LAP-BASED MOTION-COMPENSATED FRAME INTERPOLATION

BY

TEJAS JAYASHANKAR

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Bachelor of Science in Electrical and Computer Engineering
in the Undergraduate College of the
University of Illinois at Urbana-Champaign, 2019

Urbana, Illinois

Adviser:

Professor Pierre Moulin

ABSTRACT

High-quality video frame interpolation often necessitates accurate motion estimates between consecutive frames. Standard video encoding schemes often estimate the motion between frames using variants of block matching algorithms. For the sole purposes of video frame interpolation, more accurate estimates can be obtained using modern optical flow methods.

In this thesis, we use the recently proposed Local All-Pass (LAP) algorithm to compute the optical flow between two consecutive frames. The resulting flow field is used to perform interpolation using cubic splines. We compare the interpolation results against a well-known optical flow estimation algorithm as well as against a recent convolutional neural network scheme for video frame interpolation. Qualitative and quantitative results show that the LAP algorithm performs fast, high-quality video frame interpolation, and perceptually outperforms the neural network and the Lucas-Kanade method on a variety of test sequences. We also perform a case study to compare LAP interpolated frames against those obtained using two leading methods on the Middlebury optical flow benchmark. Finally, we perform a user study to gauge the correlation between the quantitative and qualitative results.

Keywords: Motion Compensation, Frame Interpolation, Optical Flow, Splines, Convolutional Neural Network, Adaptive Separable Convolution, Lucas-Kanade algorithm, Local All-Pass filters

ACKNOWLEDGMENTS

I would like to express my deepest appreciation for my advisor and mentor, Professor Pierre Moulin, who has provided me with guidance, support and invaluable research experience during the course of our research project. Ever since our first conversation in my freshman year, I knew that I would have the ability to deepen my knowledge and become an independent researcher by working with Professor Moulin. During the course of our research project, I have expanded my understanding of signal processing, statistical inference and machine learning. These skills have proven not only valuable for my work here at UIUC, but they will also play an important role during my future research internships and for my Ph.D.

I am also very fortunate to author a paper with Professor Moulin, Professor Thierry Blu and Dr. Christopher Gilliam, that has been submitted to the IEEE Conference for Image Processing (ICIP 2019). I would like to also express my deepest appreciation for Professor Thierry Blu at the Chinese University of Hong Kong and Dr. Christopher Gilliam at RMIT University, Melbourne for their valuable inputs while writing the paper and for sharing code that was used during the initial development of the LAP algorithm. Professor Moulin was very supportive during the conference paper submission phase and gave me constructive feedback that I will forever remember while writing a research paper.

I would also like to thank all my other professors and staff at UIUC who have made the past four years a fun, energetic and unforgettable experience. I would like to especially thank Professor Radhakrishnan for his invaluable support and mentorship over the past few years. I have not only learned from my professors, but also from my friends. I am glad to have made so many caring and motivated friends during my time here and it is the daily

interactions with them that make college so much more fun and lively.

I would also like to thank my family for all the love, care and support that they have given me. I would like to especially thank my parents and my sister, Bhavya, for moving half way across the world to help me achieve my dream of studying in the US. I would like to especially thank my dad, the most hard-working person I know, and it is his hard work and motivating spirit that allow me to study at a top-notch university. Without my mom's love, hard work, support and tasty food, my experience at UIUC would be bland. My sister constantly reminds me to have fun while working and I am truly grateful for her love and companionship. Finally, I would like to thank my grandparents who have encouraged me from a young age to achieve my goals and aspirations.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

Abbreviation	**Expansion**
4SS	Four-Step Search
ARPS	Adaptive Rood Pattern Search
BA	Black and Anandan
CNN	Convolutional Neural Network
FRUP	Frame-Rate Up-Conversion
FS	Full Search
HS	Horn-Schunck
LAP	Local All-Pass
LK	Lucas-Kanade
MCFI	Motion-Compensated Frame Interpolation
MC-FRUP	Motion-Compensated Frame-Rate Up-Conversion
MOMS	Maximal-Order Interpolation of Minimal Support
NTSS	New Three-Step Search
OMOMS	Optimal Maximal-Order Interpolation of Minimal Support
PF	Poly-Filter
TSS	Three-Step Search

CHAPTER 1

INTRODUCTION

1.1 Background

1.1.1 Motion-Compensated Frame Interpolation

Frame-Rate Up-Conversion (FRUC) is the process by which new intermediate frames are inserted into a video sequence with the goal of improving the temporal resolution (frame-rate) of the video sequence. These intermediate frames are generated by only using the frames in the original video. These intermediate frames could be a copy of one of the original frames in the sequence or they could be newly generated frames.

FRUC plays a pivotal role in video processing and has many applications to various fields. In a video coding and compression setup, with a limited bit transmission rate, the system is constrained by memory and processing power. FRUC can not only be used to interpolate between the input frames, but an advanced algorithm can also accurately estimate the displacement fields, which can then be transmitted. In [1], Krishnamurthy *et al.* encode the displacement field of a subsampled video using a block-matching approach under bit constraints. The decoder performs motion-compensated FRUC (MC-FRUC) to interpolate new frames using bidirectional predictions to tackle occlusions. Literature that covers FRUC for compressed video can also be found in [2, 3]. In this thesis we specifically study motion-compensated frame interpolation (MCFI), wherein the frame-rate is increased through the interpolation of new frames.

Frame interpolation might still be desired even if the system is equipped with high transmission rates and large memory bandwidth. MCFI has been

used for interpolation in echocardiographical video sequences [4]. Instead of interpolating a single frame, multiple frames are interpolated between a pair of images to augment the observable ventricular functions and better visualize cardiac dynamics. Image registration techniques are used for motion estimation and dimensionality reduction is employed to optimize for the number of intermediate frames to produce. It should also be noted that many MCFI algorithms are specifically designed for the underlying system that it is deployed on [5].

1.2 Thesis Objectives

In this thesis we use the recently developed Local All-Pass (LAP) algorithm for optical flow estimation to perform motion estimation. These motion estimates are then used to create an interpolated frame using cubic splines. We compare the interpolation results against the Lucas-Kanade method, a standard optical flow algorithm and a recently proposed convolutional neural network (CNN) approach for video-frame interpolation based on separable filter estimation.

The interpolated frames are evaluated quantitatively using the Mean-Squared Error (MSE) [6] and qualitatively by observing the interpolated frames. A detailed analysis of the interpolation quality is performed and finally a user study is conducted to further understand the correlation between MSE and subjective visual quality.

1.3 Organization of Thesis

This thesis is organized into six different chapters as follows:

1. Chapter 1 introduces the MCFI problem and details the organization of the thesis.

2. Chapter 2 is concerned with motion estimation. Section 2.1 covers state-of-the-art block matching algorithms for motion estimation. Section 2.2 introduces five standard optical flow algorithms for motion estimation.

3. Chapter 3 explains the MCFI problem statement and covers various interpolation algorithms used for interpolating between signals. Section 3.3 briefly explains a new convolutional neural network for frame interpolation.

4. Chapter 4 explains the Local All-Pass (LAP) algorithm for motion estimation. The main principles and algorithm details are explained.

5. Chapter 5 reports interpolation results on three different datasets. It also includes results from a user study performed to understand the correlation between MSE and visual quality.

6. Chapter 6 concludes the thesis and contains final remarks on the experiments and results.

CHAPTER 2

MOTION ESTIMATION

Motion estimation has been a hot research topic since the 1970s. Many of the original and currently used video coding standards use motion estimation to encode and decode information [7, 8]. Motion estimation between images is often computed in two different ways:

1. Block Matching: The images are broken up into blocks (also called macro-blocks), which is essentially a cluster of pixels. Typical block sizes are 4x4, 8x8 and 16x16. The best matches in relation to some error metric, between corresponding blocks in both images, are used to estimate the motion vectors.

2. Optical Flow: The motion between a pair of images is estimated for each pixel. Optical flow motion estimates tend to be much more accurate than block matching motion vectors. However, these methods also tend to be much more complex and computationally expensive.

In the next few sections we will review literature that discusses block-matching and optical flow methods for motion estimation.

2.1 Block Matching

Block matching algorithms implicitly assume that objects and patterns vary in position between images but remain structurally similar. The idea is to divide the images into blocks of size $n \times n$ where n is typically 4, 8 or 16. The illustration in 2.1 is taken from [9] and it shows a typical block size. For each block $B_1 \in \mathcal{B}$ in I_1, where $\mathcal{B}$ is the space of all blocks, the corresponding block B_2 in image I_2 can be found by minimizing some error criterion. Typical error criteria used are the Mean Squared Error (MSE) and

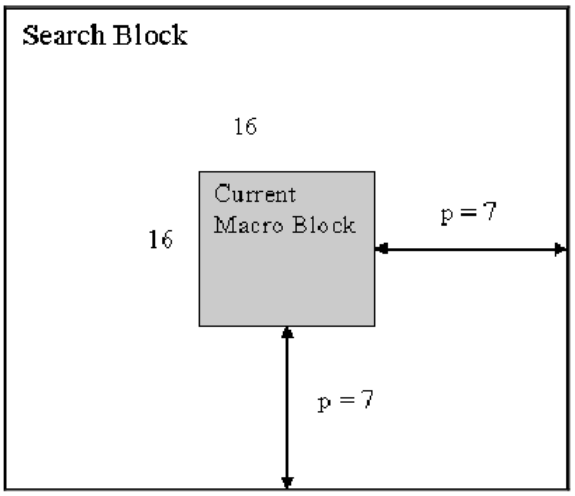


Figure 2.1: Block matching a block of width 16 pixels within a search radius of 7 pixels.

the Mean Absolute Error (MAE). The formula for these criteria between two $M \times N$ matrices is given below:

$$\mathbf{MSE}(x, y) = \frac{1}{MN} \sum_{i=1}^{M} \sum_{j=1}^{N} (x(i, j) - y(i, j))^2, \tag{2.1}$$

$$\mathbf{MAE}(x, y) = \frac{1}{MN} \sum_{i=1}^{M} \sum_{j=1}^{N} |x(i, j) - y(i, j)|. \tag{2.2}$$

Most algorithms also restrict the search radius to be within p pixels from the center of the current block under consideration as is shown in Figure 2.1. This is done for two reasons:

1. For faster computation. Searching over the whole image can be computationally intensive. Moreover, it lengthens the execution time.

2. In most situations, especially MCFI, assumptions on the velocity of the object can be made to ensure that the highest correlation between blocks only occurs within a fixed radius with high probability. This is because most natural occurring scenes have generally smoothly varying motion.

In the next few subsections we will briefly review commonly used block-matching algorithms for motion estimation. A more in-depth treatment of these algorithms can be found in [9, 10].

2.1.1 Full Search

In Full Search (FS), every block between I_1 and I_2 is compared within a search window. For every block in I_1, the block in I_2 that results in the lowest MSE is chosen. Let the images be broken up into blocks of size $K \times K$ and let the search window radius be p pixels. Then, the number of blocks, n, compared for each reference block is:

$$n = 4p^2. \tag{2.3}$$

Thus, the total number of operations performed is:

$$\text{ops} = n \times K^2 = \mathcal{O}((pK)^2). \tag{2.4}$$

FS gives the best match amongst all block matching algorithms for the same block width and search window radius. However, this comes at the cost of increased computational complexity and longer execution time.

2.1.2 Three-Step Search

The Three-Step Search (TSS) block matching algorithm [9] is one of the first block matching algorithms introduced and it forms the basis for many more complex block matching algorithms. Instead of searching all blocks within a window, TSS employs a much more strategic algorithm to find the best match for a reference frame while also reducing the time complexity. TSS finds a matching block in three steps.

1. **Step 1:** Match the central block and all blocks at a distance of 4 pixels away from the center of the reference block, x. Choose the central pixel of the matched block with lowest MSE. Call this position x_1.

2. **Step 2:** Reduce the search radius from 4 pixels to 2 pixels. Now match all blocks at a distance of 2 pixels from x_1 to the reference block. Choose the central pixel of the matched block with the lowest MSE. Call this position x_2.

3. **Step 3:** Reduce the search radius from 2 pixels to 1 pixel. Now match all blocks at a distance of 1 pixel from x_2 to the reference block. Choose

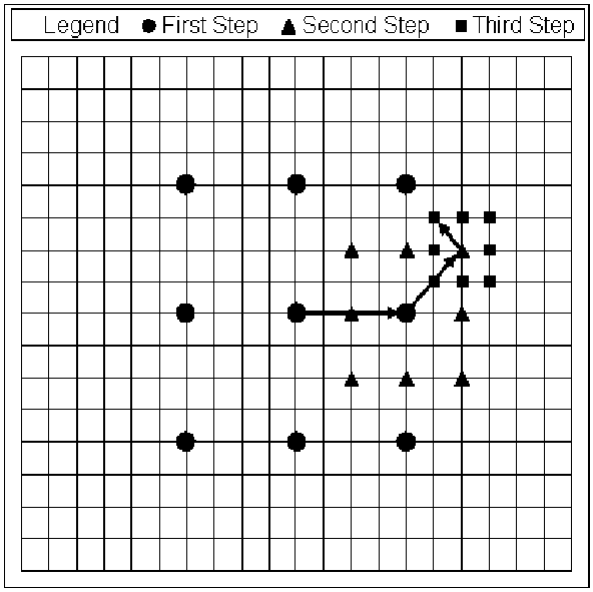


Figure 2.2: Illustration of Three-Step Search.

the central pixel of the matched block with the lowest MSE. Call this position x_3. The final motion vector is $x_3 - x$.

