsion and Pattern Recognition (CVPR)* (San-Francisco, CA, June 2010).

[89] Shechtman, Eli, Rav-Acha, Alex, Irani, Michal, and Seitz, Steven M. Regenerative morphing. In *CVPR* (2010), IEEE, pp. 615–622.

[90] Shi, Jianbo, and Malik, Jitendra. Motion segmentation and tracking using normalized cuts. In *Computer Vision, 1998. Sixth International Conference on* (1998), IEEE, pp. 1154–1160.

[91] Simakov, Denis, Caspi, Yaron, Shechtman, Eli, and Irani, Michal. Summarizing visual data using bidirectional similarity. In *CVPR* (2008), IEEE Computer Society.

[92] Stauffer, Chris, and Grimson, W. Eric. L. Learning patterns of activity using real-time tracking. *IEEE Transactions on Pattern Analysis and Machine Intelligence 22*, 8 (2000), 747–757.

[93] Steinbrucker, Frank, Pock, Thomas, and Cremers, Daniel. Large displacement optical flow computation withoutwarping. In *Computer Vision, 2009 IEEE 12th International Conference on* (2009), IEEE, pp. 1609–1614.

[94] Steinbruecker, F., Pock, T., and Cremers, D. Advanced data terms for variational optic flow estimation. In *Proceedings Vision, Modeling and Visualization* (Braunschweig, Germany, 2009).

[95] Sudderth, Erik B. *Graphical models for visual object recognition and tracking.* PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, USA, 2006. AAI0809973.

[96] Sun, D., Roth, S., and Black, M. J. Secrets of optical flow estimation and their principles. In *IEEE Conf. on Computer Vision and Pattern Recognition, CVPR* (jun 2010), pp. 2432–2439.

[97] Sun, Deqing. *From Pixels to Layers: Joint Motion Estimation and Segmentation.* PhD thesis, Brown University, Department of Computer Science, July 2012.

[98] Sun, Deqing, Roth, Stefan, Lewis, J.P., and Black, Michael J. Learning optical flow. In *European Conf. on Computer Vision, ECCV* (Oct. 2008), D. Forsyth, P. Torr, and A. Zisserman, Eds., vol. 5304 of *LNCS*, Springer-Verlag, p. 8397.

[99] Szeliski, Richard. Image alignment and stitching: A tutorial. *Foundations and Trends® in Computer Graphics and Vision 2*, 1 (2006), 1–104.

[100] Szeliski, Richard. Image alignment and stitching: a tutorial. *Found. Trends. Comput. Graph. Vis. 2* (January 2006), 1–104.

[101] Tomasi, C., and Manduchi, R. Bilateral filtering for gray and color images. In *Proceedings of the International Conference on Computer Vision* (1998), pp. 839–.

[102] Tompkin, James, Pece, Fabrizio, Subr, Kartic, and Kautz, Jan. Towards moment images: Automatic cinemagraphs. In *Visual Media Production (CVMP)* (November 2011), pp. 87–93.

[103] Weber, Joseph, Malik, Jitendra, Devadas, S., and Michel, P. Robust computation of optical flow in a multi-scale differential framework. *International Journal of Computer Vision 14* (1994), 12–20.

[104] Wei, Li-Yi, and Levoy, Marc. Fast texture synthesis using tree-structured vector quantization. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques* (2000), ACM Press/Addison-Wesley Publishing Co., pp. 479–488.

[105] Weinzaepfel, Philippe, Revaud, Jerome, Harchaoui, Zaid, and Schmid, Cordelia. Deepflow: Large displacement optical flow with deep matching. In *ICCV* (2013), pp. 1385–1392.

[106] Werlberger, Manuel. *Convex Approaches for High Performance Video Processing*. PhD thesis, Institute for Computer Graphics and Vision, Graz University of Technology, Graz, Austria, June 2012.

[107] Wexler, Yonatan, Shechtman, Eli, and Irani, Michal. Space-time video completion. In *Computer Vision and Pattern Recognition, 2004. CVPR 2004. Proceedings of the 2004 IEEE Computer Society Conference on* (2004), vol. 1, IEEE, pp. I–120.

[108] Yamato, Junji, Ohya, Jun, and Ishii, Kenichiro. Recognizing human action in time-sequential images using hidden markov model. In *Computer Vision and Pattern Recognition, 1992. Proceedings CVPR'92., 1992 IEEE Computer Society Conference on* (1992), IEEE, pp. 379–385.

[109] Yin, Zhaozheng, Porikli, Fatih, and Collins, Robert T. Likelihood map fusion for visual object tracking. In *Proceedings of the 2008 IEEE Workshop on Applications of Computer Vision* (2008), pp. 1–7.

[110] Zheng, Ke Colin, Colburn, Alex, Agarwala, Aseem, Agrawala, Maneesh, Curless, Brian, Salesin, David, and Cohen, Michael. Parallax photography: Creating 3D cinematic effects from stills. In *Graphics Interface 2009* (May 2009).

[111] Zitnick, C Lawrence, Kang, Sing Bing, Uyttendaele, Matthew, Winder, Simon, and Szeliski, Richard. High-quality video view interpolation using a layered representation. In *ACM Transactions on Graphics (TOG)* (2004), vol. 23, ACM, pp. 600–608.