SUPER RESOLUTION ON LINUX TELEVISIONS

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

AHMET EGE MAHLEÇ

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
ELECTRICAL AND ELECTRONICS ENGINEERING

FEBRUARY 2016

Approval of the thesis:

SUPER RESOLUTION ON LINUX TELEVISIONS

submitted by **AHMET EGE MAHLEÇ** in partial fulfillment of the requirements for the degree of **Master of Science in Electrical and Electronics Engineering Department, Middle East Technical University** by,

Prof. Dr. Gülbin Dural Ünver _____
Dean, Graduate School of **Natural and Applied Sciences**

Prof. Dr. Gönül Turhan Sayan _____
Head of Department, **Electrical and Electronics Engineering**

Prof. Dr. Gözde Bozdağı Akar _____
Supervisor, **Electrical and Electronics Eng. Dept., METU**

Examining Committee Members:

Prof. Dr. Aydın Alatan
Electrical and Electronics Engineering Dept., METU _____

Prof. Dr. Gözde Bozdağı Akar
Electrical and Electronics Engineering Dept., METU _____

Prof. Dr. Kemal Leblebicioğlu
Electrical and Electronics Engineering Dept., METU _____

Prof. Dr. Ziya Telatar
Electrical and Electronics Engineering Dept., AU _____

Assoc. Prof. Dr. Ahmet Oğuz Akyüz
Computer Engineering Dept., METU _____

Date: _____

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last Name: AHMET EGE MAHLEÇ

Signature :

ABSTRACT

SUPER RESOLUTION ON LINUX TELEVISIONS

Mahleç, Ahmet Ege
M.S., Department of Electrical and Electronics Engineering
Supervisor : Prof. Dr. Gözde Bozdağı Akar

February 2016, 116 pages

Demand of obtaining better quality image is becoming very important topic in television market. Since many content providers still broadcast videos in SD or HD resolution, upscaling input video to display it on FHD or UHD television with better quality is the key point of image quality. Because basic interpolation algorithms do not satisfy this demand, super resolution algorithms are needed to obtain better quality output video. Therefore, many chip providers claim that their chip supports super resolution algorithms.

Throughout this thesis, single-frame and multi-frame super resolution algorithms' performance and complexity will be investigated to find an optimal algorithm which is able to run on Linux televisions in real-time. Moreover, effect of the motion estimation on multi-frame super resolution algorithms will be investigated.

As a second contribution of this thesis, mmrgLibrary will be presented. This image processing library is developed in order to unify all the third party libraries into one library and in order to implement customize algorithms. This library is cross-platform API which can be executed on ARM platforms, X86 platforms or any platforms if the toolchain of this platform is provided.

Keywords: Super Resolution, Image/Video Enhancement, Motion Estimation

ÖZ

LİNUX TELEVİZYONLARDA SÜPER ÇÖZÜNÜRLÜK

Mahleç, Ahmet Ege
Yüksek Lisans, Elektrik ve Elektronik Mühendisliği Bölümü
Tez Yöneticisi : Prof. Dr. Gözde Bozdağı Akar

Şubat 2016 , 116 sayfa

Televizyon pazarında daha kaliteli resim elde etmek çok önemli bir konudur. Birçok içerik sağlayıcı halen daha SD veya HD çözünürlükte içerik dağıttığı için; girdi videonun yeniden boyutlandırılarak FHD veya UHD ekranlarda daha iyi gösterilebilmesi resim kalitesi açısından anahtar noktalardan birisidir. Basit ara değerleme algoritmaları bu ihtiyacı karşılayamadıkları için, süper çözünürlük algoritmaları daha iyi kalite çıktı video oluşturmak için gereklidir. Bundan dolayı, birçok çip sağlayıcı kendi çiplerinin süper çözünürlük algorithmalarını desteklediklerini iddia etmektedirler.

Bu tez boyunca, Linux televizyonlarda gerçek zamanlı çalışan en uygun algoritmayı bulmak için tek-resim kullanan ve çok-resim kullanan süper çözünürlük algoritmaları performans ve işlem yükü olarak karşılaştırılacaktır. Bunun yanı sıra, hareket kestirimi algoritmalarının çok-resim kullanan süper çözünürlük üzerindeki etkileri araştırılacaktır.

Bu tez kapsamında bir yenilik olarak mmrgLibrary sunulacaktır. Bu görüntü işleme kütüphanesi bütün yardımcı kütüphaneleri birleştirmek ve ihtiyaca göre düzenlenmiş algoritmaları uygulamak için geliştirilmiştir. Bu kütüphane ARM, X86 ve araç zinciri sağlanmış her platformda çalışabilir.

Anahtar Kelimeler: Süper Çözünürlük, Resim/Video İyileştirme, Hareket Kestirimi

To my family

Ayten Mahleç, Mustafa Mahleç, Ece Mahleç Yılmaz and Hakan Yılmaz

To her

Esra Özcan

ACKNOWLEDGMENTS

I would like to thank to my advisor, Prof Dr Gözde Bozdağı Akar, for her guidance, support and understanding throughout my graduate program.

I would like to express my gratitude to my company, Arçelik Inc, so that they let me know Linux Television environment. I would remiss if I didn't mention my colleagues, Sezer Bağlan and Özgür Çelebican. I would like to thank Özgür Çelebican for his understanding and I would like to thank Sezer Bağlan for his support on mmrgLibrary.

Since this thesis is the output of SANTEZ program, I would like to express my deep appreciation to Osman Solakoğlu and Mustafa Uğuz for their support.

Finally, I would like to thank to my family; Ayten Mahleç, Mustafa Mahleç, Ece Mahleç Yılmaz, Hakan Yılmaz and Esra Özcan for their support throughout this thesis. Everytime I gave up, they made me motivated again.

TABLE OF CONTENTS

ABSTRACT v

ÖZ vi

ACKNOWLEDGMENTS viii

TABLE OF CONTENTS ix

LIST OF TABLES xiii

LIST OF FIGURES xiv

LIST OF ABBREVIATIONS xix

CHAPTERS

1 INTRODUCTION 1

1.1 Introduction 1

1.2 Scope of Thesis 3

1.3 Outline of Thesis 3

2 BACKGROUND METHODS 5

2.1 Image Quality Metrics 6

2.1.1 Peak Signal to Noise Ratio (PSNR) 6

2.1.2 Differential Mean Opinion Score (DMOS) 7

2.1.3 Attention Weighted Differential Mean Opinion Score (ADMOS) 10

2.2 Motion Estimation Algorithms 11

2.2.1 Projection-Based Motion Estimation 12

2.2.2 Block Matching Exhaustive Search Motion Estimation 13

2.3 Bilateral Filter 17

2.3.1 Adaptive Bilateral Filter 21

2.3.2 Adaptive Bilateral Filter Considering Local Characteristics 21

3 LITERATURE SURVEY 25

3.1 Basic Interpolation Algorithms 25

3.2 Single Frame Super Resolution 28

3.2.1 Fast Edge-Adaptive Interpolation 28

3.2.2 Iterative Back-Projection Approach 29

3.3 Multi-Frame Super Resolution 30

3.3.1 Interpolation-Restoration Type Method 31

3.3.2 Model Based Super Resolution 32

3.3.3 Bayesian Methods 34

3.3.3.1 Maximum Likelihood Method 35

3.3.4 Super Resolution With Probabilistic Motion Estimation 36

3.3.5 Fast Video Interpolation/Up-Sampling Using Linear Motion Model 38

4 RESULT OF ANALYZED ALGORITHMS 41

4.1 Performance Results of Super Resolution Algorithms 42

4.1.1 Fast Edge-Adaptive Interpolation 43

4.1.2 Iterative-Back Projection Algorithm 45

4.1.3 Maximum-Likelihood Algorithm 47

4.1.3.1 Maximum Likelihood with Projection-Based Motion Estimation 47

4.1.3.2 Maximum Likelihood with Block Matching Full Search Motion Estimation . . 50

4.1.4 Super Resolution with Probabilistic Motion Estimation Algorithm 52

4.1.5 Fast Video Interpolation/Up-Sampling Using Linear Motion Model 54

4.1.5.1 Fast Video Interpolation without Motion Estimation 54

4.1.5.2 Fast Video Interpolation with Projection Based Motion Estimation 56

4.1.5.3 Fast Video Interpolation with Block Matching Full Search Motion Estimation 58

4.2 Comparison of Super Resolution Algorithms 60

4.3 Discussion 63

5 PARALLEL PROGRAMMING 65

5.1 Introduction to OPENCL 66

5.2 mmrgLibrary OPENCL Support 68

6 CONCLUSION AND FUTURE WORKS 71

6.1 Conclusion 71

6.2 Future Works 72

REFERENCES 73

APPENDICES

A ALGORITHMS 75

B INTRODUCTION TO MMRGLIBRARY 83

B.1 Structure of mmrgLibrary 84

B.2 Class Diagram of mmrgLibrary 86

B.3 mmrgResize Example 91

C VISUAL RESULTS 93

D CPU DETAILS 111

LIST OF TABLES

TABLES

Table 2.1 Mean Opinion Scores 7

Table 2.2 Differential Mean Opinion Scores 8

Table 2.3 Execution time of motion estimation algorithms for different resolutions on INTEL architecture whose details are listed in Appendix D . . . 16

Table 2.4 Execution time of motion estimation algorithms for different resolutions on ARM architecture whose details are listed in Appendix D 16

Table 4.1 Comparison Table of Super Resolution Algorithms for Flower Video 60

Table 4.2 Comparison Table of Super Resolution Algorithms for Foreman Video 61

Table 4.3 Comparison Table of Super Resolution Algorithms for News Video . 61

Table 4.4 Comparison Table of Super Resolution Algorithms for Stefan Video 61

Table 4.5 Execution time of super resolution algorithms for different resolutions on INTEL architecture whose details are listed in Appendix D 62

Table 4.6 Execution time of super resolution algorithms for different resolutions on ARM architecture whose details are listed in Appendix D 62

Table 5.1 Execution time of Serial and Parallel Fast Edge-Adaptive Interpolation for different resolutions on INTEL architecture whose details are listed in Appendix D 69

LIST OF FIGURES

FIGURES

Figure 2.1 Illustration of the effect of the noise on Low Frequency Image and High Frequency Image 9

Figure 2.2 Illustration of the effect of the temporal frequency on DMOS . . . 9

Figure 2.3 Illustration of the effect of the contrast on DMOS metric 10

Figure 2.4 Correlation of Subjective DMOS resuls with PSNR, DMOS and ADMOS results 11

Figure 2.5 The illustration of working principle of Projection-based Motion Estimation 12

Figure 2.6 PSNR Comparison of Block Matching Full Search and Projection-Based Motion Estimation 14

Figure 2.7 DMOS Comparison of Block Matching Full Search and Projection-Based Motion Estimation 15

Figure 2.8 ADMOS Comparison of Block Matching Full Search and Projection-Based Motion Estimation 15

Figure 2.9 Illustration of the effect of the range and distance filters for bilateral filter. 17

Figure 2.10 Gaussian noise is added to original Lena image by using different variance values 18

Figure 2.11 Gaussian noise is added to original MMRG image by using different variance values 19

Figure 2.12 Examined regions for MMRG Image and Lena Image 19

Figure 2.13 PSNR results of the bilateral filter response to noisy images with different noise levels.BF(50) means that variance of the range filter is 50. ABF is the abbreviation of the Adaptive Bilateral Filter, whereas ABF CLC is the abbreviation of the "Adaptive Bilateral Filter Considering Local Characteristics" 20

Figure 2.14 The illustration of the working principle for Adaptive bilateral filter for different regions 22

Figure 3.1 Fast Edge-Adaptive Interpolation Algorithm [9] 29

Figure 3.2 Illustration of Extended Motion Trajectory 38

Figure 4.1 Picture Quality Analyzer User Interface 41

Figure 4.2 PSNR Comparison of Fast Edge-Adaptive Interpolation and Bicubic Interpolation 44

Figure 4.3 DMOS Comparison of Fast Edge-Adaptive Interpolation and Bicubic Interpolation 44

Figure 4.4 ADMOS Comparison of Fast Edge-Adaptive Interpolation and Bicubic Interpolation 45

Figure 4.5 PSNR Comparison of Iterative Back Projection and Bicubic Interpolation 46

Figure 4.6 DMOS Comparison of Iterative Back Projection and Bicubic Interpolation 46

Figure 4.7 ADMOS Comparison of Iterative Back Projection and Bicubic Interpolation 47

Figure 4.8 PSNR Comparison of Maximum Likelihood with Projection-Based Motion Estimation and Bicubic Interpolation 48

Figure 4.9 DMOS Comparison of Maximum Likelihood with Projection-Based Motion Estimation and Bicubic Interpolation 48

Figure 4.10 ADMOS Comparison of Maximum Likelihood with Projection-Based Motion Estimation and Bicubic Interpolation 49

Figure 4.11 PSNR Comparison of Maximum Likelihood with Block Matching Full Search and Bicubic Interpolation 50

Figure 4.12 DMOS Comparison of Maximum Likelihood with Block Matching Full Search and Bicubic Interpolation 51

Figure 4.13 ADMOS Comparison of Maximum Likelihood with Block Matching Full Search and Bicubic Interpolation 51

Figure 4.14 PSNR Comparison of Super Resolution with Probabilistic Motion Estimation and Bicubic Interpolation 52

Figure 4.15 DMOS Comparison of Super Resolution with Probabilistic Motion Estimation and Bicubic Interpolation 53

Figure 4.16 ADMOS Comparison of Super Resolution with Probabilistic Motion Estimation and Bicubic Interpolation 53

Figure 4.17 PSNR Comparison of Fast Video Interpolation without Motion Estimation and Bicubic Interpolation 54