An illustration of the TSS search pattern taken from [9] is shown in Figure 2.2. Nine block comparisons are performed in the first step and eight block comparisons are performed in steps 2 and 3. Thus, the total number of block comparisons per reference frame is:

$$n = 9 + 8 + 8 = 25. \tag{2.5}$$

The total number of operations performed per reference frame is:

$$\text{ops} = n \times K^2 = \mathcal{O}(K^2). \tag{2.6}$$

Notice that for $p = 7$, which is a typical search radius, this amounts to more than a 50 factor decrease in operations per block in comparison to FS.

2.1.3 New Three-Step Search

The underlying assumption of New Three-Step Search (NTSS) is that the block motion field of a real world sequence is smooth and varies slowly. Li *et al.* [11] developed NTSS in 1994 as an extension of TSS that optimizes for

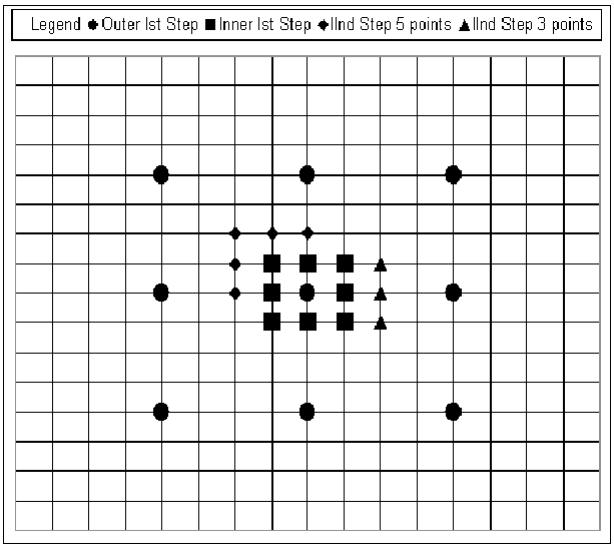


Figure 2.3: Illustration of New Three Step Search. If minimum occurs at a corner pixel five additional points are added (squares) else three additional points are added (triangles).

the center-bias of motion vectors. They incorporate a halfway-stop technique to reduce the time complexity of TSS in certain scenarios. The procedure is given below:

1. **Step 1:** In addition to matching the central block and all blocks at a distance of 4 pixels from the center, match 8 additional blocks at a distance of 1 pixel from the center of the reference frame. This is done to enforce the center-biased motion vector assumption.

2. **Step 2:** If the best match occurs at the center of the search window, the motion vector is $(0, 0)$. End the procedure. Otherwise, if the best match occurs at one of the inner 8 points, the new search pattern depends on the position of this point.

   (a) If a match occurs at a corner point, five additional points are checked as shown in Figure 2.3.

   (b) If a match occurs at one of the central points of the edges, three additional points are checked.

   If the best match occurs at one of the outer 9 points, follow Step 3.

3. **Step 3:** In this case the best match occurs at one of the outermost

points at a distance of 4 pixels from the center of the reference frame. Carry out TSS until convergence.

An illustration of the NTSS search pattern taken from [9] is shown in Figure 2.3. In NTSS, 17 comparisons are done in step 1. Step 2 can can carry out 0, 3 or 5 additional comparisons. Step 3 carries out 16 comparisons. Thus,

$$n \in \{17, 20, 22, 33\}. \tag{2.7}$$

According to [11], the chances of 33 comparisons occuring is small. Thus, NTSS has a slight improvement in performance over TSS. The NTSS algorithm was widely used in MPEG and H261.1 coding standards [9].

2.1.4 Four-Step Search

The Four-Step Search (4SS) was introduced by Po and Ma [12], and it is a further extension of NTSS. It is also based on the center-biased motion vector distribution assumption. Instead of three steps as in NTSS, four steps are used to incorporate a finer search in the last step combined with additional halfway-stop criteria. The procedure is given below:

1. **Step 1:** Find the best match from all blocks centered at a distance of 2 pixels from the center of the reference block. If the best match occurs at the center of the search window go to Step 4.

2. **Step 2:** If the best match occurs at one of the outer 8 points, the new search pattern depends on the position of this point.

   (a) If a match occurs at a corner point, five additional points are checked as shown in Figure 2.4.

   (b) If a match occurs at one of the central points of the edges, three additional points are checked.

   If the best match is found at the center of the newly created search window go to Step 4, else follow on to Step 3.

3. **Step 3:** Again using a search radius of 2 pixels, follow Step 2 and then move onto Step 4.

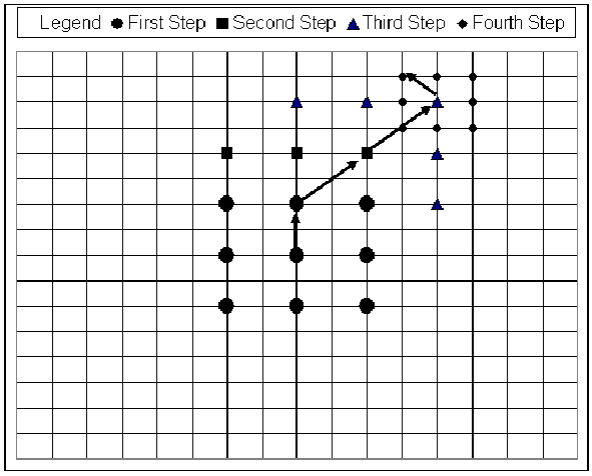


Figure 2.4: Illustration of Four Step Search.

4. **Step 4:** Now use a search radius of 1 pixel to refine the motion vector estimate.

The main difference between 4SS and NTSS is the addition of Step 4. There is also an additional halfway-stop point in Step 2. An illustration of the 4SS search pattern taken from [9] is shown in Figure 2.4. In the first step there are 9 comparisons made. In the second step there could be 0, 3 or 5 comparisons. In the third step an additional 3 or 5 comparisons are made. In the final step 8 comparisons are made. Thus, the total number of comparisons is

$$n \in \{17, 20, 22, 23, 25, 27\}. \tag{2.8}$$

Thus, in the worst case, 4SS performs 5 fewer comparisons than NTSS. In [12] it is shown that the performance of 4SS is similar to NTSS in terms of MSE. A further extension of 4SS is Diamond Search (DS) [13], where the search window is in the shape of a diamond instead of a square. Additionally, the number of steps is not fixed and the procedure can carry on in the same manner after Step 4. It performs similarly to 4SS and performance can be improved by increasing the number of steps.

2.1.5 Adaptive Rood Pattern Search

The underlying assumption of Adaptive Rood Pattern Search (ARPS) [14] is based on the observation that the motion estimates become more precise

as one moves towards the global minimum of the error criterion. Thus, the authors make the assumption that the error surface is uni-modal. They also incorporate some correlation between motion vectors. Specifically, they assume that the horizontally adjacent blocks have similar motion vectors. For a certain block under consideration, the blocks in the direction of the motion vector of the left adjacent block are first searched before any further refinement. The procedure for ARPS is below:

1. **Step 1:** Compute the error between the current block and the corresponding block in the reference frame. If the error is less than some threshold T ($T = 150$ in [14]), then set the motion vector estimate to $(0, 0)$ and end the procedure. Else, if the current block is the leftmost block in the search window set $\Gamma = 2$. Otherwise, $\Gamma = \max\{|MV(x)|, |MV(y)|\}$ where Γ is a measure of the maximum displacement of the left adjacent block and MV is the motion vector of the left adjacent block.

2. **Step 2:** Let the center of the current block be (x, y). Match the blocks at the positions $(x \pm \Gamma, y \pm \Gamma)$ and $(x + MV(x), y + MV(y))$. Find the point with the minimum error.

3. **Step 3:** Setup the window in the shape of the unit-size rood pattern and find the best match. Keep on iterating this procedure until the best match is found at the center of the search window.

2.2 Optical Flow

Optical flow methods relate changes between images by determining flow of intensities between images. Block matching algorithms are often counted as subsets of optical flow algorithms.Traditional optical flow algorithms often assume local smoothness and constant displacement flow to determine the displacement field. There are also methods which enforce global consistency of the field. In this section we will review standard optical flow methods and two recent methods based on deep neural networks.

2.2.1 Lucas-Kanade

The Lucas-Kanade (LK) method [15] is an optical flow algorithm developed in 1981. The method assumes that the displacement of a pixel between successive frames is small and constant within a small neighborhood around the point. Furthermore, the method assumes that the brightness remains unchanged between frames. These assumptions can be expressed mathematically using the optical flow equations. Let the current pixel under consideration be (x, y, t) where t represent the current frame. Let the next frame occur in time Δt. If the pixel moves in the direction (v_x, v_y) then according to the brightness consistency equation

$$I(x, y, t) = I(x + v_x, y + v_y, t + \Delta t). \tag{2.9}$$

Expanding the right-hand side of (2.9) using a first-order Taylor series expansion we have:

$$I(x, y, t) = I(x, y, t) + \frac{\partial I}{\partial x} v_x + \frac{\partial I}{\partial y} v_y + \frac{\partial I}{\partial t} \Delta t. \tag{2.10}$$

From equation (2.10) we end up with the optical flow equation:

$$\frac{\partial I}{\partial x} v_x + \frac{\partial I}{\partial y} v_y = -\frac{\partial I}{\partial t} \Delta t. \tag{2.11}$$

To cope with the aperture problem, the Lucas-Kanade method solves for $\mathbf{v}$ using local least-squares matching in a neighborhood around the pixel under consideration. Let the pixel under consideration be a and the neighborhood of points be $P = \{p_1, \ldots, p_n\}$. The Lucas-Kanade method solves the following minimization problem:

$$\begin{bmatrix} v_x \\ v_y \end{bmatrix} = \underset{v_1, v_2}{\operatorname{argmin}} \sum_{p \in P} \left(I_x(p) v_1 + I_y(p) v_2 + I_t \Delta t \right)^2. \tag{2.12}$$

A common variant of this method is to weight the pixels closer to a more than those farther away in the window. The minimization problem (2.12) is modified as:

$$\begin{bmatrix} v_x \\ v_y \end{bmatrix} = \underset{v_1, v_2}{\operatorname{argmin}} \sum_{p \in P} w_p \left(I_x(p) v_1 + I_y(p) v_2 + I_t \Delta t \right)^2, \tag{2.13}$$

where w_p are the weights for each pixel. The optical flow equations can be written in matrix form and can be solved using the normal equations.

$$v = (A^\top W A)^{-1} A^\top W b \tag{2.14}$$

$$A = \begin{bmatrix} I_x(p_1) & I_y(p_1) \\ \vdots & \vdots \\ I_x(p_n) & I_y(p_n) \end{bmatrix} \quad v = \begin{bmatrix} v_x \\ v_y \end{bmatrix}$$

$$W = \begin{bmatrix} \sqrt{w_{p_1}} & 0 & \dots & 0 & \\ 0 & \sqrt{w_{p_2}} & 0 & \dots & 0 \\ 0 & \vdots & \vdots & \dots & 0 \\ 0 & 0 & 0 & \dots & \sqrt{w_{p_n}} \end{bmatrix} \quad b = \begin{bmatrix} -I_t(p_1) \\ -I_t(p_2) \\ \vdots \\ -I_t(p_n) \end{bmatrix}$$

In this report we use a modern version of the Lucas-Kanade method available in Piotr's Computer Vision Toolbox [16]. This version uses the weighted version of the Lucas-Kanade method and image pyramids. The algorithm uses a seven layer image pyramid with the highest level computing the flow on the image subsampled by 64 and refining the estimate in each level by warping the first image closer to the second. Best results were obtained by using a neighborhood of radius 10 pixels. The operation count on a 1024x640 image is shown in Table 2.1.

Table 2.1: Operation count for Lucas-Kanade to interpolate single frame of size 1024x640.