Figure 4.18 DMOS Comparison of Fast Video Interpolation without Motion Estimation and Bicubic Interpolation 55

Figure 4.19 ADMOS Comparison of Fast Video Interpolation without Motion Estimation and Bicubic Interpolation 55

Figure 4.20 PSNR Comparison of Fast Video Interpolation with Projection-Based Motion Estimation and Bicubic Interpolation 57

Figure 4.21 DMOS Comparison of Fast Video Interpolation with Projection-Based Motion Estimation and Bicubic Interpolation 57

Figure 4.22 ADMOS Comparison of Fast Video Interpolation with Projection-Based Motion Estimation and Bicubic Interpolation 58

Figure 4.23 PSNR Comparison of Fast Video Interpolation with Block Matching Full Search and Bicubic Interpolation 59

Figure 4.24 DMOS Comparison of Fast Video Interpolation with Block Matching Full Search and Bicubic Interpolation 59

Figure 4.25 ADMOS Comparison of Fast Video Interpolation with Block Matching Full Search and Bicubic Interpolation 60

Figure 5.1 OPENCL Platform Model [16] 66

Figure 5.2 OPENCL Memory Model [16] 67

Figure 5.3 Simple Example of Scalar Versus Parallel Implementation [16] . . 68

Figure 5.4 mmrgLibrary Resize Image Example by using OPENCL implementation of Fast Edge-Adaptive Interpolation 69

Figure B.1 View of mmrgLibrary Source and Build Directory 85

Figure B.2 mmrgLibrary CMake Configuration File 87

Figure B.3 mmrgLibrary CMake Configuration File 88

Figure B.4 mmrgLibrary Class Diagram of Resize module 90

Figure B.5 mmrgLibrary Resize Image Example by using Bicubic Interpolation 91

Figure B.6 mmrgLibrary Resize Image Example by using Fast Edge-Adaptive Interpolation 91

Figure B.7 mmrgLibrary Resize Raw Video Example by using Fast Video Interpolation 92

Figure C.1 Flower Video 200th HR Frame with three bi-directional LR Frames 93

Figure C.2 Foreman Video 155th HR Frame with three bi-directional LR Frames 94

Figure C.3 News Video 91st HR Frame with three bi-directional LR Frames . . 94

Figure C.4 Stefan Video 22nd HR Frame with three bi-directional LR Frames . 94

Figure C.5 Flower Video 200th Upscaled HR Frame 95

Figure C.6 PSNR Difference Map for Flower Video 200th Upscaled HR Frame 96

Figure C.7 DMOS Difference Map for Flower Video 200th Upscaled HR Frame 97

Figure C.8 ADMOS Difference Map for Flower Video 200th Upscaled HR Frame 98

Figure C.9 Foreman Video 155th Upscaled HR Frame 99

Figure C.10 PSNR Difference Map for Foreman Video 155th Upscaled HR Frame 100

Figure C.11 DMOS Difference Map for Foreman Video 155th Upscaled HR Frame 101

Figure C.12 ADMOS Difference Map for Foreman Video 155th Upscaled HR Frame 102

Figure C.13 Foreman Video 91st Upscaled HR Frame 103

Figure C.14 PSNR Difference Map for News Video 91st Upscaled HR Frame . 104

Figure C.15 DMOS Difference Map for News Video 91st Upscaled HR Frame . 105

Figure C.16 ADMOS Difference Map for News Video 91st Upscaled HR Frame 106

Figure C.17 Stefan Video 22nd Upscaled HR Frame 107
Figure C.18 PSNR Difference Map for Stefan Video 22nd Upscaled HR Frame . 108
Figure C.19 DMOS Difference Map for Stefan Video 22nd Upscaled HR Frame 109
Figure C.20 ADMOS Difference Map for Stefan Video 22nd Upscaled HR Frame 110

LIST OF ABBREVIATIONS

QCIF	Quarter Common Intermediate Format (176x144)
CIF	Common Intermediate Format (352x288)
SD	Standard Definition (576x704)
HD	High Definition (1280x720)
FHD	High Definition (1920x1080)
UHD	Ultra High Definition (3840x2160)
PBME	Projection-Based Motion Estimation
BMFS	Block Matching Full Search Motion Estimation
PME	Probabilistic Motion Estimation
ITU	International Telecommunication Union
HDMI	High-Definition Multimedia Interface
HEVC	High Efficiency Video Codec
PSNR	Peak Signal to Noise Ratio
SSIM	Structural Similarity Index Measurement
DMOS	Differential Mean Opinion Score
ADMOS	Attention Weighted Differential Mean Opinion Score
PQASW	Picture Quality Analyzer Software
CPU	Central Processing Unit
GPU	Graphic Processing Unit
FPGA	Field Programmable Gate Array
ME	Motion Estimation
BF	Bilateral Filter
ABF	Adaptive Bilateral Filter
ABF_CLC	Adaptive Bilateral Filter Considering Local Characteristic
HR	High Resolution
LR	Low Resolution
SR	Super Resolution

CHAPTER 1

INTRODUCTION

1.1 Introduction

With the evolution of the higher resolution displays, obtaining high resolution images is must for consumer electronics. With the advent of the UHD resolution, many standards are implemented in order to display UHD resolution which is defined by ITU. Especially by the aid of High Efficiency Video Codec Standard, HDMI 2.0 Standard and VP9 Standard; the use of 4K resolution are increasing day by day. Since formerly recorded movies are low resolution and also some television broadcasters are still broadcasting in low resolution due to limit of transmission bandwidth, the low resolution frames must be upscaled to be fitted to screen. This upscaling process mostly handled by using basic interpolation algorithms ,due to their low computational load, which may cause too much artifacts depend on the feature of input images. If this case happens, the perceptual picture quality will decrease remarkably. Since quantity of the sales are dependent of picture quality for the devices which use larger displays, such as televisions and computers; obtaining high resolution image with high quality is a remarkable topic which is highly correlated with the demand of the industry. In order to satisfy this demand; nowadays, many chip providers which produce solutions for television industry use super resolution algorithms in their chips to obtain high resolution image with high quality.

Super resolution is the general name of the techniques to increase spatial or optical resolution of input image while retrieving the lost high frequency components, also called as "details". These techniques use low resolution image or a set of low res-

olution images as input to reconstruct a high resolution image as output. If a super resolution approach uses single low resolution image to reconstruct high resolution image, this super resolution approach is classified as single-frame super resolution approach. On the other hand, if a super resolution approach uses multiple low resolution images to reconstruct high resolution image, this super resolution approach is classified as multi-frame super resolution approach.

As it is mentioned, the straightforward way to upscale an image is applying basic interpolation algorithms. Well-known examples of these algorithms are Nearest-Neighbor interpolation, Bilinear interpolation and Bicubic interpolation. These algorithms are suitable for real time implementation because of their simplicity.

Single-frame super resolution algorithms need more computational power when compared with basic interpolation algorithms. However, they are formed an output image which has higher quality when compared with basic image interpolation algorithms. Among all the algorithms in the literature; Iterative-Back Projection algorithm and Edge-Adaptive Interpolation algorithm will be examined in this thesis.

Multi-frame super resolution algorithms use a set of low resolution images to form high resolution output image. In order to obtain high resolution image, these algorithms estimate motion information between consecutive frames. By compensating motion between consecutive frames, they are fusing several low resolution images into one high resolution image.

The output image quality of classical multi-frame super resolution algorithms highly depend on accuracy of motion estimation algorithms. In the classical multi-frame super resolution algorithms, the aligned frames are fused into output high resolution image regardless of the correlation of the input images. Since general content movies have complex motion, it is hard to estimate the motion between consecutive frames accurately.

Recently, a few video interpolation algorithms are proposed to obtain a better result for general content movies. These video interpolation algorithms are the special case of the multi-frame super resolution algorithms. In the formulation process, they ignore the blurring and adding noise operations and in addition to that they offer

probabilistic motion estimation to prevent the artifacts caused by motion estimation algorithms. However, these algorithms generates blur output image, since the algorithms are very similar with the bilateral filter implementation.

1.2 Scope of Thesis

In this thesis, super resolution algorithms will be compared with respect to their performance and complexity. The aim of this thesis is to find a suitable algorithm which can be used in real time applications running in linux platforms. All the algorithms will be implemented on computer which has Intel i7 processor and on the Arçelik television which has Arm processor whose details can be found in Appendix D.

1.3 Outline of Thesis

In chapter 2, motion estimation, picture quality assessment and bilateral filter algorithms will be explained.

In chapter 3, literature survey of super resolution algorithm is going to be given.

In chapter 4, super resolution algorithms' performance-wise and complexity-wise results will be presented.

In chapter 5, OPENCL API will be introduced and performance of OPENCL support of mmrgLibrary whose details can be found in Appendix B will be explained.

CHAPTER 2

BACKGROUND METHODS

In this chapter, auxiliary algorithms will be explained. These algorithms are used in various parts of the super resolution algorithms. This chapter will be discussed in three parts.

First part is dedicated to explain the quality metrics; namely Peak Signal to Noise Ratio (PSNR), Differential Mean Opinion Score (DMOS) and Attention Weighted Differential Mean Opinion Score (ADMOS).

Second part of this chapter is dedicated to explain motion estimation algorithms which will be used in the registration part of multi-frame super resolution methods. In the registration part of multi-frame super resolution methods, the motion between consecutive frames are estimated and these motions are eliminated so that consecutive frames are aligned with reference frame. Therefore, motion estimation forms a set of aligned frames to be used in the reconstruction part of the multi-frame super resolution algorithms. In this thesis, Block Matching Exhaustive Search and Projection-based Motion Estimations will be investigated to be used in the registration part. These two algorithms are selected because the characteristic of these two algorithms are very different. While Projection-Based Motion Estimation is only capable of estimating global motion, Block Matching Full Search algorithm can estimate local motion. The other difference between these two algorithms is that while Block Matching Full Search algorithm has too much computational load, Projection-based Motion Estimation algorithm is as fast as possible to be applicable to real time implementation. Therefore, these two motion estimation algorithm will be used in the registration part of the multi-frame super resolution algorithm and the results will

be discussed.

In the third and final part of this chapter, bilateral filter denoising algorithm and its adaptive variants will be explained. This denoising technique is the origin of the modern super resolution algorithms which is discussed in this thesis. The detailed explanation and the results will be given in the third part of this chapter.

2.1 Image Quality Metrics

Throughout this thesis; PSNR, DMOS and ADMOS image quality metrics are used to measure the quality of processed images. These three methods are the members of full reference objective quality metrics. In these metrics, processed image and original image are compared to measure quality of processed image.

Although the working principles of these metrics are similar, the performance of these quality metrics to simulate human visual system is different. PSNR is the most primitive method among all the three metrics. This measurement is well known as having less correlation with subjective ratings. However, it is easy to implement and easy to adopt to real time systems. DMOS is perceptual-based measurements using human visual system models. It provides more accurate rating results, correlating more closely to human subjective tests than the PSNR measurement alone. DMOS measurement evaluates how much impairment viewers will perceive in test video content. ADMOS is the special type of DMOS measurement with the addition of a part of human cognition that accounts for what spectators are most likely to watch in any given scene.[2]

2.1.1 Peak Signal to Noise Ratio (PSNR)

PSNR is the most widely used quality metric to evaluate the super resolution algorithm in the literature. As it is indicated above, the calculation is very straight forward.

$$PSNR = 10 * log_{10}(\frac{(2^n - 1)}{\sqrt{MSE}}) \tag{2.1}$$

Table 2.1: Mean Opinion Scores

MOS	Quality	Impairment
5	Excellent	Imperceptible
4	Good	Perceptible but not annoying
3	Fair	Slightly annoying
2	Poor	Annoying
1	Bad	Very annoying

MSE is calculated as

$$MSE = \frac{1}{MN}\sum_{i=1}^{M}\sum_{j=1}^{N}(x(i,j) - y(i,j))^2 \tag{2.2}$$

Where x(i,j) represents the original image and y(i,j) represents the processed(modified) image. i and j are the pixel position of the MxN image and n is the channel bit depth.

As it is stated before, PSNR and MSE is the most primitive quality metrics which are not suitable for simulating human visual system as it is shown in figure 2.4a. For example, it is a well-known fact that humans are more sensitive to noise occurred in the low frequency area than the noise occurred in the high frequency area. However, if we measure the PSNR values of these two cases, it will give the same result for both cases. The other example can be counted as the perceptual difference of same noise on different luminance levels. Although the same noise, which is observed on regions which has different luminance values, is perceived differently; PSNR metric gives same result for both regions. Although PSNR has these disadvantages; it will be used to measure performance of the algorithms in this thesis, since it is widely used in the literature.

2.1.2 Differential Mean Opinion Score (DMOS)

Mean Opinion Score (MOS) is used to evaluate the video quality subjectively. The scores and their meanings of MOS can be found in table 2.1.

Differential Mean Opinion Score (DMOS) is calculated by finding the difference be-

Table 2.2: Differential Mean Opinion Scores

MOS	Quality	Impairment
1	Excellent	Imperceptible
2	Good	Perceptible but not annoying
3	Fair	Slightly annoying
4	Poor	Annoying
5	Bad	Very annoying

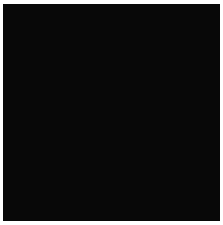
tween original and processed sequence's MOS. Less the difference between MOS score, better will be quality of processed video. Therefore, the scores and their meaning of DMOS can be found in table 2.2.

Although DMOS is a subjective quality metric, Tektronix Picture Quality Analyzer Software (PQASW) gives predicted DMOS scores by comparing the original video and processed video objectively. Since it is a commercial product, technical details of this algorithm cannot be found. However, advantage of this approach is shared by Tektronix.[2]

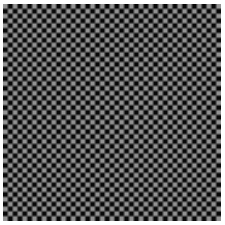
This predicted DMOS method takes into consideration different display types used to view the video (for example, interlaced or progressive and CRT or LCD) and different viewing conditions (for example, room lighting and viewing distance) and mean luminance, spatial frequency, temporal frequency of evaluated regions. It is a well known fact that PSNR does not take into consideration the spatial or temporal frequency and the characteristic of the input image. The examples of these cases can be found below.

Assuming that low frequency image and high frequency image are suffered from the same noise as it can be shown in the figure 2.1. Although PSNR results of these images are same, DMOS results state that noise on the low frequency image can be perceived higher than noise on the high frequency image as it can be seen in figure 2.1e and figure 2.1f.

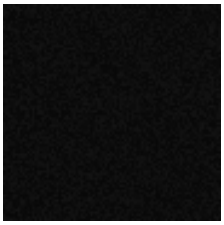
The other advantage of predicted DMOS result is that it considers the temporal frequency of input videos. Flicker video (luminance value of even frame's pixels are 128 whereas luminance value of odd frame's pixels are 0) is played in 10Hz and



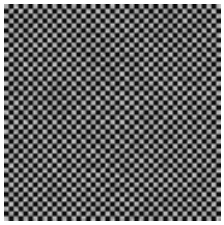
(a) Low Frequency Image



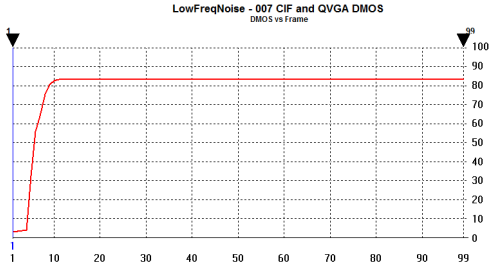
(b) High Frequency Image



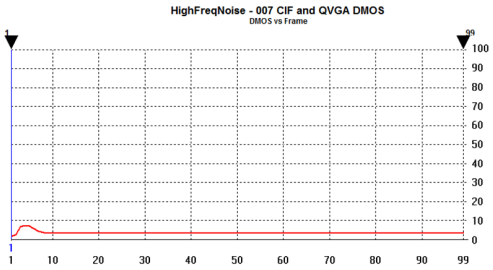
(c) Low Frequency Image with Noise



(d) High Frequency Image with Noise



(e) DMOS Graph of Low Frequency Image with Noise

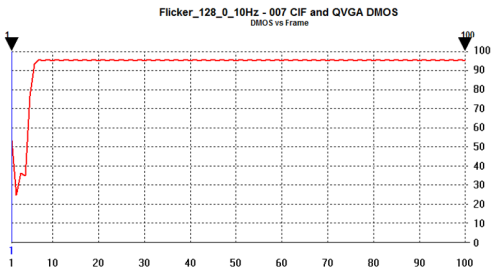


(f) DMOS Graph of High Frequency Image with Noise

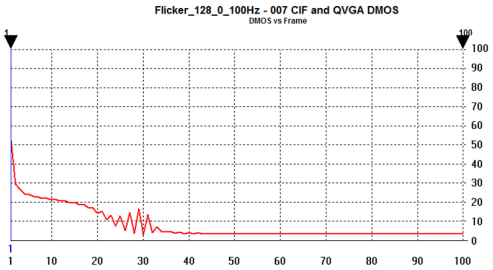
Figure 2.1: Illustration of the effect of the noise on Low Frequency Image and High Frequency Image

100Hz while video (luminance value of all frame's pixels are 128) is used as reference video. While PSNR gives same result for this case, DMOS gives consistent result with human visual system as it can be seen in figure 2.2. It is because of the fact that sensitivity of human eye decreases as long as the temporal frequency is increasing.

The last advantage of predicted DMOS result is that it considers characteristic of input images. Contrast and sharpening is the major factor of perceived picture quality.



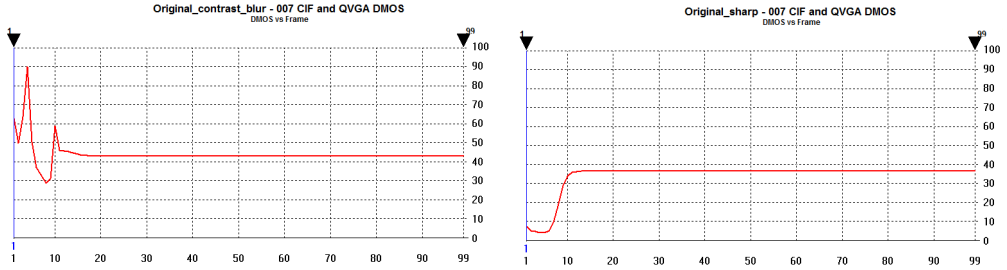
(a) DMOS Graph of Flicker Video played in 10Hz



(b) DMOS Graph of Flicker Video played in 100Hz

Figure 2.2: Illustration of the effect of the temporal frequency on DMOS

(a) Original Image (b) Low Contrast Image (c) High Contrast Image



(d) DMOS Graph of Low Contrast Image (e) DMOS Graph of High Contrast Image

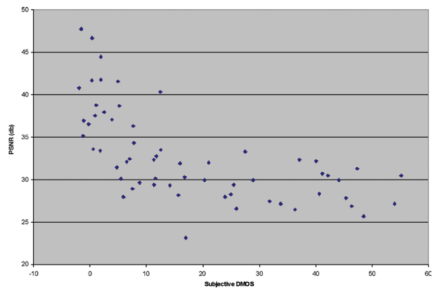
Figure 2.3: Illustration of the effect of the contrast on DMOS metric

Assuming that the original image, and two output image, which are shown in figures 2.3a, 2.3b and 2.3c respectively, will be compared to evaluate the algorithms performance. Although the PSNR gives same result for these two output images, predicted DMOS algorithm gives consistent result with human visual system. Therefore, DMOS shows that the second algorithm outgoes first algorithm as it can be seen in figure 2.3d and figure 2.3e.

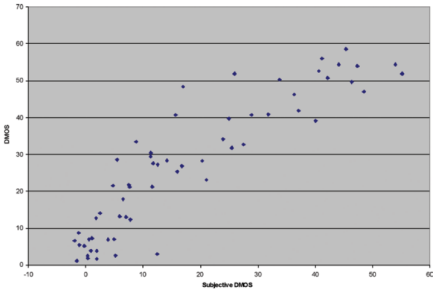
2.1.3 Attention Weighted Differential Mean Opinion Score (ADMOS)

As it is stated before, Attention Weighted DMOS measurement provides DMOS result with weighting apportioned by probable areas in the sequences on which the human eye is focusing. The weights of the features; namely motion, center, foreground, contrast, color, shape and size can be arranged in PQASW.

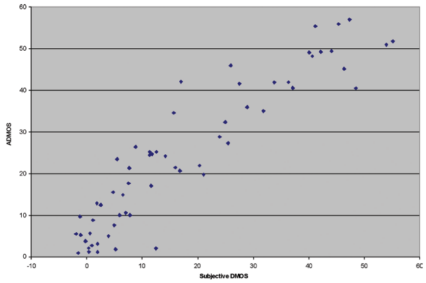
Communication Research Center in Canada conducted an experiment to measure the performance of PSNR, DMOS and ADMOS quality metrics.[1] The subjective scores had been obtained by contribution of the 33 participants. Every reference and processed image sequence is shown to participants to evaluate the reference and processed sequence by using MOS scores. These MOS scores had been used to obtain



(a) PSNR vs Subjective DMOS[1]



(b) DMOS vs Subjective DMOS[1]



(c) ADMOS vs Subjective DMOS[1]

Figure 2.4: Correlation of Subjective DMOS resuls with PSNR, DMOS and ADMOS results

DMOS scores. These subjective DMOS scores had been compared with the result of PSNR, DMOS and ADMOS quality metrics. As it can be seen in the figures 2.4a, 2.4b and 2.4c, the results shows that ADMOS has the best correlation with subjective DMOS results, while PSNR has the worst correlation.

2.2 Motion Estimation Algorithms

Motion between consecutive frames can be in any form. Form of the motion can be divided in two categories with respect to size of moving blocks. If every objects in the scene has the identical motion, this motion type is called as global motion. This motion type is the result of the camera movement. While the objects are stationary, if camera has horizontal and/or vertical motion, global motion is occurred in video. This motion type is easy to detect and can be compensated with the motion estimation algorithms.

The other motion type is local motion. While some objects are moving in the scene, if

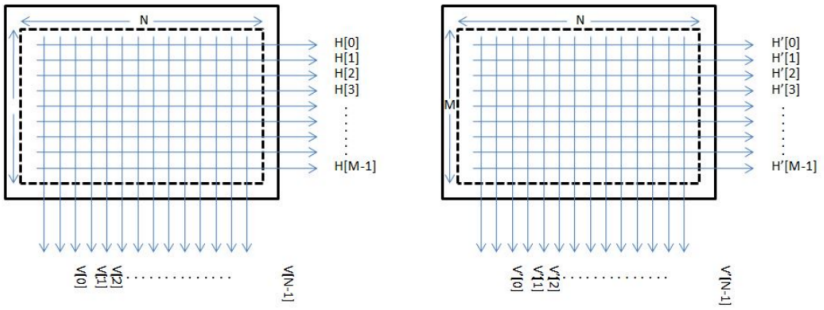


Figure 2.5: The illustration of working principle of Projection-based Motion Estimation

the others are stationary, this motion type is called as local motion. This motion type is hard to detect and needs more computational power when compared with global motion. The reason why local motion is hard to detect is that the moving objects can be in any shape and velocity. In addition to that, luminance change or noise may affect the performance of these algorithms.

Motion can also be divided in two categories with respect to the way blocks move. First one of them is the translational motion. If a moving block has vertical and/or horizontal motion, this motion is called as translational motion. Translational motion can be local motion or global motion.

The second of them is the rotational motion. If a moving block or a group of pixels has circular motion, this motion is called rotational motion. Rotational motion is very hard to detect and the motion estimation algorithms which are going to be explained in this thesis are not capable of detecting rotational motion.

In this chapter, Projection-Based Motion Estimation and Block Matching Exhaustive Search Motion Estimation will be explained.

2.2.1 Projection-Based Motion Estimation

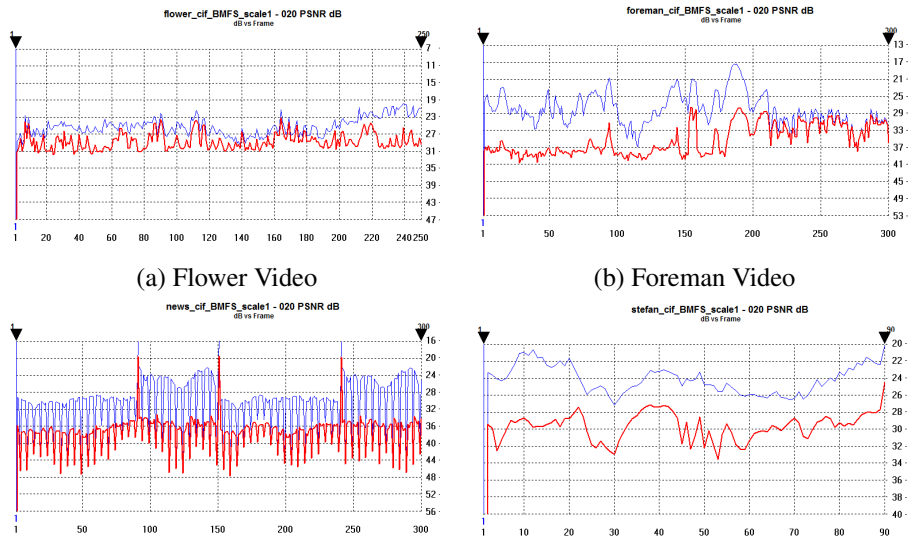
This method is only capable of detecting global translational motion. Local motion and any rotational motion cannot be detected by using this method. The working principle is very simple.

As it is shown in the figure 2.5, four arrays are created to store the sum of the values for horizontal lines and vertical lines. Two of them will be used for the reference image and others will be used for the image whose motion vectors are tried to be calculated. After sum of each lines is calculated and saved to the arrays, mean absolute error is calculated by sliding one array on another array. The motion vector which produces minimum mean absolute error will give us the global motion vector. This process is applied both for vertical direction and horizontal direction. As a result, translational vertical and horizontal global motion is estimated.[4]

The main advantage of this algorithm is the fast enough to be adoptable to real time implementation. The other advantage is that the output image does not suffer from the blockiness artifact. However, its algorithm is partially appropriate for parallel implementation. In other words, summing the vertical and the horizontal line part can be implemented by using parallel programming; however, finding motion vectors can hardly be implemented by using parallel programming. The other disadvantage of this motion estimation algorithm is that projection-based motion estimation is not capable of local translational motion or any rotational motion.