Layer Description	**Image Size**	**Number of Multiply-Adds**
Layer 1: Smoothen	16x10	3200
Layer 1: Matrix Calculations	16x10	82240
Layer 2: Smoothen	32x20	12800
Layer 2: Matrix Calculations	32x20	32960
Layer 3: Smoothen	64x40	51200
Layer 3: Matrix Calculations	64x40	131840
Layer 4: Smoothen	128x80	204800
Layer 4: Matrix Calculations	128x80	527360
Layer 5: Smoothen	256x160	819200

Layer 5: Matrix Calculations	256x160	2109440
Layer 6: Smoothen	512x320	3276800
Layer 6: Matrix Calculations	512x320	8437760
Layer 7: Smoothen	1024x640	13107200
Layer 7: Matrix Calculations	1024x640	33751040
Total Operation Count		6.05×10^7
Number of ops/pixel		92

2.2.2 Horn-Schunck

The Horn-Schunck (HS) method [17], introduced in 1980, is a global method for optical flow estimation. Instead of assuming that the motion vectors remain constant locally around a point, the method optimizes for the flow by introducing a global smoothness parameter. The algorithm minimizes the distortion in the flow and the smoothness can be controlled by specifying the weight of a regularization parameter. The optical flow equation (2.11), and smoothness requirement are ensured through the minimization of the following energy function:

$$E = (I_x v_x + I_y v_y + I_t)^2 + \alpha^2(\|\nabla v_x\|^2 + \|\nabla v_y\|^2), \tag{2.15}$$

where the first term ensures brightness consistency and the second term ensures global smoothness. The above equation can be discretized for implementation. The discretization is based on the central difference decomposition of the Laplacian [18]. Details about the discretization can be found in [17]. Thus, the energy function can be approximated by:

$$E = (I_x v_x + I_y v_y + I_t)^2 + \alpha^2((\overline{v_x} - v_x)^2 + (\overline{v_y} - v_y)^2), \tag{2.16}$$

where $\overline{v_x}$ and $\overline{v_y}$ are a weighted average of the gradient from the surrounding pixels. Taking the derivative with respect to v_x and v_y, we have

$$\frac{\partial E}{\partial v_x} = -2\alpha^2(\overline{v_x} - v_x) + 2(I_x v_x + I_y v_y + I_t)I_x, \tag{2.17}$$

$$\frac{\partial E}{\partial v_y} = -2\alpha^2(\overline{v_y} - v_y) + 2(I_x v_x + I_y v_y + I_t)I_y. \tag{2.18}$$

Solving the above two equations we get an expression for v_x and v_y which can then be used to solve for the motion vectors.

$$(\alpha^2 + I_x^2 + I_y^2)(v_x - \overline{v_x}) = -Ix(I_x\overline{v_x} + I_y\overline{v_y} + I_t), \tag{2.19}$$

$$(\alpha^2 + I_x^2 + I_y^2)(v_y - \overline{v_y}) = -Iy(I_x\overline{v_x} + I_y\overline{v_y} + I_t). \tag{2.20}$$

The above equation tells us that the optimal motion vector should be close to the average of its surrounding motion vectors, i.e. the field is smooth. This will in-turn enforce brightness consistency. In practice, the motion vectors are solved for iteratively, by calculating the weighted average and then updating the vectors alternately.

$$v_x^{n+1} = \overline{v_x}^n - \frac{Ix(I_x\overline{v_x}^n + I_y\overline{v_y}^n + I_t)}{(\alpha^2 + I_x^2 + I_y^2)}, \tag{2.21}$$

$$v_y^{n+1} = \overline{v_y}^n - \frac{Iy(I_x\overline{v_x}^n + I_y\overline{v_y}^n + I_t)}{(\alpha^2 + I_x^2 + I_y^2)}. \tag{2.22}$$

This algorithm is much more computationally intensive than the Lucas-Kanade method. The execution time depends on the smoothness of the image and the number of iterations required for convergence.

2.2.3 Black and Anandan

Black and Anandan introduced the idea of robust penalization for optical flow estimation in their paper in 1991 [19]. Until 1991, most optical flow algorithms, such as LK and HS, used the ℓ_2 norm (least squares) as the cost function or penalization function. The growth rate of the ℓ_2 norm is faster and it tends to not penalize erroneous estimates much more than accurate estimates. Black and Anandan proposed the use of a cost function that strongly penalizes large errors while rewarding accurate estimates. In particular, the cost function for LK and HS can be modified as shown in (2.23) and (2.24) respectively.

$$E_{LK} = \sum_{p \in P} \rho\left(I_x(p)v_1 + I_y(p)v_2 + I_t\Delta t, \sigma\right) \tag{2.23}$$

$$E_{HS} = \lambda_D\rho_D(I_xv_x + I_yv_y + I_t, \sigma_D) + \alpha^2(\rho_S(\overline{v_x} - v_x, \sigma_S) + \rho_S(\overline{v_y} - v_y, \sigma_S)). \tag{2.24}$$

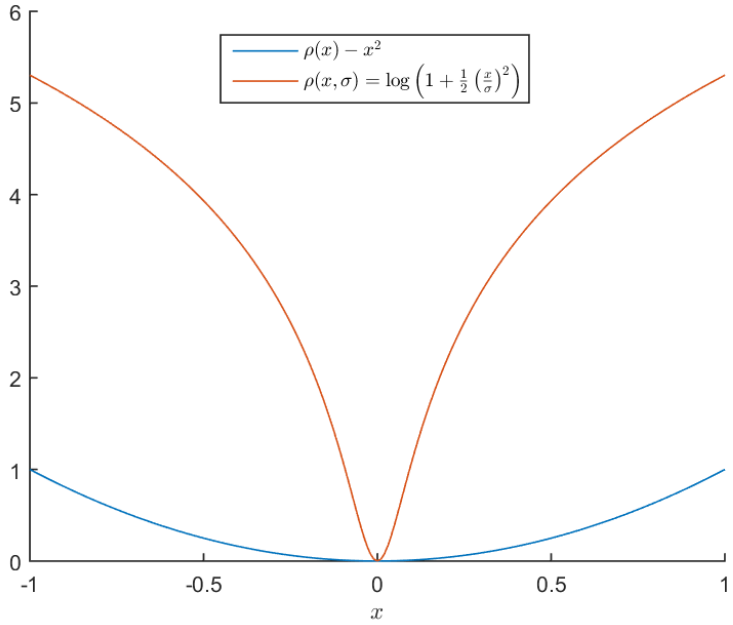


Figure 2.5: Lorentzian norm and ℓ_2 norm.

Here ρ_D, ρ_S are the robust norms and σ_D, σ_S are control parameters. Black and Anandan use the Lorentzian error norm for both ρ_D and ρ_S. Notice how it penalizes large errors in Figure 2.5.

The motion vectors can be solved for by minimizing the above cost functions using gradient descent. The derivatives of the cost function can be calculated the same way as in the LK or HS method. A detailed derivation of the gradient descent formulas can be found in [19]. The final equations are:

$$v_x^{(n+1)} = v_x^{(n)} - \omega \frac{1}{T(x,y)} \frac{\partial E}{\partial v_x}, \tag{2.25}$$

$$v_y^{(n+1)} = v_y^{(n)} - \omega \frac{1}{T(x,y)} \frac{\partial E}{\partial v_y}. \tag{2.26}$$

Here $T(x,y)$ is a bound on the second derivative of the cost function and ω is the step-size of the update. The Black and Anandan method heralded the use of robust penalization cost functions for optical flow estimation. Such cost functions are still used in many modern deep learning optical flow algorithms, the most popular error metric being the ℓ_1 norm.

2.2.4 MDP-Flow2 and Deepflow2

There are many more optical flow algorithms that can be discussed. Optical flow estimation is one of the most challenging computer vision problems to date and a thorough discussion of all algorithms would require many more pages. Further literature on optical flow can be found in [20, 21, 22]. It is worth mentioning two recent approaches to optical flow based on feature matching and neural networks.

Motion Detail Preserving Flow v2 (MDP-Flow2) [23] is currently one of the top ranked flow algorithms on the Middlebury flow rankings and it held the top spot for some time in 2010. The algorithm uses an image pyramid to predict flow on different levels (i.e. different amplitudes). In each level, the flow is computed by performing feature matching across the whole image and patch matching using 16×16 blocks. Following this, multiple flow candidates are generated to account for occlusions. A simple least squares optimization provides the best candidate. The flow is then made continuous by minimizing the Total Variation (TV) energy function. The image is then warped closer to the true version using the flow estimates and the estimates are continued on to the next level.

Deepflow 2 [24] performs a similar feature matching algorithm as in MDP-Flow2. The matching algorithm is called Deepmatching. It uses a pyramid structure with data movement interleaved between the levels, akin to the structure of a convolutional neural network. The cost function minimized is a robust penalizer and it is nearly the same as the BA cost function with an additional matching energy introduced from the Deepmatching algorithm. Deepflow2 is also ranked as one of the top optical flow methods on the Middlebury dataset.

An in-depth analysis of the algorithms can be found in the papers cited previously. Details have been left out as the analysis of these algorithms is not the goal of this thesis. It should be noted that the above algorithms have been specially optimized for the Middlebury dataset and are computationally expensive.

CHAPTER 3

FRAME INTERPOLATION

Frame interpolation refers to generating a new image given an observed input sequence. In MCFI, interpolation is performed by using the displacement field obtained by the motion estimation algorithm. In most applications, frame interpolation is performed by using only two reference frames. Better interpolation results can be obtained by using multiple images of the same scene. In such cases, the images can be mapped to the same frame of reference by using image registration techniques [25].

3.1 MCFI Problem Statement

In this thesis we will focus on frame interpolation with the only priors being two input frames. Let the input frames be I_t and I_{t+1}. Let $\mathbf{u}$ be the displacement field after performing motion compensation. The goal of MCFI is to generate an intermediate frame $I_{t+1/2}$ such that

$$I_{t+1/2}(\mathbf{x}) = I_t(\mathbf{x} - \mathbf{u}/\mathbf{2}), \tag{3.1}$$

$$I_{t+1/2}(\mathbf{x}) = I_{t+1}(\mathbf{x} + \mathbf{u}/\mathbf{2}). \tag{3.2}$$

The generation of this intermediate frame using the motion vectors is the task of frame interpolation. In the next few subsections we will review some common interpolation techniques. Finally, we will look at a recently developed CNN approach for video frame interpolation that estimates separable interpolation kernels.

3.2 Interpolation Algorithms

We will review five different image interpolation algorithms in this section. While images are defined on a discrete grid, we will assume that the grid is continuous to simplify the theory. Furthermore, we will assume that the points in the input image are spaced at a distance of 1 unit apart in the vertical and horizontal directions.

Image averaging is the only technique that requires two input images. All other interpolation techniques discussed in this section require a single input image.

3.2.1 Image Averaging

Image averaging is the simplest form of interpolation that one can perform between two images. The interpolated image I_{int} can be expressed in terms of the input images I_1 and I_2 as:

$$I_{int}(x, y) = \frac{I_1(x, y) + I_2(x, y)}{2}. \tag{3.3}$$

This interpolation is sufficient to use when there is very minimal or no motion between frames. This method tends to blend colors and creates artificial looking images with many artifacts.

3.2.2 Nearest Neighbor Interpolation

Nearest neighbor assigns the interpolated point with the value of its closest neighbor in the input image. For a 1D signal this can be carried out through convolution with the following kernel:

$$\varphi(x) = \begin{cases} 0 & x < -\frac{1}{2} \\ 1 & -\frac{1}{2} \leq x \leq \frac{1}{2} \\ 0 & x > \frac{1}{2} \end{cases}. \tag{3.4}$$

For a 2D signal, this kernel is applied in the horizontal and vertical direction in a separable fashion.

3.2.3 Bandlimited vs. Spline Interpolation

In traditional interpolation, a function f is approximated using the samples $f_n = f(nT)$, and a set of interpolation basis functions $\{\varphi_{int}\left(\frac{x}{T} - k\right)\}_{k\in\mathbb{Z}}$. The basis function satisfies $\varphi_{int}(n) = \delta_n$, $n \in \mathbb{Z}$, where δ_n is the Kronecker delta function. The approximated function can be constructed as follows:

$$f_{int}(x) = \sum_{k\in\mathbb{Z}} f_k \varphi_{int}\left(\frac{x}{T} - k\right). \tag{3.5}$$

In the case of signals bandlimited to $\left[-\frac{\pi}{T}, \frac{\pi}{T}\right]$, we can make $f_{int} = f$ by using $\varphi_{int} = \text{sinc}$. The sampling property and infinite support of the basis functions result in these functions being impractical to use in practice.

In generalized interpolation, the goal is to still approximate a function as closely as possible. However, the interpolation basis functions are no longer required to satisfy the sampling property. Furthermore, for computational purposes, basis functions with finite support are chosen. One particular example of such an interpolation basis function is a spline which is discussed in more detail in Section 3.2.4. (3.5) is modified as shown below:

$$f_{int}(x) = \sum_{k\in\mathbb{Z}} c_k \varphi\left(\frac{x}{T} - k\right). \tag{3.6}$$

Notice that the coefficients in (3.6) are no longer the sample values of the function. Thus, we need an additional constraint. We should require that the sample values at integer multiples of T can be reconstructed. Thus,

$$f_n = f(nT) = \sum_{k\in\mathbb{Z}} c_k \varphi\left(n - k\right) = (c * \varphi)_n. \tag{3.7}$$

In the z-domain, this relationship can be expressed as a filtering operation.

$$C(z) = \frac{1}{\sum_{n\in\mathbb{Z}} \varphi(n) z^{-n}} F(z), \tag{3.8}$$

where $F(z)$ is the z-transform of the samples of f. As long as $\sum_{n\in\mathbb{Z}} \varphi(n) z^{-n}$ has no roots on the unit circle, the above filtering operation can be implemented as forward-backward filtering operation using the causal and anti-causal parts of the basis function.

Thus, generalized interpolation can be split into two steps:

1. Determine the coefficients c_n using (3.8).

2. Perform interpolation using (3.6).

Generalized interpolation is often used because we have the choice of using a larger range of interpolation basis functions based on the application and system constraints. An important metric for assessing an interpolant's quality is know as the approximation order. An interpolant with degree of approximation L satisfies the following relation:

$$\|f - f_{int}\|_2 \leq C \times T^L. \tag{3.9}$$

Here, C is called the approximation gain and T is the time step used in sampling f.