2.2.2 Block Matching Exhaustive Search Motion Estimation

This approach is the best approach to find the local motion. Working principle of block matching exhaustive search motion estimation is very straight forward. Simply, every block in the second frame is searched in the reference frame to find the best match. In other words, the algorithm tries to minimize the equation to find the optimum displacement vector d. The complexity depends on the size of the search window, image width and height.[3] This approach is very appropriate for parallel programming. However, it is not capable of finding rotational motion. If the block size is selected small enough, rotational motion can be detected partially. Although this approach is suitable for parallel programming; since it needs too much computational power, it is not appropriate for real time implementation unless a dedicated



(a) Flower Video (b) Foreman Video

(c) News Video (d) Stefan Video

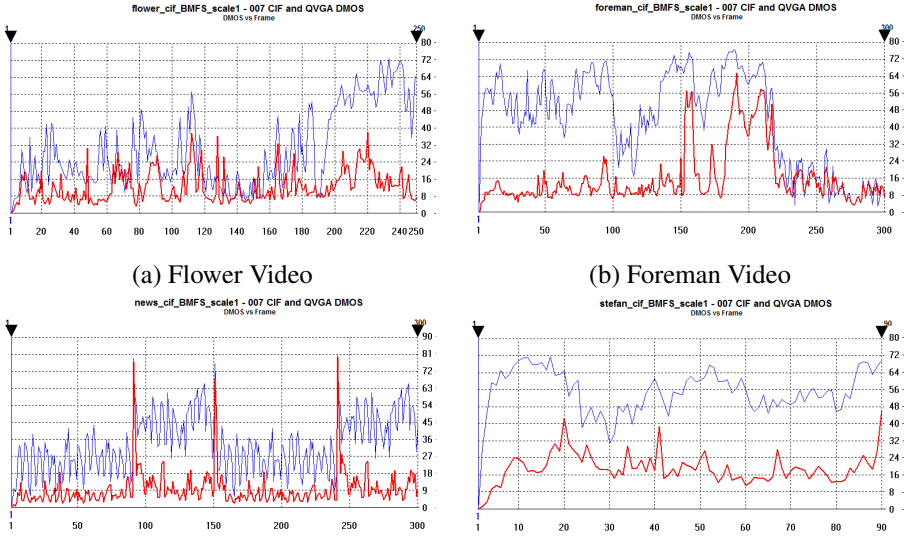
Figure 2.6: PSNR Comparison of Block Matching Full Search and Projection-Based Motion Estimation

hardware or an advanced GPU which has thousands of cores are used.

$$E(d) = \sum_{p \in R} |I(p, t_1) - I(p + d, t_2|^2 \tag{2.3}$$

where t_1 and t_2 indicate frames observed in that time, I indicates the luminance value of the pth pixel.

The PSNR, DMOS and ADMOS comparisons of motion estimation algorithms, which are explained above, can be found in the figures 2.6 - 2.8. In these figures, two consecutive frames are used to evaluate the performance of the motion estimation algorithms. By using the current frame as reference frame, motion of the previous frame with respect to reference frame is estimated and compensated. Compensated frame and reference frame are compared by using PSNR, DMOS and ADMOS metrics. In the figures 2.6 - 2.8, Block Matching Exhaustive Search Motion Estimation algorithm's results are shown in red; while Projeciton-Based Motion Estimation algorithm's results are shown in blue. It is very important to note that PSNR graphs are shown as a mirror version with respect to x-axis. While maximum value is located at the bottom of the graph, the minimum value is located at the top.



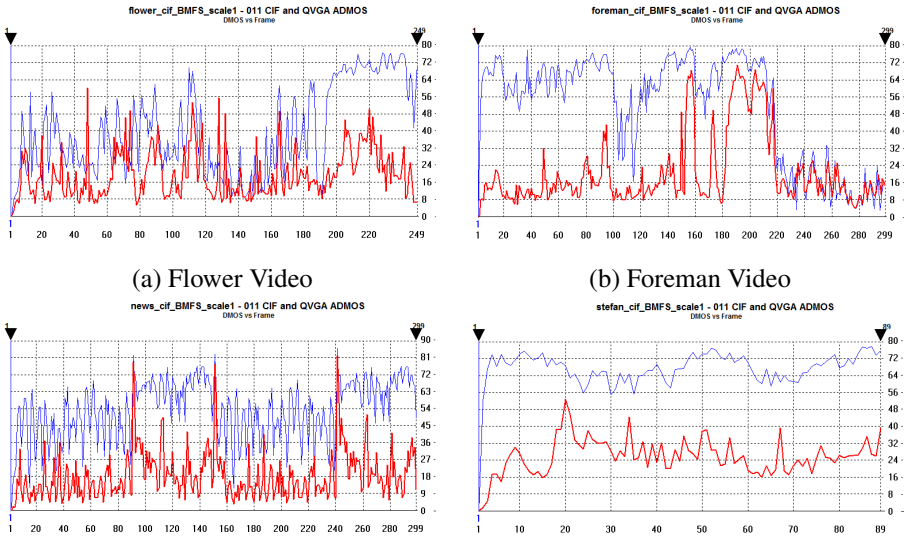
(a) Flower Video

(b) Foreman Video

(c) News Video

(d) Stefan Video

Figure 2.7: DMOS Comparison of Block Matching Full Search and Projection-Based Motion Estimation



(a) Flower Video

(b) Foreman Video

(c) News Video

(d) Stefan Video

Figure 2.8: ADMOS Comparison of Block Matching Full Search and Projection-Based Motion Estimation

Table 2.3: Execution time of motion estimation algorithms for different resolutions on INTEL architecture whose details are listed in Appendix D

	QCIF	CIF	SD	HD	FHD
PBME	1 msec	6 msec	24 msec	54 msec	120 msec
BMFS	151 msec	744 msec	3304 msec	7753 msec	17714 msec

Table 2.4: Execution time of motion estimation algorithms for different resolutions on ARM architecture whose details are listed in Appendix D

	QCIF	CIF	SD	HD	FHD
PBME	10 msec	50 msec	200 msec	450 msec	1010 msec
BMFS	1200 msec	5920 msec	26130 msec	60760 msec	139680 msec

The interpretations of figures 2.6 - 2.8 are as follows.

- When there is not any motion between consecutive frames or when there is just global motion between consecutive frames, two motion estimation algorithm has same performance as it can be seen in figures 2.6b, 2.7b and 2.8b. After 240th frame, both motion estimation algorithms have same performance.

- When scene change occurs, both motion estimation algorithms fail as it can be seen in figures 2.6c, 2.7c and 2.8c. The three minimum values which are shown in these figures happens because of the scene change.

- If there is a dominant local motion between two consecutive frames, Block Matching Motion Estimation algorithm outgoes Projection-based Motion Estimation algorithms as it can be seen in figures 2.6 - 2.8.

Although Block Matching Full Search algorithm outgoes Projection-Based Motion Estimation algorithm, Block Matching Full Search algorithm's computational load is much more higher than Projection-Based Motion Estimation algorithm. The execution time of motion estimation algorithms for different resolutions can be found in tables 2.3 and 2.4.

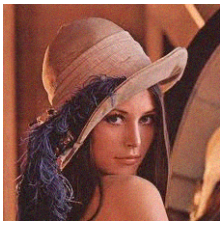
Figure 2.9: Illustration of the effect of the range and distance filters for bilateral filter.

2.3 Bilateral Filter

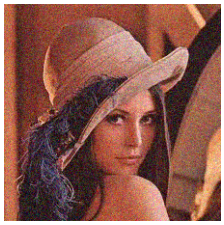
Bilateral filtering is a special kind of Gaussian filter. The advance of bilateral filtering is preserving the slope of edge while eliminating noise. It has composed of two filters: range filter and distance filter. The distance filter is very similar with the classical Gaussian filter, whereas range filter is special filter giving the characteristic of the bilateral filter. Distance filter is checking for how close neighbour pixel and the center pixel are so that it can assign larger weight to closer pixels; whereas range filter is checking for how similar neighbour and center pixel are so that it can assign larger weight to similar pixels. The effect of the filters by using different variance values is shown on the figure below.

As shown at figure 2.9 that, if we increase the variance of the distance filter, the output image suffers from "cartoonize effect", whereas if we increase the variance of the range filter, the bilateral filter is very similar with the classic Gaussian filter. In our experiment, we set the variance of the distance value to a fixed value and we change the variance of the range filter. The mathematical formula for bilateral filter

(a) Original Lena Image



(b) Gaussian Noise with 0.001



(c) Gaussian Noise with 0.006

(d) Gaussian Noise with 0.01

Figure 2.10: Gaussian noise is added to original Lena image by using different variance values

is as follows.[5]

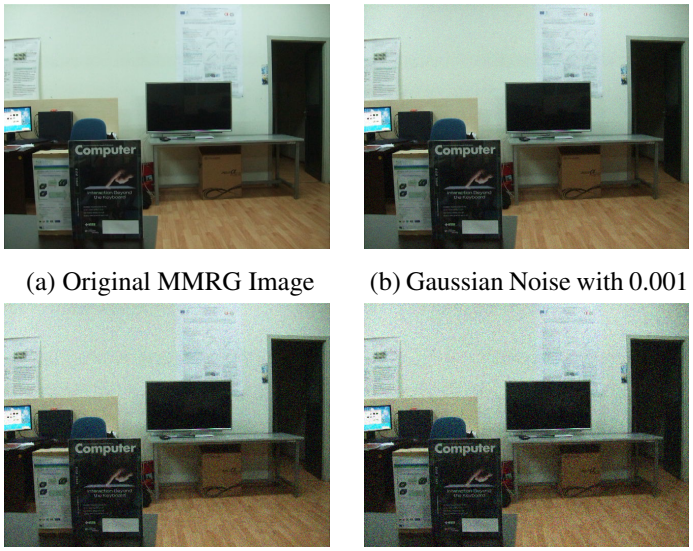
$$I_{filtered}(i,j) = \frac{\sum_{k,l} I(k,l) * w(i,j,k,l)}{\sum_{k,l} w(i,j,k,l)} \tag{2.4}$$

where the weight (w(i,j,k,l)) is calculated as

$$w(i,j,k,l) = e^{-\frac{(i-k)^2+(j-l)^2}{2\sigma_d^2} - \frac{||I(i,j)-I(k,l)||^2}{2\sigma_r^2}} \tag{2.5}$$

In order to test the performance of the bilateral filter, noisy images are prepared by using White Gaussian Noise with different variances. The noisy images which are used in our experiment can be seen in the figure 2.10 and figure 2.11.

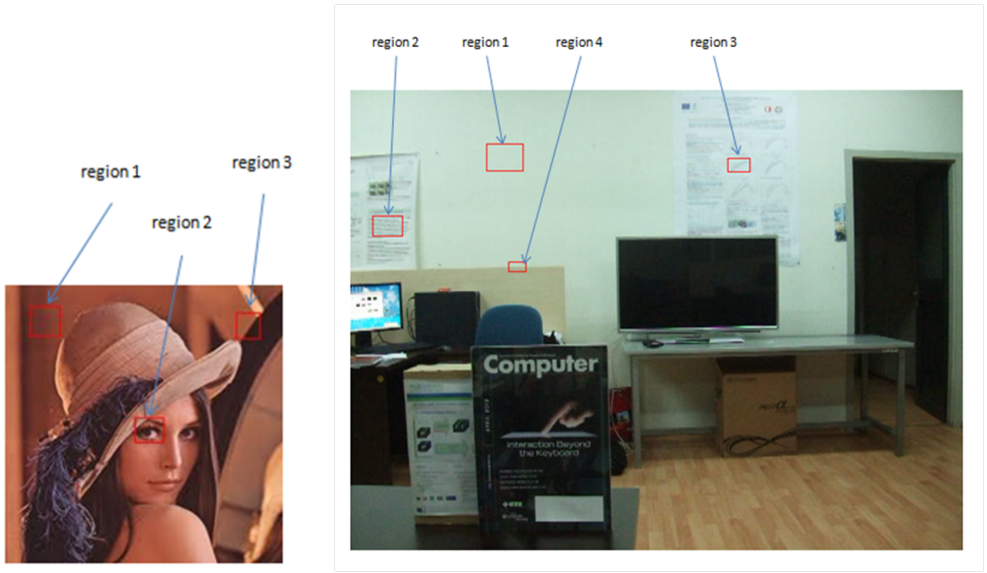
PSNR table is presented in figure 2.13 in order to understand the effect of the bilateral filter for different regions, which is shown in figure 2.12, (for Lena Image; region 1 is the smooth region, region 2 is the high frequency region and region 3 is the edge region; for MMRG Image, region 1 is the smooth region, region 2 and 3 are the high frequency region and region 4 is the edge region) and for different noise levels.



(a) Original MMRG Image (b) Gaussian Noise with 0.001

(c) Gaussian Noise with 0.006 (d) Gaussian Noise with 0.01

Figure 2.11: Gaussian noise is added to original MMRG image by using different variance values



(a) Lena Image (b) MMRG Image

Figure 2.12: Examined regions for MMRG Image and Lena Image

PSNR values of 11x11 bilateral filters (dB)

images	noise variance	region	noisy image	BF(50)	BF(200)	BF (1800)	ABF	ABF_CLC
Lena Image	0,001	1	33,59	37,38	41,61	42,46	41,46	**42,51**
		2	32,94	**33,09**	32,56	26,40	33,04	31,61
		3	32,80	35,11	**35,70**	32,57	34,66	34,45
		total	33,45	34,74	34,88	29,79	**35,13**	32,85
	0,006	1	26,08	27,24	29,91	**39,11**	26,71	34,79
		2	26,61	26,78	**27,03**	25,18	26,67	26,54
		3	26,69	27,58	29,17	30,50	26,94	**30,68**
		total	25,87	26,55	27,93	29,19	26,18	**29,21**
	0,01	1	23,65	24,52	26,26	**35,05**	23,93	30,44
		2	23,91	24,14	24,47	**24,61**	23,91	24,53
		3	23,58	24,07	25,28	**28,63**	23,68	26,71
		total	23,67	24,18	25,27	**28,10**	23,80	26,67
MMRG Image	0,001	1	33,95	39,18	44,39	46,50	44,19	**46,52**
		2	33,20	34,44	34,41	31,45	**34,61**	32,03
		3	32,78	34,94	**36,27**	34,06	36,11	34,71
		4	33,64	35,94	**36,83**	31,34	35,79	34,12
		total	33,51	36,50	**37,98**	34,07	37,63	36,10
	0,006	1	26,21	27,48	30,32	**40,89**	26,94	36,68
		2	25,70	26,65	28,31	**31,93**	26,10	30,52
		3	24,97	25,95	27,90	**32,72**	25,51	30,99
		4	27,40	28,43	30,23	31,26	27,59	**31,81**
		total	26,02	27,02	29,08	**32,66**	26,55	31,79
	0,01	1	23,99	24,83	26,78	**36,47**	24,23	31,57
		2	22,80	23,40	24,60	**30,25**	22,97	26,89
		3	24,57	25,24	27,12	**33,22**	24,77	30,32
		4	22,76	23,08	23,92	**27,42**	22,92	24,86
		total	24,03	24,73	26,23	**30,99**	24,24	28,77

Figure 2.13: PSNR results of the bilateral filter response to noisy images with different noise levels.BF(50) means that variance of the range filter is 50. ABF is the abbreviation of the Adaptive Bilateral Filter, whereas ABF CLC is the abbreviation of the "Adaptive Bilateral Filter Considering Local Characteristics"

As it can be seen in the figure 2.13, under the low variance noise condition, bilateral filter with low variance should be applied to the high frequency regions, whereas bilateral filter with high variance should be applied to low frequency regions. Under the high variance noise condition, bilateral filter with high variance should be applied to the noisy images.

Instead of selecting the values for different images and for different noise levels manually, an algorithm is needed to select the variance value automatically. Therefore, adaptive bilateral filter is explained in the next two subsection.

2.3.1 Adaptive Bilateral Filter

The aim of the adaptive bilateral filter is

- To improve the results of the output image for different regions (high frequency region and low frequency region)
- To improve the results of the output image for different noise levels. (robustness to noise)

Adaptive Bilateral Filter approach claims that variance of the range filter should be inversely proportional to the pixel fluctuations. By using this idea, the variance of the range filter is calculated by using the equations given below.[6]

$$\Delta = \frac{\sum_{q \epsilon \Omega} |\bar{I}_q - I_q|}{m}; \quad n = 11; \quad \sigma_d = 3 * n; \quad \sigma_r = \frac{96}{\Delta} \tag{2.6}$$

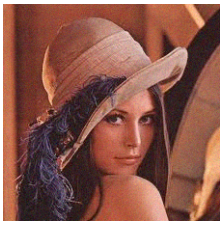
This approach is very useful for different regions. When we are in the low frequency region with low pixel fluctuations, variance of the range filter is high. Therefore, bilateral filter compress the noise more powerfully when we compare with the high frequency region. To prove that our statement is true, figure 2.14 is a good choice to show this.

The difference map shows that adaptive bilateral filter is not modifying the high frequency regions (such as texture regions or edges); on the other hand, it applies powerful filtering for the low frequency regions. This algorithm is very effective when we consider the different regions, however it is not that far effective for different noise levels as it can be seen in the figure 2.13.

2.3.2 Adaptive Bilateral Filter Considering Local Characteristics

This approach chooses bilateral filter's range parameter considering the "Gradient Map". Gradient map of an image is created using horizontal and vertical Gaussian gradients. After that, they associate the range parameter of bilateral filter with gradient value.

(a) Original Lena Image



(b) Gaussian Noise with 0.001

(c) Adaptive Bilateral Filter is applied to the image which is shown in (b)

(d) Difference map ((b) - (c))

Figure 2.14: The illustration of the working principle for Adaptive bilateral filter for different regions

As it is mathematically stated in the equations 2.4 and 2.5, σ_r and σ_d are the standard deviations of the range filter and the domain filter respectively. These parameters control the strength of the bilateral filter. As it is stated before, range filter is important filter which makes bilateral filter unique. Therefore, this approach adjusts the standard deviation of the range parameter, σ_r, according to the context of the image in order to process the image locally and adaptively.Gaussian gradient map is used since it gives the local information about regions. The Gaussian gradient map equation is given as

$$G = \sqrt{G_x^2 + G_y^2} \tag{2.7}$$

G_x and G_y are two directional horizontal and vertical Gaussian gradients respectively.[7]

According to article, the value of current gradient and the value of the range standard deviation, σ_r have certain relationship. Therefore, they set the range filter standard deviation, σ_r as the inverse function of gradient to classify the pixels automatically without setting threshold manually. The range filter variance is calculated according

to equation below.

$$\sigma_r(x,y) = k(x,y) * \frac{1}{G(x,y)} + b(x,y) \tag{2.8}$$

where (x,y) denotes the current pixel coordinate; k(x,y) and b(x,y) are the parameters which control the relation of current $\sigma_r(x,y)$ and the current G(x,y). For simplicity, k and b values are set to same value for the whole image. Moreover, k is equalized to maximum value of G which is shown in equation (2.7).

CHAPTER 3

LITERATURE SURVEY

Super resolution has been investigated over the past two decades. In these two decades, many different approaches have been proposed to obtain an output image having highest quality. These methods can be divided into two main groups, namely, single-frame super resolution methods and multi-frame super resolution methods. Before super resolution algorithms were proposed, basic interpolation algorithms had been used to upscale images. These basic interpolation algorithms are also used in today's world because of their simplicity and low computational load. Therefore, first of all, basic interpolation algorithms will be introduced in this chapter; after that super resolution algorithms will be explained.

3.1 Basic Interpolation Algorithms

Basic interpolation techniques rely on the surface fitting by using the known pixels. These techniques do not consider the feature of input image. Three well-known examples of these techniques are Nearest Neighbour Interpolation, Bilinear Interpolation and Bicubic Interpolation.

While Nearest Neighbour Interpolation fills all the missing pixels with the value of closest pixel, Bilinear Interpolation estimate missing pixels by assuming that the known pixels are located on the linear surface. Therefore, while Nearest Neighbour Interpolation causes jagged edges, Bilinear Interpolation incurs blur edges.

Bicubic Interpolation is the most widely used benchmark algorithm in the literature.

Every articles in the literature compares their algorithms with the Bicubic Interpolation as it is also done in this thesis. Every algorithm will be compared with the Bicubic Interpolation by using PSNR, DMOS and ADMOS quality metric. The results will be presented in Chapter 5.

Bicubic Interpolation is the two dimensional version of the Cubic Interpolation. In order to apply Bicubic Interpolation to an image; first, the image divides into four by four windows to calculate the sub-pixel values located between the middle of the two pixels. For every windows, missing pixels or sub-pixels are calculated in the x-direction by using Cubic Interpolation. After that, Cubic Interpolation is applied to find the missing pixels in the y-direction. This approach is used to take advantage of the separable filters. This approach requires less computational power when compared with the 2-D convolution.

Cubic Interpolation uses third degree polynomial and its derivative to find the missing values. Here is the derivation of the cubic interpolation. Assuming that the values and the derivatives of the points at x = 0 and x = 1 are known.[8]

$$f(x) = ax^3 + bx^2 + cx + d \tag{3.1}$$

$$f'(x) = 3ax^2 + 2bx + c \tag{3.2}$$

The values of the polynomial and its derivative at x = 0 and x = 1 are

$$f(0) = d \tag{3.3}$$

$$f(1) = a + b + c + d \tag{3.4}$$

$$f'(0) = c \tag{3.5}$$

$$f'(1) = 3a + 2b + c \tag{3.6}$$

The four equations ,which are listed above, can be written is as follows.

$$a = 2f(0) - 2f(1) + f'(0) + f'(1) \tag{3.7}$$
$$b = -3f(0) + 3f(1) - 2f'(0) - f'(1) \tag{3.8}$$
$$c = f'(0) \tag{3.9}$$
$$d = f(0) \tag{3.10}$$

Since we are using four by four window to determine the missing pixel, we know the values at the point x = -1, x = 0, x = 1 and x = 2, whose values are denoted as p_0, p_1, p_2, p_3 respectively. By convention, in order to find the value and derivative of the polynomial at x = 0 and x = 1, we use the formulas below.

$$f(0) = p_1 \tag{3.11}$$
$$f(1) = p_2 \tag{3.12}$$
$$f'(0) = \frac{p_2 - p_0}{2} \tag{3.13}$$
$$f'(1) = \frac{p_3 - p_1}{2} \tag{3.14}$$

By using the four formulas given above, we can rewrite the polynomial function's coefficients in terms of the known points as it is given above.

$$a = -\frac{p_0}{2} + \frac{3p_1}{2} - \frac{3p_2}{2} + \frac{p_3}{2} \tag{3.15}$$
$$b = p_0 - \frac{5p_1}{2} + 2p_2 - \frac{p_3}{2} \tag{3.16}$$
$$c = -\frac{p_0}{2} + \frac{p_2}{2} \tag{3.17}$$
$$d = p_1 \tag{3.18}$$

Therefore, the final formula becomes

$$f(p_0, p_1, p_2, p_3, x) = (-\frac{p_0}{2} + \frac{3p_1}{2} - \frac{3p_2}{2} + \frac{p_3}{2})x^3 + (p_0 - \frac{5p_1}{2} + 2p_2 - \frac{p_3}{2})x^2 + (-\frac{p_0}{2} + \frac{p_2}{2})x + p_1 \tag{3.19}$$

Every missing pixels which are located between x = 0 and x = 1 will be estimated by using the equation (3.19).

Bicubic interpolation produces higher quality images when compared with the bilinear and nearest neighbour interpolation. However, since the algorithm doesn't consider the feature of the images, the edge regions will be blurry and ringing effects are seen.

3.2 Single Frame Super Resolution

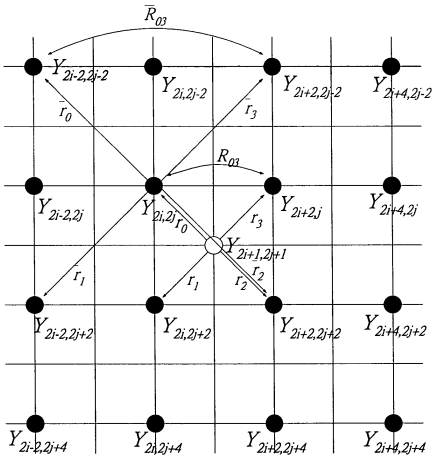
For single frame super resolution techniques, only spatial information of video is used to improve the resolution of the output image. A Fast Edge-Adaptive Interpolation and Iterative-Back Projection algorithm will be examined as example of single frame super resolution techniques. In this section, advantages and disadvantages of this technique will be discussed.

3.2.1 Fast Edge-Adaptive Interpolation

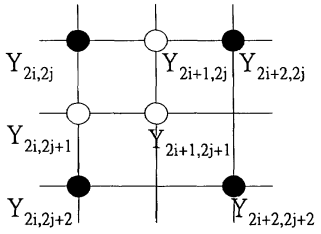
As it is mentioned in the previous section of this chapter, the basic interpolation techniques do not consider input image characteristics. The difference of this technique is that it looks for the best correlation between consecutive pixels (two by two pixels). Therefore, missing pixels will be filled by averaging the two pixels having best correlation. This leads to better quality for the edge region of the images and slope of edges is preserved. As a result, the output high resolution image has sharp edges depend on the edge condition when we compare with Bicubic Interpolation.

Figure 3.1a shows pixel map of the up-sampled input image by a factor of two. While $Y_{2i+2p,2j+2q}$ pixels are low resolution pixels, $Y_{2i+2p+1,2j+2q+1}$ pixels are high resolution pixels where $p, q \in Z$. This algorithm preserves the low resolution pixels by up-sampling them on high resolution grid as it can be seen in figure 3.1b. The missing pixels which are labelled as $Y_{2i,2j+1}$, $Y_{2i+1,2j}$ and $Y_{2i+1,2j+1}$ are estimated by using the Algorithm 1 which can be found in Appendix A.

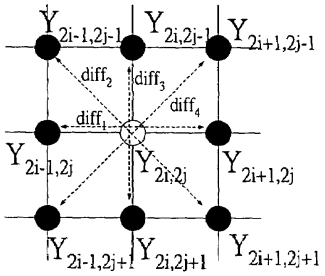
As it can be seen in Algorithm 1, this algorithm can be parallelized since the main



(a) Pixel Map of High Resolution Image



(b) 2x2 block of pixels with low resolution pixels labelled as filled circle and high resolution pixels(missing pixels) labelled as hollow circle



(c) Calculating similarities for every direction on edge region

Figure 3.1: Fast Edge-Adaptive Interpolation Algorithm [9]

loop has no inter-dependency between different iterations. The results of this algorithm will be shared in Chapter 5.

3.2.2 Iterative Back-Projection Approach

This approach is very similar with the reconstruction of 2-D object from 1-D projections in Computer-Aided Tomography. Estimated high resolution image is constructed by upsampling the averaged form of the motion compensated low resolution images. This estimated high resolution image is used to form estimated low resolution images to compare with the original low resolution images. Error between the original low resolution images and estimated low resolution images are estimated to

back-project it to high resolution image. The mathematical expression can be found below.[10]

$$\widehat{x}^{n+1}[n_1, n_2] = \widehat{x}^{n}[n_1, n_2] + \sum_{m_1,m_2 \in Y_k^{m_1,m_2}} (y_k[m_1, m_2] - \widehat{y}_k^n[m_1, m_2]) \times h^{BP}[m_1, m_2; n_1, n_2] \tag{3.20}$$

where $\widehat{y}_k^n = DH^{PSF}\widehat{x}^n$ are the estimated low resolution images after nth iteration using the imaging model. In our experiment, 3x3 box filter is used for H^{PSF}. The important part of this approach is choosing the iterative back projection kernel which is represented as h^{BP}. In our experiment, 3x3 box filter is also used for h^{BP}, since Irani et al claims that choosing h^{BP} equals to H^{PSF} is a good choice because this choice prevents nonzero frequency components from varying much in few iterations, hence remaining similar in their initial value.For the same reason, noise is not amplified by such frequency components. The other possible choice of h^{BP} is choosing inverse filtering for debluring. Irani's article claims that this choice tends to amplify noise.