3.2.4 Splines

Splines are smooth piecewise polynomials. A spline with degree K has continuous derivatives up to order $K-1$ and it is differentiable up to order K everywhere except at the piece boundaries. Since splines are piecewise, the expression of the polynomials that compose the spline change at different points. These points are called *knots*. We will only consider *uniform splines*, where the knots are evenly spaced out.

The most commonly used spline basis functions are the elementary B-splines. The elementary B-spline of degree 0 is given by:

$$\beta^{(0)}(x) = \begin{cases} 1 & -\frac{1}{2} \leq x < \frac{1}{2} \\ 0 & \text{otherwise} \end{cases}. \tag{3.10}$$

There are slight differences between the elementary B-spline of degree 0 and the nearest neighbor interpolation kernel; particularly, the spline averages values at half-integer points while the latter kernel does not [26]. The B-splines of higher degree can be derived from the elementary B-spline of degree

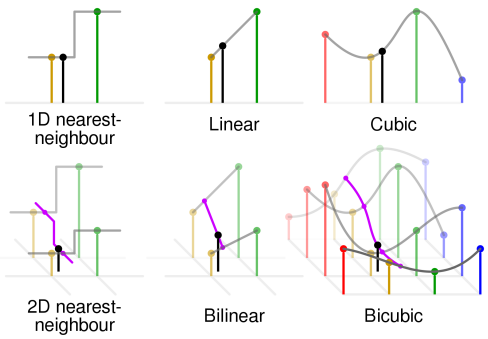


Figure 3.1: Illustration of different interpolation techniques.

0 by the following recursive formula:

$$\beta^{(n)}(x) = \beta^{(n-1)} * \beta^{(0)}(x). \tag{3.11}$$

Splines of degree n have an approximation order of $n+1$. This means that the function is very smooth. Splines also have the property of having the highest approximation order for a given support. A thorough review of splines can be found in [27]. Figure 3.1 [28] shows common interpolation techniques employed in image processing.

3.2.4.1 Linear Interpolation

Linear interpolation is performed by using the elementary B-spline of degree 1.

$$\beta^{(1)}(x) = \begin{cases} 1+x & 1 \leq x < 0 \\ 1-x & 0 \leq x < 1 \\ 0 & \text{otherwise} \end{cases}. \tag{3.12}$$

Notice that the range of the kernel is 2 units. In an image this would utilize up to three pixels in both the horizontal and vertical direction to compute the interpolated pixel. Interpolation is first carried out in the horizontal direction followed by interpolation in the vertical direction. This is commonly known as bilinear interpolation.

The coefficients c_n can be determined by solving (3.8). For an image we need to first evaluate $\varphi = \beta^{(1)}$ at all integer points in the basis function's support. Thus, $\varphi_n = 2\delta_{n+1} + \delta_n + 2\delta_{n-1}$. The coefficients can now be found by taking

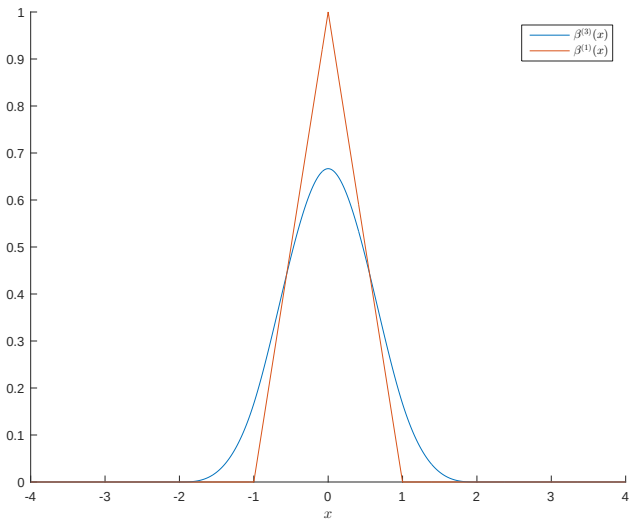


Figure 3.2: Linear and cubic elementary B-splines.

the inverse z-transform of $C(z)$.

$$C(z) = \frac{1}{1 + 2(z + 1/z)} F(z), \tag{3.13}$$

The above filtering operation can be expressed as a forward-backward filtering operation by observing that $1 + 2(z + 1/z) = \frac{1}{\lambda^2}(1 - \lambda z^{-1})(1 - \lambda z)$ where $\lambda = \frac{1}{4} i(\sqrt{15} + i)$.

3.2.4.2 Cubic Interpolation

Cubic interpolation is performed by using the elementary B-spline of degree 3.

$$\beta^{(3)}(x) = \begin{cases} \frac{2}{3} - \frac{1}{2}|x|^2(2 - |x|) & 0 \leq |x| < 1 \\ \frac{1}{6}(2 - |x|)^3 & 1 \leq |x| < 2 \\ 0 & \text{otherwise} \end{cases}. \tag{3.14}$$

Notice that the range of the kernel is 4 units. In an image this would utilize up to five pixels in both the horizontal and vertical direction to compute the interpolated pixel. Figure 3.2 illustrates the difference between the elementary linear and cubic spline kernels. Interpolation is first carried out in the horizontal direction followed by interpolation in the vertical direction. This is commonly known as bicubic interpolation.

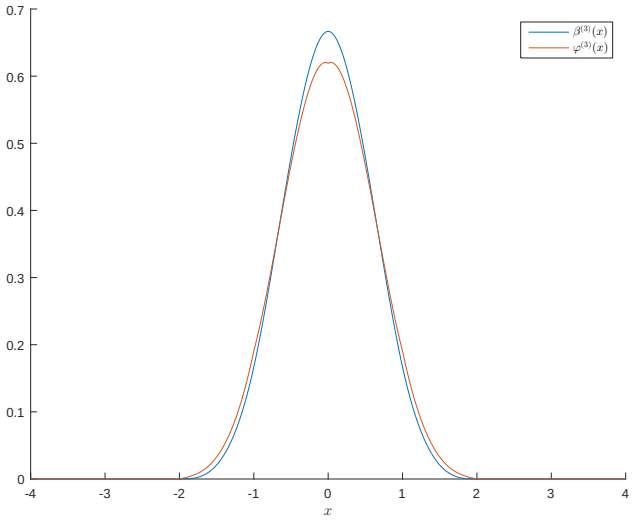


Figure 3.3: Cubic B-spline and OMOMS comparison.

The coefficients c_n can be determined by solving (3.8). For an image we need to first evaluate $\varphi = \beta^{(3)}$ at all integer points in the basis function's support. Thus, $\varphi_n = \frac{1}{6}\delta_{n+1} + \frac{2}{3}\delta_n + \frac{1}{6}\delta_{n-1}$. The coefficients can now be found by taking the inverse z-transform of $C(z)$ as was detailed in Section 3.2.4.1.

3.2.5 Cubic-OMOMS

Cubic splines fall under the category of interpolating functions known as Maximal-Order Interpolation of Minimal Support (MOMS) [29]. These are the class of functions that have the highest degree of approximation for a given support. The class of interpolants knows as Optimal Maximal-Order Interpolation of Minimal Support (OMOMS) [29] are those MOMS functions with the smallest approximation gain, C, in the ℓ^2 sense. These OMOMS functions can be expressed in terms of the elementary B-spline functions and its derivatives. The formula for the cubic-OMOMS interpolant is:

$$\varphi_o^{(3)}(x) = \beta^{(3)}(x) + \frac{1}{42}\frac{d^2}{dx^2}\beta^{(3)}(x) \tag{3.15}$$

$$\varphi_o^{(3)}(x) = \begin{cases} \frac{1}{2}|x|^3 - |x|^2 + \frac{1}{14}|x| + \frac{13}{21} & 0 \le |x| < 1 \\ -\frac{1}{6}|x|^3 + |x|^2 - \frac{85}{42}|x| + \frac{29}{21} & 1 \le |x| < 2 \\ 0 & \text{otherwise} \end{cases}. \tag{3.16}$$

The coefficient C is 4.6 times smaller for OMOMS in comparison to the similar coefficient for cubic splines. Figure 3.3 illustrates the difference in scaling between the elementary cubic spline and the cubic-OMOMS kernel. OMOMS have shown to produce slightly better interpolation results than MOMS functions [29].

3.3 Adaptive Separable Convolution

Instead of using optical flow estimates to perform frame interpolation, Niklaus *et al.* [30] use a deep convolutional neural network (CNN) to estimate separable convolution kernels that can be used to determine the intensity of each pixel in the output image. This method is known as adaptive separable convolution.

3.3.1 Problem Statement

The goal is to interpolate a frame I_{int} between two input frames I_1 and I_2. For each output pixel (x, y) in I_{int}, the CNN estimates a pair of separable filters K_1 and K_2 such that

$$I_{int}(x, y) = K_1(x, y) * P_1(x, y) + K_2(x, y) * P_2(x, y), \tag{3.17}$$

where P_1 and P_2 are patches of the images centered around the pixel (x, y) in the images I_1 and I_2 respectively. The separable filters can be represented as $\langle K_{1v}, K_{1h} \rangle$ and $\langle K_{2v}, K_{2h} \rangle$. Then,

$$K_i = K_{iv} \times K_{ih}^{\top}, \quad i = 1, 2 \tag{3.18}$$

Niklaus *et al.* use kernels of size 51×51. Thus, a total of 204 floating point values need to be stored per pixel. For a 1080p image, this would require $204 \times 1920 \times 1080 = 1.58$ GB memory.

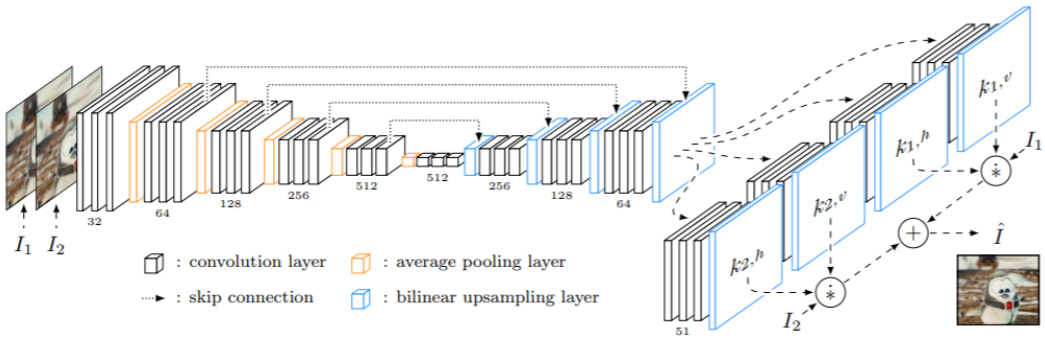


Figure 3.4: Illustration of neural network architecture. Given input frames I_1 and I_2, an encoder-decoder network extracts features that are given to four sub-networks that each estimate one of the four 1D kernels for each output pixel in a dense pixel-wise manner. The estimated pixel-dependent kernels are then convolved with the input frames to produce the interpolated frame I_{int}.

3.3.2 Network Architecture

The network has a contracting section followed by an expanding section as is shown in Figure 3.4 (The figure was taken from [30]). The layers are organized as a sequence of three convolutional layers followed by a ReLU activation layer. Skip connections are present between the contracting and expanding parts to facilitate the sharing of information before the downsampling operation and to improve the stability of the network at the upsampling layers. The authors use bilinear interpolation in the upsampling layers.

The authors trained the model with two loss functions. The first one is the $\mathcal{L}_1$ norm.

$$\mathcal{L}_1 = \|I_{int} - I_{gt}\|_1 \tag{3.19}$$

They also trained it with a perceptual loss function ($\mathcal{L}_F$). Perceptual loss is based on certain high level features that the user chooses to extract from the images. The `relu4_4` layer from the VGG19 network was used for the aforementioned purpose.

$$\mathcal{L}_F = \|\phi(I_{int}) - \phi(I_{gt})\|_2^2 \tag{3.20}$$

The authors trained the network for 20 hours on an NVIDIA Titan X (Pascal) GPU. A detailed breakdown of the various layers and operation count is

given in Table 3.1. The authors trained it on patches of images taken from common YouTube videos using AdaMax.

The operation count was separated into multiply-additions and in-place computations to ease the counting process. The authors implemented the last two layers in CUDA to perform fast computations.

Table 3.1: Operation count for Adaptive Separable Convolution to interpolate single frame of size $960 \times 540 \times 3$.

Layer	**Output Size**	**Multiply-Adds**	**Add, Div etc.**
Input	(1024, 640, 6)	0	0
Conv + ReLU	(1024, 640, 32)	1132462080	20971520
Conv + ReLU	(1024, 640, 32)	6039797760	20971520
Conv + ReLU	(1024, 640, 32)	6039797760	20971520
Avg. Pool	(512, 320, 32)	0	20971520
Conv + ReLU	(512, 320, 64)	3019898880	10485760
Conv + ReLU	(512, 320, 64)	6039797760	10485760
Conv + ReLU	(512, 320, 64)	6039797760	10485760
Avg. Pool	(256, 160, 64)	0	10485760
Conv + ReLU	(256, 160, 128)	3019898880	5242880
Conv + ReLU	(256, 160, 128)	6039797760	5242880
Conv + ReLU	(256, 160, 128)	6039797760	5242880
Avg. Pool	(128, 80, 128)	0	5242880
Conv + ReLU	(128, 80, 256)	3019898880	2621440
Conv + ReLU	(128, 80, 256)	6039797760	2621440
Conv + ReLU	(128, 80, 256)	6039797760	2621440
Avg. Pool	(64, 40, 256)	0	2621440
Conv + ReLU	(64, 40, 512)	3019898880	1310720
Conv + ReLU	(64, 40, 512)	6039797760	1310720
Conv + ReLU	(64, 40, 512)	6039797760	1310720
Avg. Pool	(32, 20, 512)	0	1310720
Conv + ReLU	(32, 20, 512)	1509949440	327680
Conv + ReLU	(32, 20, 512)	6039797760	327680
Conv + ReLU	(32, 20, 512)	6039797760	327680
Upsample Bilinear	(64, 40, 512)	11796480	0
Conv + ReLU	(64, 40, 512)	6039797760	1310720

Conv + ReLU	(64, 40, 256)	3019898880	655360
Conv + ReLU	(64, 40, 256)	1509949440	655360
Conv + ReLU	(64, 40, 256)	1509949440	655360
Upsample Bilinear	(128, 80, 256)	23592960	0
Conv + ReLU	(128, 80, 256)	6039797760	2621440
Conv + ReLU	(128, 80, 128)	3019898880	1310720
Conv + ReLU	(128, 80, 128)	1509949440	1310720
Conv + ReLU	(128, 80, 128)	1509949440	1310720
Upsample Bilinear	(256, 160, 128)	47186920	0
Conv + ReLU	(256, 160, 128)	6039797760	5242880
Conv + ReLU	(256, 160, 64)	3019898880	2621440
Conv + ReLU	(256, 160, 64)	1509949440	2621440
Conv + ReLU	(256, 160, 64)	1509949440	2621440
Upsample Bilinear	(512, 320, 64)	94371840	0
Conv + ReLU	(512, 320, 64)	6039797760	10485760
Separable Filters	(1024, 640, 51)	10^{11}	10^{8}
Output	(1024, 640, 3)	2.5×10^{11}	0
Total Operation Count		4.89×10^{11}	
Number of ops/pixel		314429	

CHAPTER 4

LOCAL ALL-PASS FILTERS

Most traditional optical flow algorithms estimate the optical flow using the optical flow equation defined in (2.11). They either enforce this constraint by assuming a locally constant flow and thus a small displacement field (e.g. LK) or by enforcing global consistency (e.g. HS). However, even in many cases when the optical flow field is smooth, the displacement amplitude between two images might not be small. Gilliam *et al.* solve this problem by relating local changes through an all-pass filtering framework known as the Local All-Pass algorithm (LAP) [31, 32, 33, 34]. In the rest of this chapter we will summarize the LAP algorithm detailed in [34].

4.1 Main Principles of the LAP Framework

The All-Pass filter framework is built upon five guiding principles which are detailed below.

4.1.1 Shifting is All-Pass Filtering

If the brightness consistency holds, shifting an image in the time domain in the direction of a constant displacement $\mathbf{u}(x, y) = (u_1, u_2)^\top$ can be expressed in the frequency domain as

$$\hat{I}_2(\omega_1, \omega_2) = \hat{I}_1(\omega_1, \omega_2) e^{-j\omega_1 u_1 - j\omega_2 u_2}, \tag{4.1}$$

where $\hat{I}$ is the Fourier transform of I. The previous expression can be interpreted as filtering I_1 with a filter h having frequency response

$$\hat{h}(\omega_1, \omega_2) = e^{-j\omega_1 u_1 - j\omega_2 u_2}. \tag{4.2}$$

This is a filter that is separable, real valued and all-pass i.e. $|\hat{h}(\omega_1, \omega_2)| = 1$. Notice that the flow displacement information is contained within the phase of this filter. Thus, once this filter is estimated, the optical flow can be extracted through simple differentiation operations. While the filter discussed above is continuous, the same properties hold in the discrete domain under ideal sampling of h with the sinc kernel.

4.1.2 Rational Representation of All-Pass Filter

Any real all-pass filter can be expressed as the ratio of two real digital filters with the same amplitude and opposite phase. If the desired phase of the all-pass filter $\hat{h}(\omega_1, \omega_2)$ is $2\arg(\hat{p}(\omega_1, \omega_2))$, then

$$\hat{h}(\omega_1, \omega_2) = \frac{\hat{p}(\omega_1, \omega_2)}{\hat{p}(-\omega_1, -\omega_2)}, \tag{4.3}$$

where $\hat{p}(\omega_1, \omega_2)$ is the forward filter and $\hat{p}(-\omega_1, -\omega_2)$ is the backward filter. Now (4.1) can be rewritten as

$$\hat{I}_2(\omega_1, \omega_2) = \hat{I}_1(\omega_1, \omega_2) \frac{\hat{p}(\omega_1, \omega_2)}{\hat{p}(-\omega_1, -\omega_2)} \tag{4.4}$$

In the time domain this can be expressed as a forward-backward filtering relation:

$$I_2[k, l] * p[-k, -l] - I_1[k, l] * p[k, l] = 0, \tag{4.5}$$

where (k, l) are discrete pixel coordinates. Thus, estimating h boils down to estimating p.

4.1.3 Linear Approximation of Forward Filter

Gilliam and Blu [31, 34] use a standard signal processing technique to approximate the forward filter p using a set of fixed real filter p_n. The approximation can be expressed as:

$$p_{app}[\mathbf{k}] = \sum_{n=0}^{N-1} c_n p_n[\mathbf{k}], \tag{4.6}$$

where $\mathbf{k} = (k, l)^\top$. Estimating p now only involves finding the appropriate coefficients c_n. This can be done via a simple least squares minimization procedure which will be discussed later. The authors also note that while the separability of the all-pass filter is lost, the filter is still real.

4.1.4 Choosing a Good Filter Basis

The filter basis was chosen keeping two important points in mind:

1. The number of filters N must be much smaller than the number of pixels in the image.
2. The number of filters N should be independent of the priori assumptions of the displacement amplitude R. In other words, the filter should be able to scale easily to estimate different displacements.

A naive choice for the filter basis would be canonical representation of finite impulse response filters (FIR) over the range $[-R, R]$, i.e. the square of side $2R + 1$:

$$\hat{p}_n = e^{-j\omega_1 k_n - j\omega_2 l_n}, \tag{4.7}$$

where $k_n, l_n \in [-R, R]$. This is not a good choice since estimating a displacement of R would require $N = \mathcal{O}(R^2)$ basis filters.

After experimenting with different empirical methods, Gilliam and Blu observed that the forward filters approximated from these methods were closely related to a Guassian filter and its derivatives. The basis filters are divided into two categories: 1) $K = 1$ (Gaussian filter and first derivatives) which results in $N = 3$ filters or 2) $K = 2$ (Gaussiam filter and first, second derivatives) which results in $N = 6$ filters. This filter basis is shown in (4.8).

$K = 1$ includes $p_i[\mathbf{k}], i = 0, 1, 2$ and $K = 2$ includes $p_i[\mathbf{k}], i = 0, 1, 2, 3, 4, 5$. The standard deviation $\sigma = (R + 2)/4$ where R is the half-support of the

filter. Notice that these filters can be easily scaled for large displacements.

$$K=1\begin{cases} p_0[\mathbf{k}] = \exp\left(-\dfrac{k^2+l^2}{2\sigma^2}\right) \\ p_1[\mathbf{k}] = kp_0[\mathbf{k}] \\ p_2[\mathbf{k}] = lp_0[\mathbf{k}] \end{cases} \begin{cases} \\ \\ \\ p_3[\mathbf{k}] = (k^2+l^2-2\sigma^2)p_0[\mathbf{k}] \\ p_4[\mathbf{k}] = klp_0[\mathbf{k}] \\ p_5[\mathbf{k}] = (k^2-l^2)p_0[\mathbf{k}] \end{cases} K=2 \tag{4.8}$$

The theoretical properties of these filters and the approximation order of the filter basis are detailed in [33]. Using Padé approximations it was shown that the approximation order of the filter basis is $L = 2K$. Thus, in the case of $K = 1$ the approximation order is 2 and for $K = 2$ the approximation order is 4. This is a promising result because most state-of-the-art optical flow methods have an approximation order of 1.

4.1.5 Displacement Vector from All-Pass Filter

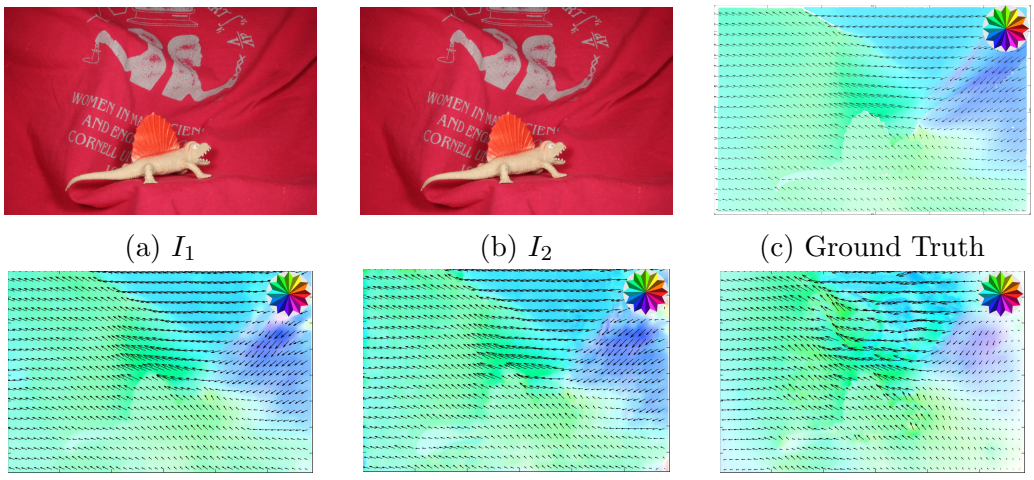
Since the all-pass filter h_{app} is expected to be close to (4.2), the displacement vectors can be extracted from the frequency response of the filter as:

$$u_{1,2} = j\frac{\partial \log(\hat{h}_{app}(\omega_1,\omega_2))}{\partial\omega_{1,2}}\bigg|_{\omega_1=\omega_2=0}. \tag{4.9}$$

Using the rational representation of the all-pass filter, the displacement vectors can be found using a simple formula in terms of the impulse response of the forward filter:

$$u_1 = 2\frac{\sum_{\mathbf{k}} kp[\mathbf{k}]}{\sum_{\mathbf{k}} p[\mathbf{k}]} \quad \text{and} \quad u_2 = 2\frac{\sum_{\mathbf{k}} lp[\mathbf{k}]}{\sum_{\mathbf{k}} p[\mathbf{k}]}, \tag{4.10}$$

where the summation is over all discrete points in the region of support of the signal.



(a) I_1 (b) I_2 (c) Ground Truth

(d) LAP (e) Lucas-Kanade (f) Horn-Schunck

Figure 4.1: Estimated flow using different methods. A radius of 10 was used for LK and HS. HS required 250 iterations for convergence. LAP flow was estimated using all six basis filters.

4.2 Estimation of All-Pass Filter

The estimation of the all-pass filter is performed by adapting (4.5) locally around every pixel in a window, $\mathcal{W}$, of size $2W+1 \times 2W+1$. In the implementation of the algorithm $W = R$ except when $R < 1$. Gilliam and Blu coin a new relation known as *local all-pass equation* defined as follows:

$$p_{app}[\mathbf{k}] * I_1[\mathbf{k}] = p_{app}[-\mathbf{k}] * I_2[\mathbf{k}], \quad \mathbf{k} \in \mathcal{W}, \tag{4.11}$$

where $p_{app}[\mathbf{k}]$ is given in (4.6).

To enforce the local all-pass equation, the difference between the left and right side in (4.11) can be minimized in the ℓ_2 sense. Defining $\langle J \rangle_{\mathcal{W}} = \sum_{\mathbf{k}\in\mathcal{W}} J[\mathbf{k}]$ and $\overline{p}_{app}[\mathbf{k}] = p_{app}[-\mathbf{k}]$, the minimization problem can be written as:

$$\min_{\{c_n\}} \langle (p_{app} * I_1 - \overline{p}_{app} * I_2)^2 \rangle_{\mathcal{W}}, \tag{4.12}$$

$$\text{where} \quad p_{app}[\mathbf{k}] = p_0[\mathbf{k}] + \sum_{n=1}^{N-1} c_n p_n[\mathbf{k}].$$

Since the minimum of (4.12) is an orthogonal projection it satisfied the fol-

lowing equations in c_n:

$$0 = \langle (p_n * I_1 - \overline{p}_n * I_2)(p_0 * I_1 - \overline{p}_0 * I_2) \rangle_{\mathcal{W}} + \sum_{n'=1}^{N-1} c_{n'} \langle (p_n * I_1 - \overline{p}_n * I_2)(p_{n'} * I_1 - \overline{p}_{n'} * I_2) \rangle_{\mathcal{W}} \tag{4.13}$$

for $n = 0, 1, \ldots, N-1$.