In Irani's and Peleg's article, motion estimation method which is proposed by Keren et al is used to compensate the motion between consecutive frames. This motion estimation techniques assumes that motion between consecutive frames are very small. Therefore, it uses Taylor expansion to find the final equation. Since small displacement between consecutive frames is not the usual scenario, this method is implemented as single frame super resolution algorithm. It is because of the fact that Irani also states that Iterative-Back Projection algorithm can also be implemented as single-frame super resolution algorithm. Therefore, this approach is explained in single frame super resolution section. In this thesis, the result of Single-Frame Iterative-Back Projection Super Resolution method will also be shared in Chapter 5.

3.3 Multi-Frame Super Resolution

Temporal information of input video is used to enhance the resolution of the output video in multi-frame super resolution algorithms. In other words, a set of frame is used to enhance the resolution of output frame. These techniques are much more

powerful to recover the lost information of images in case of motion estimation is done accurately. However, for general content movies, it is hard to accomplish motion estimation since general content movies have complex motion which consists of global and local motion.

In this section, multi-frame super resolution methods will be explained. Moreover, advantages and disadvantages of these methods will be discussed.

3.3.1 Interpolation-Restoration Type Method

Interpolation- restoration type method consists of three steps; namely, registration, interpolation and restoration. In the registration step, according to the reference frame, motion vectors of other frames are estimated by using a motion estimation algorithm. In this thesis, block matching exhaustive search is used for the registration step. After motion of every block is estimated and compensated; in the interpolation part, high resolution output image is obtained by using reference frame and motion compensated form of other frame. The missing pixels of the high resolution output frame are calculated inversely proportional to the sub-pixel distance of the low resolution reference frame pixels and low resolution compensated from of the other frames. In the restoration part, a filter is used for denoising and deblurring.

This method is the basic method among the multi-frame super resolution methods. It is easy to implement and suitable for parallel implementation. Nevertheless, the artifacts for the general content movies are quite disturbing. It is because of the fact that motion estimation cannot be done accurately even if we use the block matching exhaustive search or optical flow, which are the best methods among the motion estimation algorithms. Therefore, especially at the edge region of the image, the staircase effect is observed because of the fact that blocks with fixed square size are used in the registration part.

3.3.2 Model Based Super Resolution

This method tries to minimize the cost function which is given below. [11]

$$Cost = \sum_{k=1}^{K} ||DHF_k x - y_k||_2^2 \tag{3.21}$$

Equation 3.21 is used by assuming the following forward problem equation.

$$y_k = DHF_k z + n_k \quad k = 1, ...K \tag{3.22}$$

D, H and F_k are decimation operator, blurring operator and motion warp operator respectively. x is the estimated high resolution image while z is the original high resolution image. y_k is the set of adjacent low resolution frames. The number of elements in this set will be denoted by K. The cost function, which is shown in equation (3.21), should be calculated for every iteration and for every adjacent frames, and the estimated high resolution image should be updated after every iteration.

In order to minimize the cost function, gradient descent method was used in this thesis. The derivative of the cost function is computed as follows.

$$\nabla Cost = \sum_{k=1}^{K} 2(DHF_k)^T((DHF_k)x - y_k) \tag{3.23}$$

By substituting $\nabla Cost$ into $x^{n+1} = x^n - \beta \nabla Cost$,

$$x^{n+1} = x^n + \beta \sum_{k=1}^{K} (DHF_k)^T(DHF_k x^n - y_k) \tag{3.24}$$

where n denotes the number of iteration. In this thesis, β is selected as 0.5 and Projected-Based Motion Estimation and Block Matching Full Search algorithm were used to compensate motion between consecutive frames. Therefore, if there is a global motion between consecutive frames, this algorithm gives high quality image no matter which motion estimation algorithm is used; whereas if there is a local motion between consecutive frames, while this algorithm with Projection-Based Motion

Estimation gives low quality output image with motion blur, the algorithm which use Block Matching Motion Estimation gives high quality output result. This approach is sensitive to scene cuts. If the consecutive frames are different, the performance of this approach will decrease sharply.

The article claims that this approach may produce unstable result for noisy input images, which means small amounts of noise on the input images will be amplified by this algorithm and many disturbing artifacts can be seen on the output image. Therefore, regularization is needed to obtain stable result. Regularization term compensates the missing measurement information with some general prior information of the desirable high resolution image. The generalized minimization cost function is as follows.

$$Cost = \sum_{k=1}^{K} ||DHF_k x - y_k||_2^2 + \lambda \Upsilon(x) \tag{3.25}$$

where λ,the regularization parameter , will be used to arrange the weight of the regularization cost function which is shown by Υ.

One of the most widely used regularization cost function is the Tikhonov cost function

$$\Upsilon_T(x) = ||Cx||_2^2 \tag{3.26}$$

where C is the highpass filter. The logic behind this regularization is to smooth the result to compress the noise. Since both noisy and edge regions have high frequency energy, they will be compressed in the regularization process and the output high resolution image will not contain sharp edges. This is not a desired outcome for the super resolution techniques.

Other widely used regularization method in the literature is the total variation (TV) method. The TV criterion penalizes the total amount of the change in the image by using the following regularization cost function. Since TV regularization does not

penalize the steep local gradient, it preserves edge regions.

$$\Upsilon_{TV}(x) = ||\nabla x||_1 \tag{3.27}$$

3.3.3 Bayesian Methods

Bayesian methods try to estimate the most probable super resolution image by maximizing a posteriori pdf, namely, $P(x|y_1, y_2, ...y_K)$ with respect to x. The mathematical representation is as follows.

$$\widehat{x}_{MAP} = argmax(P(x|y_1, y_2, ...y_K)) \tag{3.28}$$

Bayes' theorem states that

$$P(A|B) = \frac{P(B|A) * P(A)}{P(B)} \tag{3.29}$$

If we apply the Bayes' theorem which is stated at equation (3.29) on the Bayesian general equation stated at equation (3.28), the equation becomes

$$\widehat{x}_{MAP} = argmax(\frac{P(y_1, y_2, ...y_K|x) * P(x)}{P(y_1, y_2, ...y_K)}) \tag{3.30}$$

Since $P(y_1, y_2, ...y_K)$ term is used for normalization and will not affect the maximization, this term can be removed.

$$\widehat{x}_{MAP} = argmax(P(y_1, y_2, ...y_K|x) * P(x)) \tag{3.31}$$

Assuming that all the low resolution images, y_k, are independent. The equation becomes,

$$\widehat{x}_{MAP} = argmax(\{\prod_{k=1}^{K} P(y_k|x)\} * P(x)) \tag{3.32}$$

3.3.3.1 Maximum Likelihood Method

If there is no priori information for the high resolution image, maximum likelihood method is used to find the best estimate of the high resolution image by using the formula.

$$\widehat{x}_{ML} = argmax(\prod_{k=1}^{K} P(y_k|x)) \tag{3.33}$$

Assuming that the noise distribution in equation (4.1) is Gaussian, the noise is modeled as

$$P(n) = \frac{1}{(2\pi\sigma)^{N/2}} e^{-\frac{n^T n}{2\sigma^2}} \tag{3.34}$$

where N denotes the size of the noise vector. By replacing the noise expression, denoted as "n", with $(y_k - (DHF_k)x)$ according to equation (4.1), equation (3.34) becomes,

$$P(y_k|x) = \frac{1}{(2\pi\sigma)^{N/2}} e^{-\frac{(y_k - DHF_k x)^T (y_k - DHF_k x)}{2\sigma^2}} \tag{3.35}$$

By concatenating equation (3.33) and equation (3.35),

$$\widehat{x}_{ML} = argmax(\prod_{k=1}^{K} \frac{1}{(2\pi\sigma)^{N/2}} e^{-\frac{(y_k - DHF_k x)^T (y_k - DHF_k x)}{2\sigma^2}}) \tag{3.36}$$

In order to find the maximum of the equation, we need to take the logarithm of the equation so that we use the advantages of the logarithm on the derivation. Since logarithm is the non-decreasing function, it will not affect the maximum.

$$\widehat{x}_{ML} = argmin(\sum_{k=1}^{K} \frac{(y_k - DHF_k x)^T (y_k - DHF_k x)}{2\sigma^2} - ln(\frac{1}{(2\pi\sigma)^{N/2}})) \tag{3.37}$$

Therefore, the cost function for maximum likelihood solution under the Gaussian Distributed noise is

$$Cost = \sum_{k=1}^{K} ||DHF_k x - y_k||_2^2 \tag{3.38}$$

As it can be noticed easily, equation (3.21) and equation (3.38) are the same. Although they are applying different ways to solve the linear equation, they obtained same cost function.

3.3.4 Super Resolution With Probabilistic Motion Estimation

This algorithm assumes that low resolution images are obtained without applying blurring operation and adding noise operation in common. It proposes new way to prevent the artifacts caused by motion estimation. For this purpose, the algorithm uses the probabilistic motion estimation in the reconstruction instead of using classical motion estimation methods in the registration part. For this purpose, this approach modifies the formulation posed in equation (3.21) by proposing the following probabilistic maximum likelihood penalty, [12]

$$Cost = \frac{1}{2} \sum_{m=1}^{M} \sum_{t=1}^{T} ||DHF_m x - y_t||^2 w_{m,t} \tag{3.39}$$

If the size of the maximal translation is at most D pixels, then a set of $M = (2D+1)^2$ displacements covers all the possible motion vectors. Therefore, F_m indicates the all possible motion vectors.

In order to minimize the cost function, pixel-wise method is used to obtain a high resolution image with better optical resolution. Therefore, every pixels are estimated by using the formula below.

$$x(i,j) = \frac{\sum_{[k,l] \in N(i,j)} \sum_{t=1}^{T} W_{m,t}[k,l] y_t[k,l]}{\sum_{[k,l] \epsilon N(i,j)} \sum_{t=1}^{T} W_{m,t}[k,l]} \tag{3.40}$$

by defining the neighborhood set as

$$N(i,j) = [k,l] | \forall m \in [1,M], s.k = i + dx(m), s.l = j + dy(m) \tag{3.41}$$

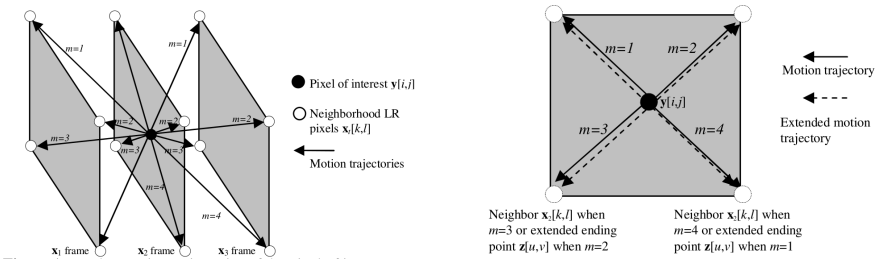
where s is the scaling factor.

Equation 3.40 shows that every candidate pixels located at the adjacent frames will be multiplied by a weight and the result will be normalized to obtain the pixel value which is located at the high resolution frame. This approach is very similar with the block matching exhaustive search algorithm which is explained in Section 2. The main difference between these two algorithms is that while block matching exhaustive search motion estimation seeks for the best match block, probabilistic motion estimation super resolution algorithm estimates the weight by considering the question how much these two blocks are similar. The weights are calculated by using the formula given below.

$$W_{m,t}[i,j] = e^{\frac{||R_{i,j}(DF_m x - y_t)||_2^2}{2\sigma^2}} f(\sqrt{(dx(m))^2 + (dy(m))^2 + (t-1)^2} \tag{3.42}$$

As it can be seen in equation (3.42), the formula is very similar with the weight equation in the bilateral filter. The only difference with bilateral filter weight equation is that probabilistic motion estimation approach are using a patch of predetermined size instead of using only one pixel. Every patch or block in the current frame are shifted by using set of probable motion and the similarities will be evaluated by using the Euclidean spatial distance of the blocks and Euclidean luminance distance of the blocks.

Although this approach arises from finding a better solution the weakness of the motion estimation, it suffers from the blurring artifacts because of its similarity to bilateral filter denoising algorithm. Therefore, this approach does not represents a good quality of high resolution image. In addition to that, this approach is the most computationally costly approach among all the super resolution methods.



(a) Twelve motion trajectories of the pixel of interest y[i,j]

(b) Conventional error computation which is obtained from the difference between y[i,j] and $x_t[k,l]$(The new error is the difference between xt[k, l] and z[u, v] on the extended motion trajectory, and the second frame, x2, is shown.)