The previous equations can be solved very efficiently using pointwise multiplications and fast convolutions. Gilliam and Blu solve the system of equations using Gaussian elimination which has a constant number of operations per each pixel.

4.3 Poly-Filter Extension

The LAP algorithm is able to estimate large displacements, but it requires a filter basis with large support. To estimate smoothly varying flows, iterative refinement is performed to estimate the flow at various levels i.e., flows with varying amplitude. Instead of using image pyramids, Gilliam and Blue use filter pyramids. In each iteration of the algorithm, the support of the filter is changed. Thus, the only parameter that changes is R. This extension is termed as Poly-Filter LAP (PF-LAP). The LAP and PF-LAP algorithms from [34] have been reproduced as Algorithm 1 and 2 respectively in this thesis for sake of completion.

PF-LAP is implemented as follows: The user can choose the maximum support R_{max}. Create the filter basis for this support and estimate the deformation field using the LAP algorithm. Warp I_2 closer to I_1 using this deformation field. Perform post-processing on this deformation field to remove NaNs and to smoothen the field. In the next iteration decrease the support width by a factor of 2 and repeat the procedure again. This process continues until $R = 1$. Finally, one more iteration is performed with $R = 0.5$ to accurately estimate any subpixel flows. The operation count for the LAP algorithm to interpolate a frame of size 960x540x3 is shown in Table 4.1.

Algorithm 1 Local All-Pass (LAP estimation of a deformation field)

Inputs: Images I_1 and I_2, filter size R and number of filters N

1: Initialization: Generate the filter basis in (4.8) using R and N.
2: Filter Estimation: Using the filter basis, solve the minimization problem by using (4.13) to obtain the filter coefficients.
3: Extraction: Using the filter coefficients obtain the approximate forward filter. Use (4.10) to solve for the displacement field.

Algorithm 2 Poly-Filter Local All-Pass algorithm

Inputs: Images I_1 and I_2, number of levels R and number of filters N

1: Initialization: Set $u_{2^{R+1}} = \mathbf{0}$ and $I_2^{\text{shift}} = I_2$.
2: **for** R in $[2^R, 2^{R-1}, \ldots, 0.5]$ **do**
3: Pre-Filtering (OPTIONAL): Construct filter p_0 from (4.8) and obtain high-pass images I_1^{hp} and I_2^{hp}.
4: Estimation: Using N and R estimate the deformation increment Δu between the images I_1 and I_2 (or high-pass images if $R < 1$) using the LAP algorithm defined in Algorithm 1.
5: Post-Processing: Using inpainting procedure and median filtering to remove erroneous estimate. Then smooth the flow using a Gaussian filter.
6: Update: Set $u_R = u_{2R} + \Delta u$
7: Warping: Warp I_2 closer to I_1 using u_R to obtain I_2^{shift}. If $R \leq 2$ use cubic-OMOMS interpolation else use shifted-linear interpolation.
8: **end for**

4.4 Pre-Processing and Post-Processing Tools

In PF-LAP, image pre-processing is performed to reduce the effect of slowly varying illumination changes at very course levels. Specifically, during sub-pixel flow estimation the high-pass filtered images are used. This is done by low pass filtering the images using p_0 and subtracting this from the original image:

$$I_i^{hp} = I_i - p_0 * I_i. \tag{4.14}$$

At the end of each iteration in PF-LAP, I_2 is warped towards I_1 using the estimated field. High-quality interpolation is achieved using cubic splines or cubic-OMOMS (see Sections 3.2.4.2 and 3.2.5). To ensure fast interpolation, shifted-linear interpolation (see Section 3.2.4.1) is performed for $R > 2$ and cubic-OMOMS is used for $R \leq 2$. To remove erroneous estimates a round

of median filtering and simple convolutional image inpainting is performed on the displacement field. Finally, the flow is smoothened using a Gaussian filter with variance half the support.

In the remainder of the thesis we will refer to PF-LAP as LAP to refer to the overall algorithm.

Table 4.1: Operation count for LAP to interpolate a single frame of size 960x540x3. Below, ma stands for multiply-adds and the cleaning procedures involve NaN inpainting, mean and median filtering. Flow estimation requires 3 subtractions, 2 divisions and 1 addition per pixel.

Function	**No. of Ops**
Image pre-filtering with high-pass filter (3x3x3)	13996800
Level 1 Basis filtering (17x17x3)	1348358400
Level 1 Gaussian Elimination (5 ma)	7776000
Level 1 Flow estimate	3110400
Level 1 Cleaning Procedures	9331200
Level 1 Shifted Linear Interpolation (2 significant ma)	3110400
Level 2 Basis filtering (9x9x3)	377913600
Level 2 Gaussian Elimination (5 ma)	7776000
Level 2 Flow estimate	3110400
Level 2 Cleaning Procedures	9331200
Level 2 Shifted Linear Interpolation (2 significant ma)	3110400
Level 3 Basis filtering (5x5x3)	116640000
Level 3 Gaussian Elimination (5 ma)	7776000
Level 3 Flow estimate	3110400
Level 3 Cleaning Procedures	9331200
Level 3 Shifted Linear Interpolation (2 significant ma)	3110400
Level 4 Basis filtering (5x5x3)	116640000
Level 4 Gaussian Elimination (5 ma)	7776000
Level 4 Flow estimate	3110400
Level 4 Cleaning Procedures	9331200
Level 4 Cubic Interpolation (6 significant ma)	77760000
Total Operation Count	1.52×10^9
Number of ops/pixel	977

CHAPTER 5

VIDEO INTERPOLATION RESULTS

In this chapter we quantitatively and qualitatively assess the video interpolation quality obtained using three methods: LAP (Chapter 4), Lucas-Kanade (Section 2.2.1) and Adaptive Separable Convolution (Section 3.3). To ease notation we will refer to Lucas-Kanade as LK and Adaptive Separable as CNN (since it is based on a CNN). Note that the $K = 1$ version of the LAP algorithm is used (Section 4.3).

We perform video interpolation on Derf's Media Collection [35] and the EPIC Kitchen's dataset [36]. Both these datasets contain sequences captured at a temporal resolution of 60 frames per second which is ideal for studying interpolation. We also perform interpolation experiments on the Middlebury dataset [22], a standard dataset used for benchmarking a variety of image processing algorithms. MDP-Flow2 and Deepflow2 (Section 2.2.4) interpolation is also performed to assess the quality of the top performers on the Middlebury benchmark.

Finally, we will comment on the correlation between quantitative metrics and qualitative metrics by conducting a user study on a sample of the video sequences.

All experiments were performed on an Intel Core i7-8750H processor with 32 GB RAM. The CNN was executed on an NVIDIA GTX 1050Ti GPU with 4GB RAM. The interpolation results can be viewed at `https://bit.ly/2WqXbKR`. Sequences should be downloaded else the media player will drop the intermediate frames!

Table 5.1: MSE evaluation on the Middlebury dataset (high-speed camera samples). **Bold** values indicate best results.

	Beanbags	DogDance	MiniCooper	Walking	Backyard	Basketball	Dumptruck	Evergreen
LAP	339.0	240.1	176.7	67.9	163.5	198.9	209.3	268.6
CNN	**196.9**	**159.4**	**80.5**	**57.2**	**102.6**	**105.8**	**88.3**	**102.8**
LK	454.7	223.9	233.2	97.3	273.5	157.2	281.7	276.8

5.1 Performance Evaluation Criteria

In addition to computational speed, it is desirable to employ an objective performance metric to evaluate the match between the interpolated frame and the ground truth. The standard criterion is the ubiquitous Mean-Squared Error (MSE) [6]. Unfortunately MSE is a particularly bad criterion for measuring the quality of interpolated images, as slight misalignments can be quite acceptable perceptually yet cause large squared errors [37]. We have experimented with other metrics such as SSIM [38] and CW-SSIM [39] but they suffer from similar artifacts. Therefore we rely extensively on visual evaluation to assess performance of competing algorithms. The discrepancy between MSE and perceptual quality is often striking.

5.2 Middlebury Dataset

MSE results for the Middlebury dataset are given in Table 5.1. Although the CNN seems to perform the best interpolation in terms of MSE, we observe that actually the LAP consistently interpolates frames with the highest perceptual quality. This is evident from Figure 5.1: Notice how the palms are smudged and how the balls are distorted in the CNN interpolated frame. Yet, since the imprints of the middle two balls are slightly closer to the actual position of the balls, the CNN interpolated frame has a lower MSE. There is also a greater intensity match between the pixels in the original image and the CNN-interpolated image in comparison to the LAP-interpolated images.

The Lucas-Kanade method performed poorly on all sequences. The flow field across the juggler's torso is incorrect. The fingers are deformed in Figure 5.1, and the blinds are skewed due to inexact optical flow estimates. This is evident from the patch of blue on the door in the zoomed in image of the balls.

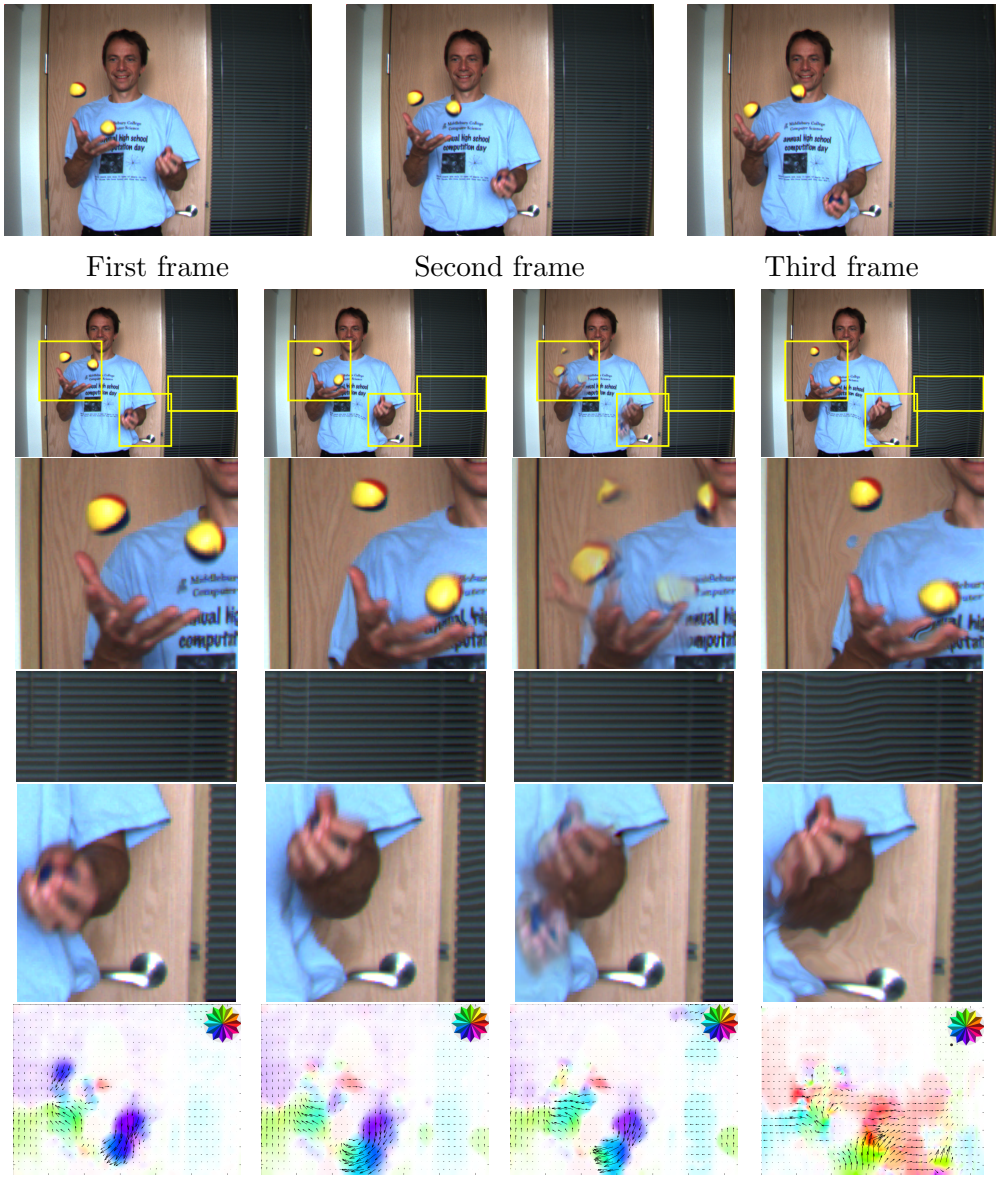


Figure 5.1: Beanbags sequence: The original sequence is shown at the top. The original second frame is shown in (a), the LAP-interpolated second frame is shown in (b), the CNN-interpolated second frame is shown in (c), the LK-interpolated second frame is shown in (d). LAP was used to compute the optical flow in (a), (b) and (c).

Table 5.2: Average interpolation MSE on Derf's Media Collection. **Bold** values indicate best results.