Figure 3.2: Illustration of Extended Motion Trajectory

3.3.5 Fast Video Interpolation/Up-Sampling Using Linear Motion Model

This approach is very similar with the Super Resolution with Probabilistic Motion Estimation approach which has been explained in the previous section. This algorithm improves its performance by using linear motion assumption. Instead of using the spatial filter which is formulated in equation (3.42), another filter to penalize the similarity along the motion trajectory is used beside the domain filter. The formula which is used to compute weight is as follows.[13]

$$W_{m,t}[k,l] = e^{\frac{||R^1_{k,l}(x_2 - x_t)||_2^2}{2\sigma_1^2}} e^{\frac{||R^2_{k,l}(F^L_{m,t} z - x_t)||_2^2}{2\sigma_2^2}} \tag{3.43}$$

where z is the low resolution frame for which the extended ending point locates. $(F_{m,t})_{m=1}^{M}$ denotes the possible warping operator. As it is stated in the previous section, $R_{k,l}$'s are the predetermined patch size to compute the error better. The figures illustrated at figure 3.2a and figure 3.2b will be very useful to explain extended ending point. As it can be seen in figure 3.2a, figure 3.2b and algorithm 4, the missing pixels are estimated by using the neighbor pixel of the missing pixel located in the previous frame(x1 in figure 3.2a) and extended ending point of this neighbor pixel located in the next frame(x3 in figure 3.2a).

As it is stated before, this approach assumes that there is a linear motion between the

consecutive frames; therefore, it is penalizing the similarity along the motion trajectory. However, in order to reduce the computational load of the algorithm, maximum motion vector is defined as one as it is shown on the figure 3.2a. Therefore, while this algorithm produces good results for the videos which have slow motion between consecutive frames, it fails for the movies which consists fast moving objects. If we increase the search window; since averaging many pixels, the output image suffers from the blurriness.

CHAPTER 4

RESULT OF ANALYZED ALGORITHMS

In this chapter, the performance-wise and complexity-wise results will be discussed. The results of analyzed algorithms which are explained in Chapter 3, will be presented. The results are composed of PSNR, DMOS and ADMOS graphs. PSNR, DMOS and ADMOS results are obtained by using PQASW Picture Quality Analyzer which is developed by Tektronix. The user inferface of this tool can be found in figure 4.1. While the original video can be seen at the top left side, the processed video is located at the top right side. The rectangle on the original video and processed video indicates region of interest. By using this region of interest, perceived difference map is estimated. Graph indicates the metric results versus frame numbers. In the first section of this chapter, algorithms' performance-wise results will be presented.

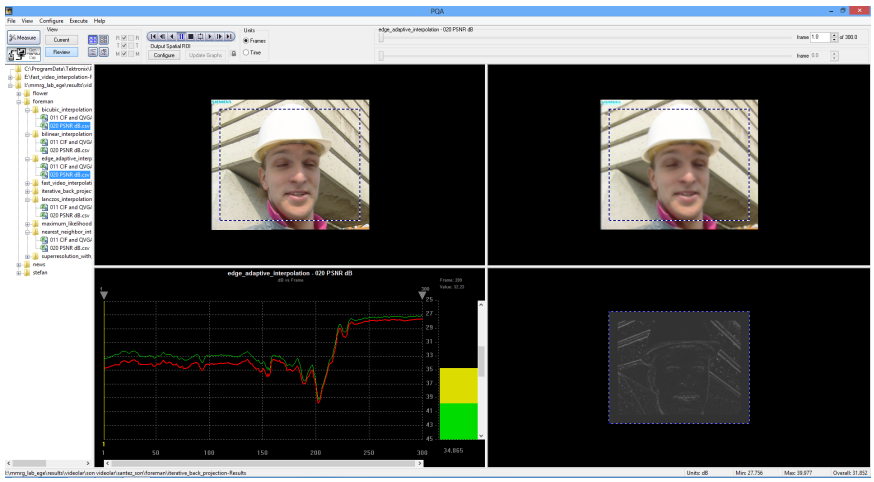


Figure 4.1: Picture Quality Analyzer User Interface

In the second section of this chapter, comparison of analyzed algorithms will be

presented. The measurements are taken by using Arçelik Television and Lenova Thinkpad T550 whose processors' details can be found in Appendix D.

4.1 Performance Results of Super Resolution Algorithms

The algorithms are applied to four videos which are widely used in the literature; namely, Flower, Foreman, News and Stefan. CIF format of these videos whose chroma sub-sampling is 4:2:0, are downsampled by a factor of two to obtain QCIF format of input videos by using the equation 4.1.

$$y_k = DHz_k \quad k = 1, ...K \tag{4.1}$$

D and H are decimation operator and blurring operator respectively. 3x3 box filter is used as blurring operator. While z_k is original kth high resolution frame y_k is kth low resolution frame. The total number of frame in a video is denoted as K.

The decimated low resolution video is upscaled by the analyzed algorithms by a factor of two. PSNR, DMOS and ADMOS graphs are given below. All the algorithms will be compared with Bicubic Interpolation. While the blue line in the graphs indicates Bicubic Interpolation, red line indicates the analyzed algorithm. It is very important to note that PSNR graphs are shown as a mirror version with respect to x-axis. While maximum value is located at the bottom of the graph, the minimum value is located at the top.

The characteristic of the videos which are used in the measurements are quite important. While Flower video has continuous global motion and texture regions, Foreman video has dominant local motion at the beginning of the video and global motion at the end. It is important to note that Foreman video has a set of scene whose energy is concentrated on the low frequency around 200th frame. News video has not any global motion whereas it has dominant local motion between 90th and 150th frames and after approximately 240th frame. Also it should be emphasized that news video has three scene cuts on the television located behind the newsmen. While Stefan Video has global motion at first 15 frames and the frames between approximately

50th and 75th, it has local motion almost every frame of the video. However, local motion which exists in Stefan Video is not so dominant.

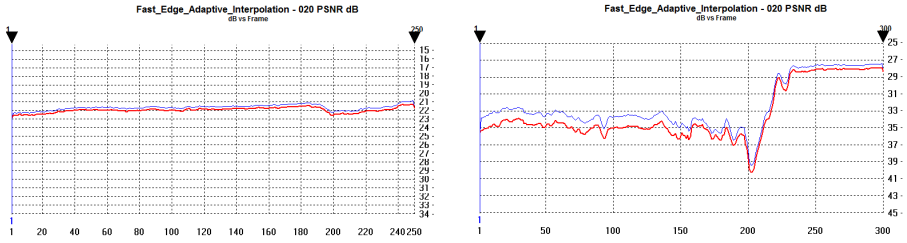
The selected frames of each video can be found in the Appendix C in order to show the characteristic of the videos. Upscaled frames by using the analyzed algorithms and perceived difference maps for PSNR, DMOS and ADMOS are also provided in Appendix C. 200th frame is selected for flower video in order to show the effect of foreground object on DMOS and ADMOS quality metric and the performance of the analyzed algorithms on texture regions. 155th frame is selected for foreman video in order to illustrate the performance of the algorithms for local motion. 22nd frame is selected for news video in order to show the performance of the algorithms in case of scene cuts occurred.

4.1.1 Fast Edge-Adaptive Interpolation

The algorithm outgoes Bicubic Interpolation for the four videos when we consider PSNR results. When we consider DMOS and ADMOS results; performance of Bicubic Interpolation Fast Edge Adaptive Interpolation are same for flower video. It is because of the fact that the flower videos has too much texture regions. It is an expected behaviour that this algorithm fails for texture regions. However, when we consider foreman, news and stefan video, Fast-Edge Adaptive algorithm has much better performance since texture region does not exist in these videos.

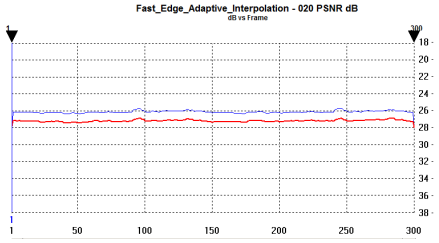
As it is also seen in perceived difference map images for foreman and news videos, shown in Appendix C, Fast Edge Adaptive Interpolation estimates the missing pixels better in the edge regions.

The PSNR results of the Fast Edge-Adaptive Interpolation can be found in the figures 4.2.

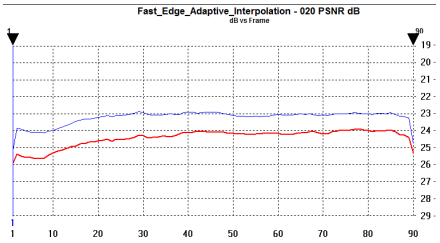


(a) Flower Video

(b) Foreman Video



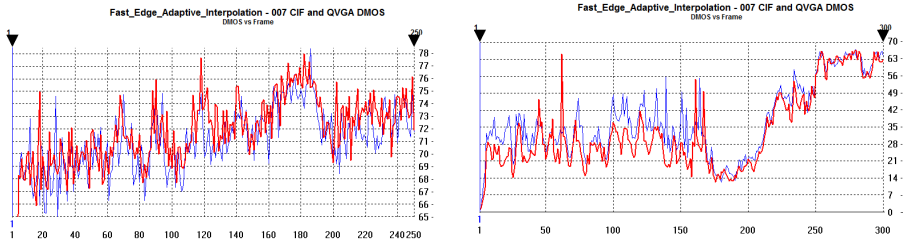
(c) News Video



(d) Stefan Video

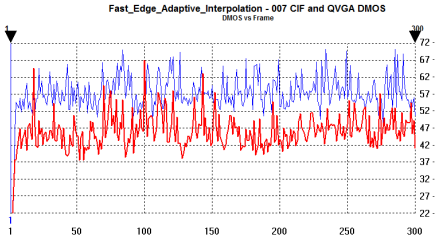
Figure 4.2: PSNR Comparison of Fast Edge-Adaptive Interpolation and Bicubic Interpolation

The DMOS results of the Fast Edge-Adaptive interpolation can be found in the figures 4.3.

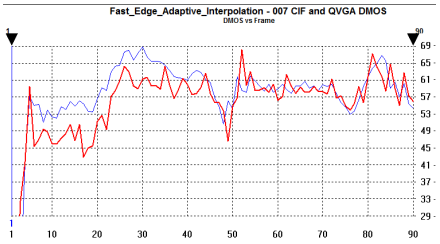


(a) Flower Video

(b) Foreman Video



(c) News Video

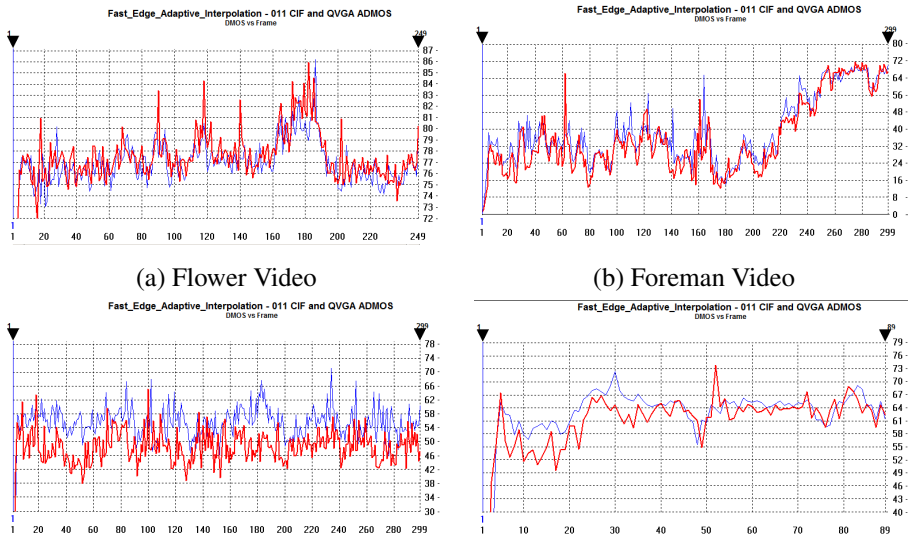


(d) Stefan Video

Figure 4.3: DMOS Comparison of Fast Edge-Adaptive Interpolation and Bicubic Interpolation

The ADMOS results of the Fast Edge-Adaptive interpolation can be found in the

figures 4.4.



(a) Flower Video

(b) Foreman Video

(c) News Video

(d) Stefan Video

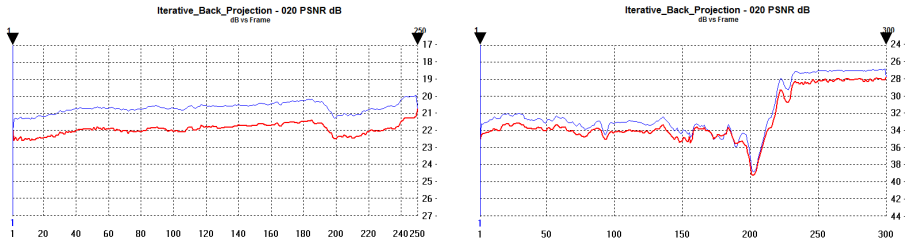
Figure 4.4: ADMOS Comparison of Fast Edge-Adaptive Interpolation and Bicubic Interpolation

In these results, the other critical point can be noticed when 200th frame of the foreman video is considered. This frame is full of low frequency regions and both algorithm has excellent performance for low frequency regions.

4.1.2 Iterative-Back Projection Algorithm

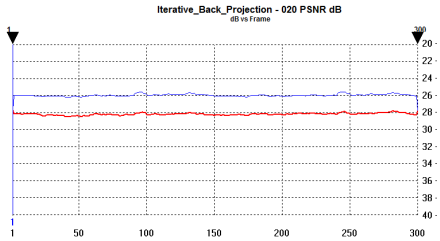
Single-Frame Iterative-Back Projection algorithm outgoes Bicubic Interpolation for the four videos when we consider PSNR, DMOS and ADMOS results. Since the decimated videos are obtained using the equation 4.1 which includes blurring kernel, Single-Frame Iterative-Back Projection algorithm deblurs the output videos as it is stated in Irani's article. Therefore, all the results are better than Bicubic Interpolation.

The PSNR results of the Iterative-Back Projection algorithm can be found in the figures 4.5. For this approach, single frame is used with twenty iterations.