	LAP	CNN	LK
city	**62.1**	79.8	111.7
crew	522.1	**406.1**	806.5
harbour	112.2	**94.8**	117.1
ice	129.6	**57.4**	109.9
soccer	848.3	**141.6**	399.5
stockholm	**54.6**	60.4	77.6
riverbed	**361.7**	409.6	445.8

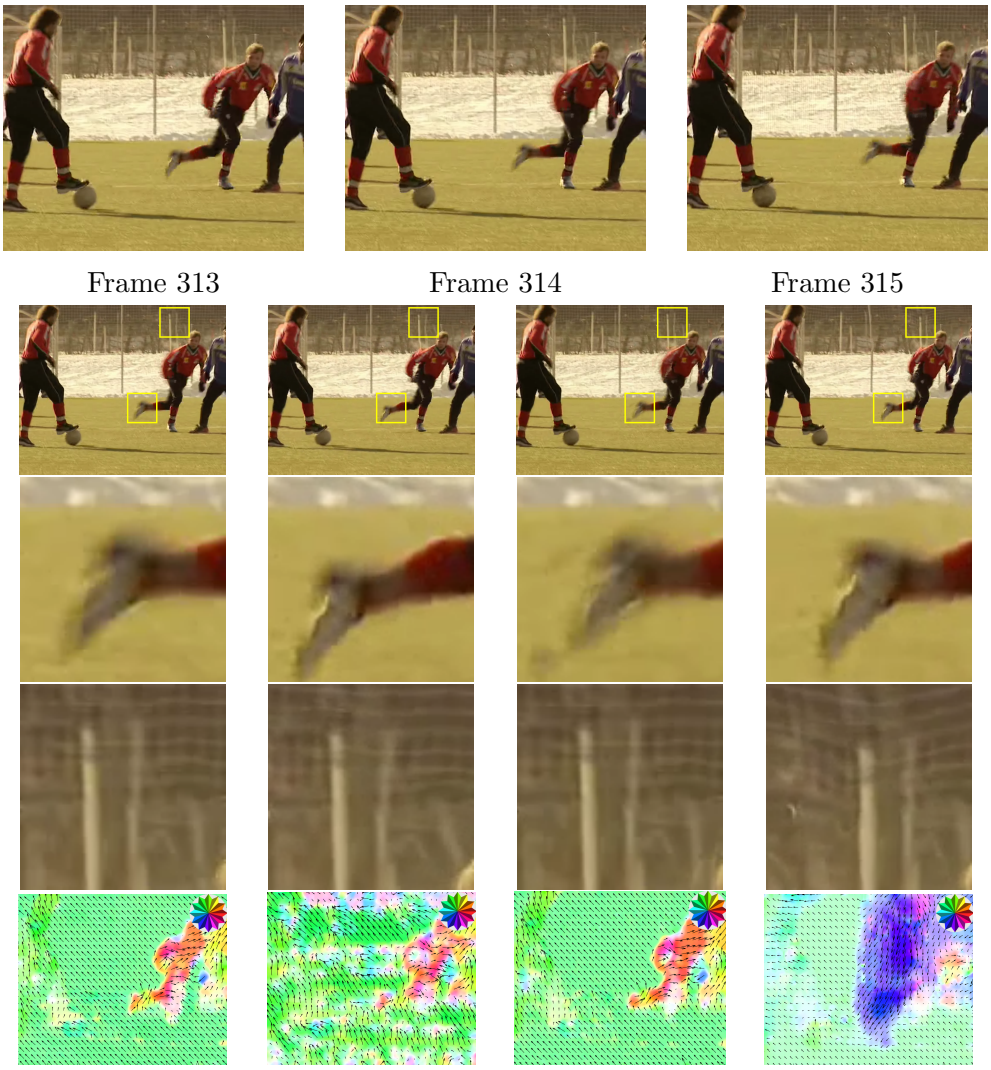
Also notice how the left side of the juggler's torso has been warped inwards.

The displacement field for the LAP algorithm is the closest to the flow between the original frames. The optical flow between the original frame and CNN-interpolated frame is not as smooth in the blue-yellow region boundary in comparison to the flow between the original frame and LAP-interpolated frame. This results in the juggler's palms being blurred in the CNN-interpolated frame.

5.3 Derf's Media Collection

This dataset consists of video sequences which are generally used to evaluate compression algorithms. Table 5.2 shows the interpolation results on seven 60 fps videos with 704x576 spatial resolution. We dropped every other frame in the video sequence and interpolated between the remaining pairs.

Table 5.2 shows MSE results, and Fig. 5.2 shows frames for the soccer sequence. While the LAP method yields good visual quality, a visible artifact is the position of the running player in the LAP-interpolated frame, which is slightly to the left of his position in the original frame. The player's leg is also slightly lower and farther away from his body in comparison to the original frame. Still, when viewed as a sequence, the LAP-interpolated video looked natural and smooth.



(a) Original (b) LAP (c) CNN (d) LK

Figure 5.2: Soccer sequence: The original sequence is shown at the top. Original frame 314 is shown in (a), LAP-interpolated frame 314 is shown in (b), CNN-interpolated frame 314 is shown in (c) and the LK interpolated 314 is shown in (d). The optical flow between frame 314 and frame 314 is shown for the original sequence in (a) and for the three methods in the other columns. LAP was used to compute the optical flow in (a), (b) and (c).

The Lucas-Kanade method did not compute a very precise flow field. Notice how the displacement vectors are pointing south across the running player. This results in the fence being highly deformed in the interpolated frames. When the interpolated frames are played at 60 frames per second, this causes flickering and strenuous visual artifacts. The neural network almost aligns

Table 5.3: Average interpolation MSE on EPIC Kitchens dataset. **Bold** values indicate best results.

	LAP	CNN	LK
P01_11	163.8	**130.2**	149.5
P01_12	**151.2**	199.0	204.1
P01_14	**152.6**	234.6	251.0
P01_15	**163.2**	273.4	301.9
P02_13	518.4	**458.9**	541.6
P03_21	**193.2**	433.1	491.9
P03_22	**307.1**	494.6	565.8
P03_23	**266.9**	462.1	537.9

the interpolated frame with the original frame. However, it blurs the limbs and leaves an imprint of the previous position of the limbs. This appears as a shadow around the limbs when viewed as a sequence.

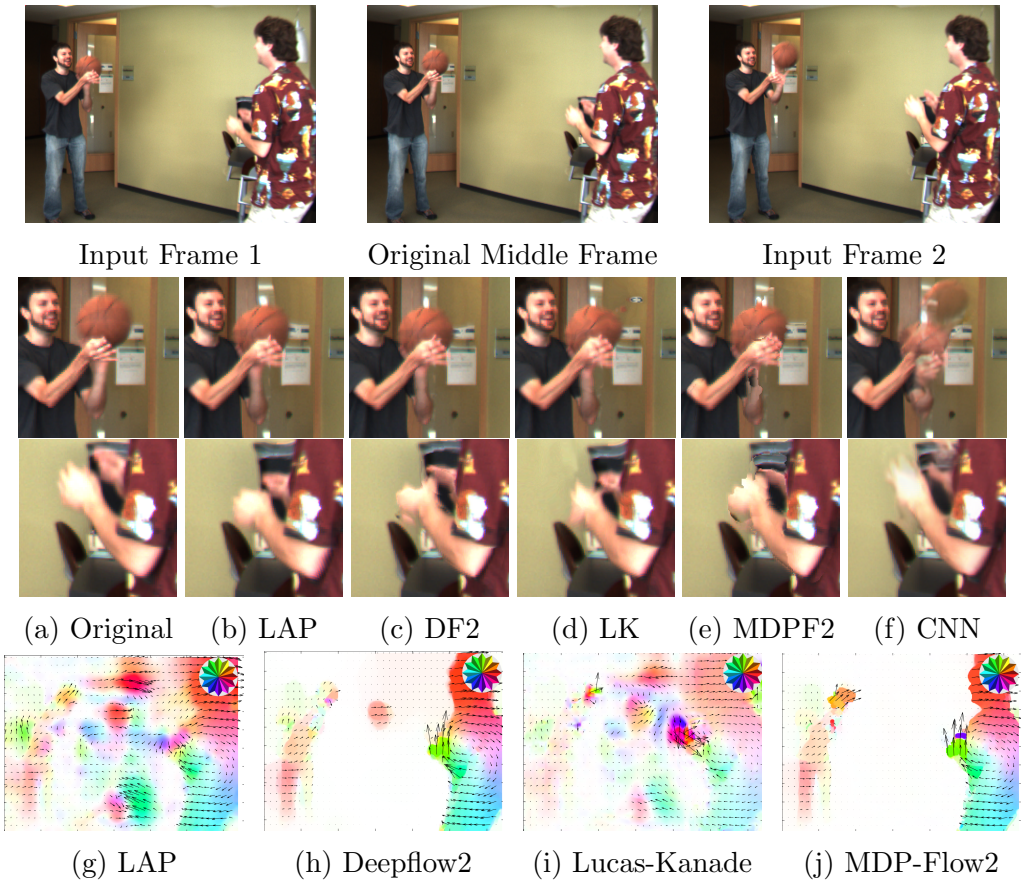
5.4 EPIC Kitchens Dataset

Finally, we perform interpolation on eight full-HD sequences from the EPIC Kitchens dataset. Since our GPU has insufficient memory to interpolate a 1920x1080 image, the spatial resolution of the video sequences was reduced to 960x540. The video sequences consist of body cam footage of people navigating around kitchens. The sequences consist of crisp fast motion such as bending down to open cabinets, cutting vegetables and sudden changes in direction. The LAP algorithm performs very well in preserving these motions. The CNN produces artifacts and jerkiness in the frames when there is a sudden change in direction or fast motion. The Lucas-Kanade method also produces similar artifacts which are more pronounced than the CNN interpolated frames.

The average interpolation MSE is reported in Table 5.3. Here the lower MSE for LAP correlates with the perceptual quality on this dataset. Since the videos have high spatial resolution, there is much more smoothness between consecutive frames. The LAP algorithm computes high accuracy optical flow estimates on smooth flowing sequences and as a result outperforms both the

Table 5.4: Execution times and MSE of the various optical flow methods and CNN on the basketball sequence. All times are in seconds.

	LAP	LK	Deepflow2	MDP-Flow2	CNN
Execution Times	4.3	0.2	50	294	1
MSE	199.7	160.7	146.9	137.0	105.8



Input Frame 1 Original Middle Frame Input Frame 2

(a) Original (b) LAP (c) DF2 (d) LK (e) MDPF2 (f) CNN

(g) LAP (h) Deepflow2 (i) Lucas-Kanade (j) MDP-Flow2

Figure 5.3: Basketball sequence (frames 9, 10, 11): The original sequence is shown at the top. The original sequence is shown in (a). In (c) DF2 stands for Deepflow2 and in (e) MDPF2 stands for MDP-Flow2. Flow fields between frame 1 and frame 2 is shown at the bottom.

CNN and Lucas-Kanade in interpolating between fast motion frames.

Table 5.5: Number of operations (multiply-adds) and execution time (CPU) required to interpolate a frame of size 960x540x3. Execution time is in seconds, measured on a machine with Intel Core i7-8750H processor and 32 GB RAM. The execution time for the CNN was extrapolated based on the time required to interpolate 200 pixels.

	CNN	LAP	LK
Operation Count	4.9×10^{11}	1.5×10^{9}	6.1×10^{7}
Execution Time	76204	4.3	0.2

5.5 Case Study: Basketball Sequence

We compare LAP against two additional optical flow methods: Deepflow2 and MDP-Flow2. These two methods are ranked high on the Middlebury flow/interpolation rankings. The flows for LAP, Deepflow2 and MDP-Flow2 are identical across both the players' body. The LAP returns a larger flow field in the center of the frame, but this flow is insignificant as the majority of the frame in this region is occupied by the wall which is shaded in a single color.

LAP also approximates a much smoother flow for the hands of the player on the right. The interpolated hands thus look the best and most natural in the LAP interpolated frame. Notice how MDP-Flow2 poorly interpolates the left side player's arm, which is evident from the large flows in this region.

The code for Deepflow2 and MDP-Flow2 was taken from the authors' websites. It is important to note that code was run using the optimizations and parameters that the authors used for their Middlebury evaluation submission. LAP has no prior optimizations for any dataset.

5.6 User Study

It is clear from the results of the Middlebury dataset, Derf's Media Collection and the case study on the basketball sequence that MSE does not correlate well with subjective visual quality. In order to better understand the interpolation results obtained, a user study was conducted. The users were asked to study the interpolated sequences using LAP, CNN and LK, and choose

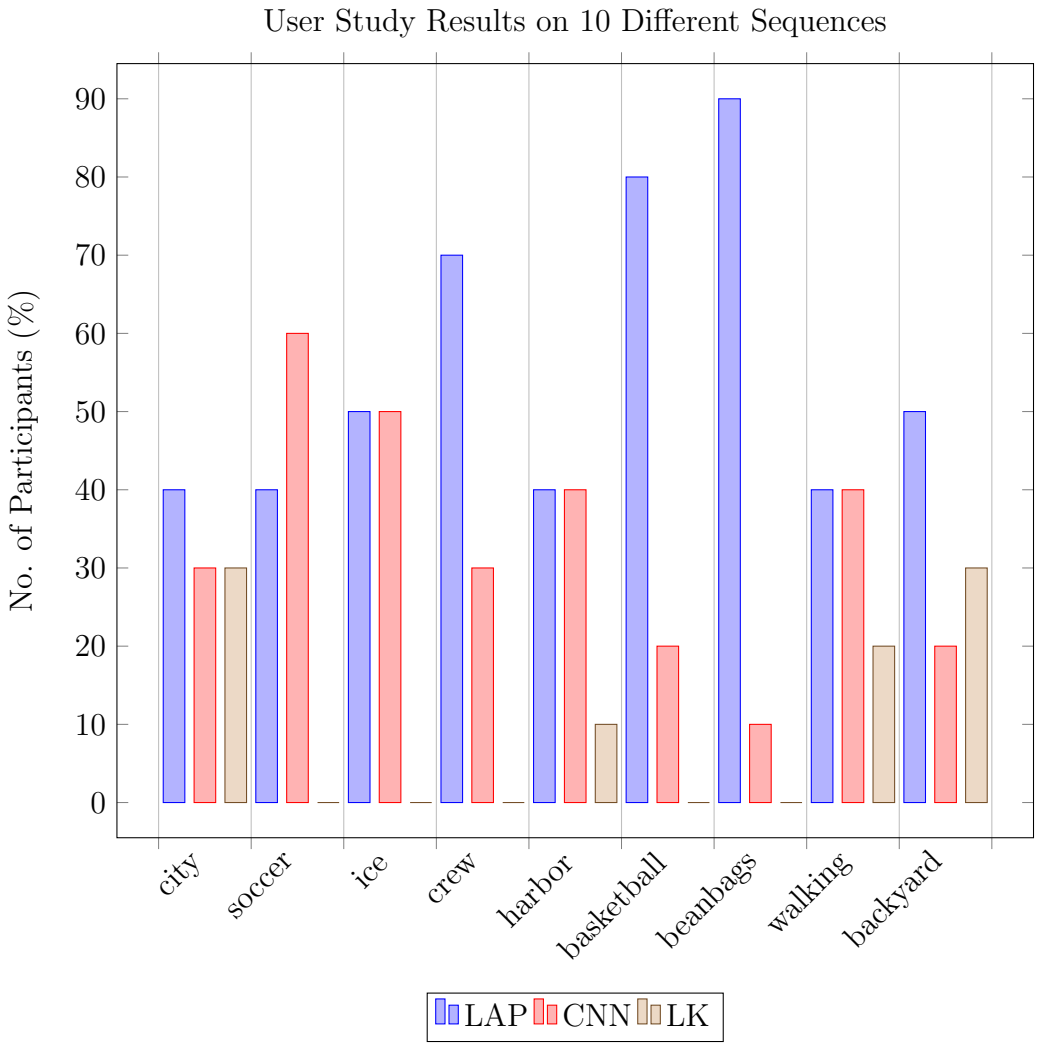


Figure 5.4: Results of the user study on 10 different sequences. The sequences reported here best illustrate the discrepancy between MSE and subjective visual quality.

the most visually pleasing and natural looking sequence.

Users were asked to view the various sequences three times in a dark room on a laptop screen. A larger screen was not chosen to prevent possible spatial aliasing. All users chosen were 18 years and older.

The user study results on some chosen sequences are shown in Figure 5.4. None of the viewers judged the LK interpolated crew, soccer or ice sequence as the best. They all noticed the visual artifacts in the sequences and many of them found it very disturbing. Most of the users felt that the LAP in-

terpolated crew sequence blended the flashes of light much better into the overall sequence in comparison to the CNN interpolated sequence. On the other hand, in the soccer sequence more users felt the players leg movement was much crisper when viewing the sequence multiple times. Notice that despite LAP having higher MSE than CNN for the ice and soccer sequences, LAP had much better visual quality according to the user study, and in some cases even better than the CNN.

The users also found the distortion in the balls in the CNN interpolated bean-bags and basketball sequence very unnatural but not as distracting as the large deforming warping introduced in the LK sequences. Users thought that all three sequences were equally good for the city and backyard sequences.

After the experiment, the users were given the opportunity to share the features or criteria they used to judge the quality of the sequences. Many users paid close attention to small objects like balls, while others compared the quality between the different sequences and judged based on smoothness and artifacts. Most users said that viewing the sequence three times allowed them to focus on different patterns and make a better decision.

CHAPTER 6

CONCLUSION

In this thesis we have shown that the LAP optical flow method is an excellent candidate for video interpolation. The method has quadratic approximation order, making it exceptionally accurate when the true displacement field is smooth. The method is also very fast, compares very favorably with a recent CNN method, and generally outperforms a pyramid version of Lucas-Kanade.

We also notice that the MSE does not correlate well with visual quality in most experiments. Even slight displacements between a ground truth frame and an interpolated frame can result in high MSE despite very pleasing visual quality. Our hypothesis about this discrepancy is proven after performing a user study. The results of the study further indicate that the LAP algorithms performs natural and smooth interpolation.

The LAP method is also compared against Deepflow2 and MDP-Flow2, two leaders on the Middlebury optical flow benchmark. A case study performed on the basketball sequence from Derf's Media Collection shows that the LAP algorithm performs high quality interpolation and results in natural looking frames in comparison to these two methods.

We also experimented with different interpolation algorithms that employ not only the forward optical flow but also the backward flow. However, such methods are computationally expensive and provide poor results due to significant occlusions. Future work involves estimating occlusions using the LAP flow vectors and incorporating these accurate occlusion masks to perform even higher quality frame interpolation.

REFERENCES

[1] R. Krishnamurthy, J. W. Woods, and P. Moulin, "Frame interpolation and bidirectional prediction of video using compactly encoded optical-flow fields and label fields," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 9, no. 5, pp. 713–726, Aug 1999.

[2] Y. Chen, I. V. Bajic, and C. Qian, "Frame rate up-conversion of compressed video using region segmentation and depth ordering," in *2009 IEEE Pacific Rim Conference on Communications, Computers and Signal Processing*, Aug 2009, pp. 431–436.

[3] S. Sekiguchi, Y. Idehara, K. Sugimoto, and K. Asai, "A low-cost video frame-rate up conversion using compressed-domain information," in *IEEE International Conference on Image Processing 2005*, vol. 2, Sep. 2005, pp. II–974.

[4] A. Daneshi, H. Behnam, and Z. Alizadeh Sani, "Frame rate up-conversion in echocardiography images, using manifold-learning and image registration," *bioRxiv*, 2018. [Online]. Available: https://www.biorxiv.org/content/early/2018/09/03/407072

[5] C. Wang, L. Zhang, Y. He, and Y. Tan, "Frame rate up-conversion using trilateral filtering," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 20, no. 6, pp. 886–893, June 2010.

[6] Z. Wang and A. C. Bovik, "Mean squared error: Love it or leave it? a new look at signal fidelity measures," *IEEE Signal Processing Magazine*, vol. 26, no. 1, pp. 98–117, Jan 2009.

[7] K. Rijkse, "H.263: video coding for low-bit-rate communication," *IEEE Communications Magazine*, vol. 34, no. 12, pp. 42–45, Dec 1996.

[8] T. Wiegand, "Final draft international standard for joint video specification H (ITU-T Rec. H.264 —ISO/IEC 14496-10AVC, ITU-T Rec.H.264 — ISO/IEC 14496-10AVC)," Jan 2003.

[9] A. Barjatya, "Block matching algorithms for motion estimation," *IEEE Transactions Evolution Computation*, vol. 8, pp. 225–239, Jan 2004.

[10] W. Hassen and H. Amiri, "Block matching algorithms for motion estimation," in *2013 7th IEEE International Conference on e-Learning in Industrial Electronics (ICELIE)*, Nov 2013, pp. 136–139.

[11] R. Li, B. Zeng, and M. L. Liou, "A new three-step search algorithm for block motion estimation," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 4, no. 4, pp. 438–442, Aug 1994.

[12] L. Po and W. Ma, "A novel four-step search algorithm for fast block motion estimation," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 6, no. 3, pp. 313–317, June 1996.

[13] S. Zhu and K. Ma, "A new diamond search algorithm for fast block-matching motion estimation," *IEEE Transactions on Image Processing*, vol. 9, no. 2, pp. 287–290, Feb 2000.

[14] Y. Nie and K. Ma, "Adaptive rood pattern search for fast block-matching motion estimation," *IEEE Transactions on Image Processing*, vol. 11, no. 12, pp. 1442–1449, Dec 2002.

[15] B. D. Lucas and T. Kanade, "An iterative image registration technique with an application to stereo vision," in *IJCAI*, 1981.

[16] P. Dollár, "Piotr's Computer Vision Matlab Toolbox (PMT)," https://github.com/pdollar/toolbox.

[17] B. K. Horn and B. G. Schunck, "Determining optical flow," MIT, Cambridge, MA, USA, Tech. Rep., 1980.

[18] R. C. oReilly and J. M. Beck, "A family of large-stencil discrete laplacian approximations in three dimensions," 2006.

[19] M. J. Black and P. Anandan, "Robust dynamic motion estimation over time," in *Proceedings. 1991 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, June 1991, pp. 296–302.

[20] T. Brox, A. Bruhn, N. Papenberg, and J. Weickert, "High accuracy optical flow estimation based on a theory for warping," in *ECCV*, 2004.

[21] T. Brox, C. Bregler, and J. Malik, "Large displacement optical flow," in *2009 IEEE Conference on Computer Vision and Pattern Recognition*, June 2009, pp. 41–48.

[22] S. Baker, D. Scharstein, J. P. Lewis, S. Roth, M. J. Black, and R. Szeliski, "A database and evaluation methodology for optical flow," *2007 IEEE 11th International Conference on Computer Vision*, pp. 1–8, 2007.

[23] L. Xu, J. Jia, and Y. Matsushita, "Motion detail preserving optical flow estimation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 34, no. 9, pp. 1744–1757, Sep. 2012.

[24] P. Weinzaepfel, J. Revaud, Z. Harchaoui, and C. Schmid, "DeepFlow: Large displacement optical flow with deep matching," in *IEEE Intenational Conference on Computer Vision (ICCV)*, Sydney, Australia, Dec. 2013. [Online]. Available: http://hal.inria.fr/hal-00873592

[25] S. C. Park, M. K. Park, and M. G. Kang, "Super-resolution image reconstruction: a technical overview," *IEEE Signal Processing Magazine*, vol. 20, pp. 21–36, 2003.

[26] P. Thevenaz, T. Blu, and M. Unser, "Interpolation revisited [medical images application]," *IEEE Transactions on Medical Imaging*, vol. 19, no. 7, pp. 739–758, July 2000.

[27] M. Vetterli, J. Kovaevi, and V. K. Goyal, *Foundations of Signal Processing*. Cambridge University Press, 2014.

[28] Cmglee, https://commons.wikimedia.org/w/index.php?curid=53064904.

[29] T. Blu, P. Thcvenaz, and M. Unser, "MOMS: maximal-order interpolation of minimal support," *IEEE Transactions on Image Processing*, vol. 10, no. 7, pp. 1069–1080, July 2001.

[30] S. Niklaus, L. Mai, and F. Liu, "Video frame interpolation via adaptive separable convolution," *2017 IEEE International Conference on Computer Vision (ICCV)*, pp. 261–270, 2017.

[31] C. Gilliam and T. Blu, "Local all-pass filters for optical flow estimation," in *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, April 2015, pp. 1533–1537.

[32] C. Gilliam, T. Kstner, and T. Blu, "3d motion flow estimation using local all-pass filters," in *2016 IEEE 13th International Symposium on Biomedical Imaging (ISBI)*, April 2016, pp. 282–285.

[33] T. Blu, P. Moulin, and C. Gilliam, "Approximation order of the lap optical flow algorithm," in *2015 IEEE International Conference on Image Processing (ICIP)*, Sep. 2015, pp. 48–52.

[34] C. Gilliam and T. Blu, "Local all-pass geometric deformations," *IEEE Transactions on Image Processing*, vol. 27, no. 2, pp. 1010–1025, Feb 2018.

[35] "Derf's media collection," https://media.xiph.org/video/derf/.

[36] D. Damen, H. Doughty, G. Farinella, S. Fidler, A. Furnari, E. Kazakos, D. Moltisanti, J. Munro, T. Perrett, W. Price, and M. Wray, "Scaling egocentric vision: The epic-kitchens dataset," in *European Conference on Computer Vision (ECCV)*, 2018.

[37] H. Men, H. Lin, V. Hosu, D. Maurer, A. Bruhn, and D. Saupe, "Technical report on visual quality assessment for frame interpolation," Jan 2019.

[38] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli, "Image quality assessment: from error visibility to structural similarity," *IEEE Transactions on Image Processing*, vol. 13, no. 4, pp. 600–612, April 2004.

[39] M. P. Sampat, Z. Wang, S. Gupta, A. C. Bovik, and M. K. Markey, "Complex wavelet structural similarity: A new image similarity index," *IEEE Transactions on Image Processing*, vol. 18, no. 11, pp. 2385–2401, Nov 2009.