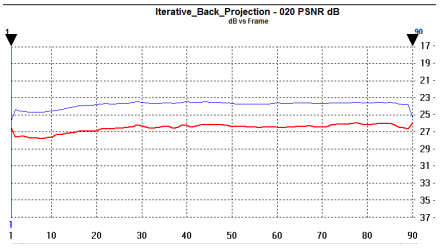


(a) Flower Video

(b) Foreman Video



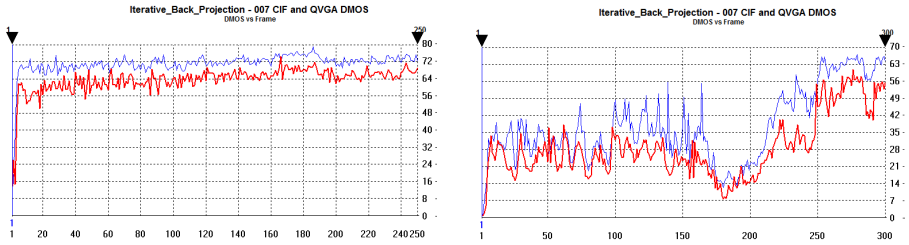
(c) News Video



(d) Stefan Video

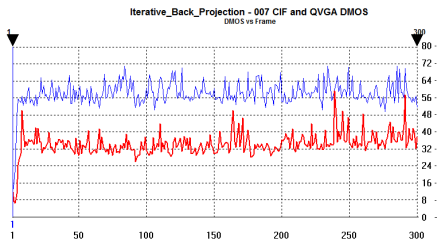
Figure 4.5: PSNR Comparison of Iterative Back Projection and Bicubic Interpolation

The DMOS results of the Iterative-Back Projection algorithm can be found in the figures 4.6.

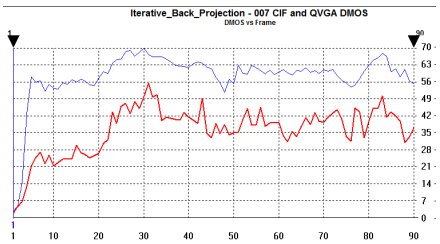


(a) Flower Video

(b) Foreman Video



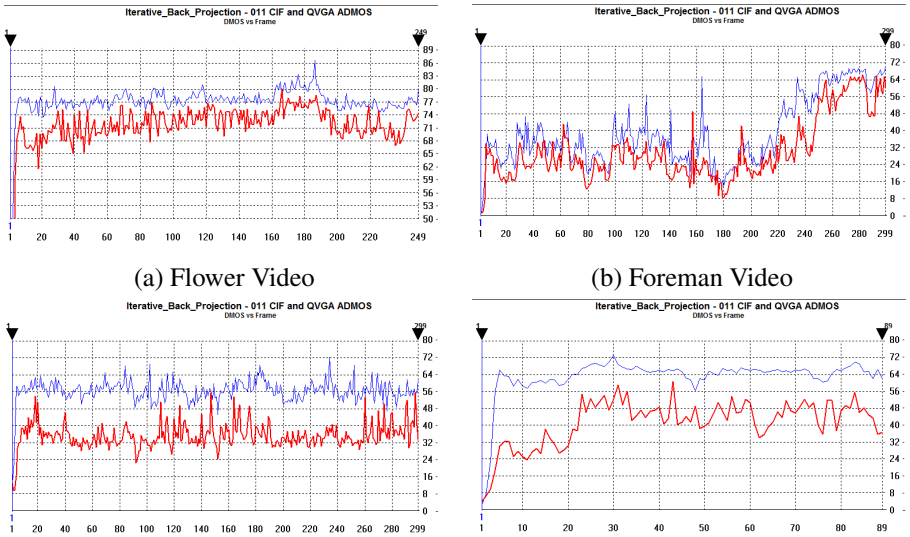
(c) News Video



(d) Stefan Video

Figure 4.6: DMOS Comparison of Iterative Back Projection and Bicubic Interpolation

The ADMOS results of the Iterative-Back Projection algorithm can be found in the figures 4.7.



(a) Flower Video

(b) Foreman Video

(c) News Video

(d) Stefan Video

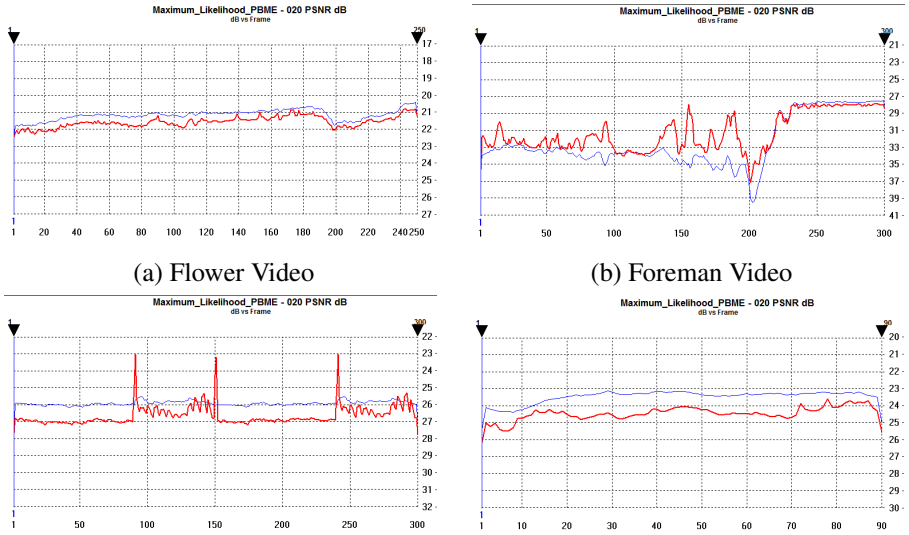
Figure 4.7: ADMOS Comparison of Iterative Back Projection and Bicubic Interpolation

4.1.3 Maximum-Likelihood Algorithm

This section will be introduced in two parts. Whereas the results of the first part belongs to the Maximum Likelihood algorithm which uses Projection-Based Motion Estimation to compensate the motion between consecutive frames, the results of the second part illustrates the Maximum Likelihood Algorithm which uses Block Matching Full Search Algorithm. For two different approaches, three bi-directional consecutive frames are used with four iterations. Search window size of both motion estimation algorithm is seventeen.

4.1.3.1 Maximum Likelihood with Projection-Based Motion Estimation

The PSNR results of the Maximum-Likelihood algorithm with Projection-Based Motion Estimation can be found in the figures 4.8.

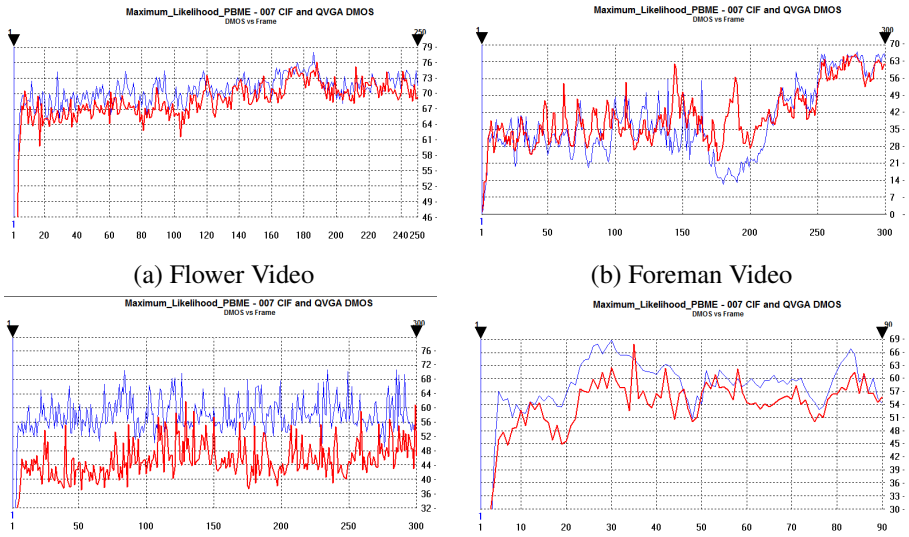


(a) Flower Video (b) Foreman Video

(c) News Video (d) Stefan Video

Figure 4.8: PSNR Comparison of Maximum Likelihood with Projection-Based Motion Estimation and Bicubic Interpolation

The DMOS results of the Maximum-Likelihood algorithm with Projection-Based Motion Estimation can be found in the figures 4.9.



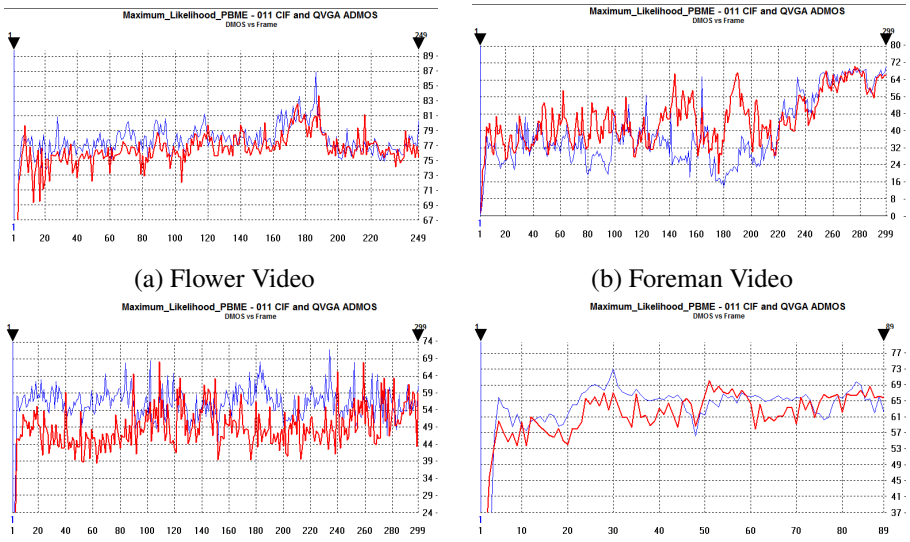
(a) Flower Video (b) Foreman Video

(c) News Video (d) Stefan Video

Figure 4.9: DMOS Comparison of Maximum Likelihood with Projection-Based Motion Estimation and Bicubic Interpolation

The ADMOS results of the Maximum-Likelihood algorithm with Projection-Based

Motion Estimation can be found in the figures 4.10.



(a) Flower Video

(b) Foreman Video

(c) News Video

(d) Stefan Video

Figure 4.10: ADMOS Comparison of Maximum Likelihood with Projection-Based Motion Estimation and Bicubic Interpolation

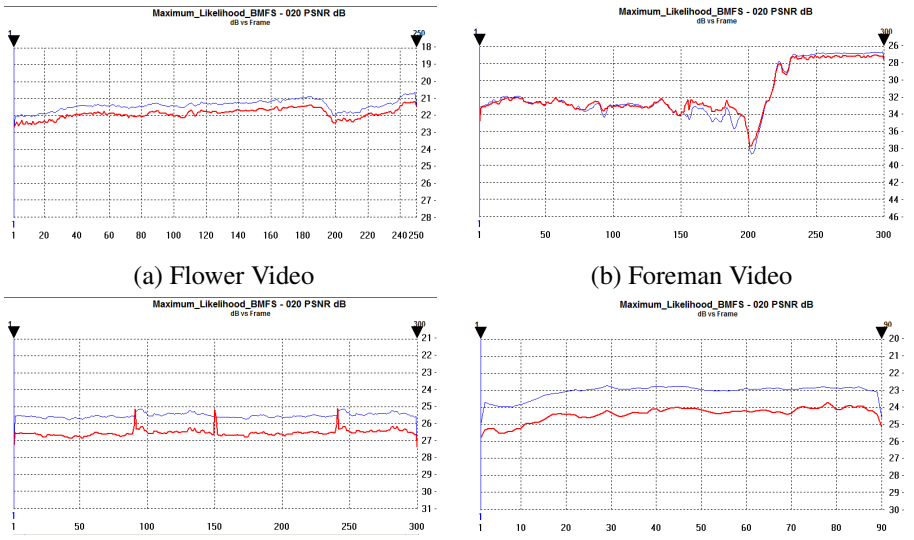
Maximum-Likelihood algorithm outgoes Bicubic Interpolation for the three videos when we consider PSNR, DMOS and ADMOS results except foreman video. Since it it takes blurring into account, it generates sharp images. Therefore, DMOS and ADMOS results are better when compared with the Bicubic Interpolation.

On the other hand, since foreman video has too much local motion in the beginning of the video, Maximum-Likelihood algorithm fails because of the performance of Projection-Based Motion Estimation algorithm. As it is shown in figure C.9, when there is local motion between consecutive frames, this algorithm generates disturbing motion blur. Nonetheless, when there is scene cuts in the video, this algorithm generates disturbing artifacts as it is shown in the figure C.13. It is because of the fact that this algorithm tries to minimize the equation 3.38 regardless of evaluating similarity between consecutive frames. Therefore, in case of scene cut, the output image is kind of averaged form of consecutive frames.

4.1.3.2 Maximum Likelihood with Block Matching Full Search Motion Estimation

When PSNR, DMOS and ADMOS results of foreman and news videos are considered, the effect of motion estimation on multi-frame super resolution algorithms can be understood easily. If there is dominant local motion between consecutive frames as it is shown in figure C.9, Maximum Likelihood with Projection-Based Motion Estimation algorithm fails because of the accuracy of Projection-Based Motion Estimation. However, Maximum-Likelihood with Block Matching Full Search gives better results for this condition.

The PSNR results of the Maximum-Likelihood algorithm with Block Matching Full Search Motion Estimation can be found in the figures 4.11.



(a) Flower Video

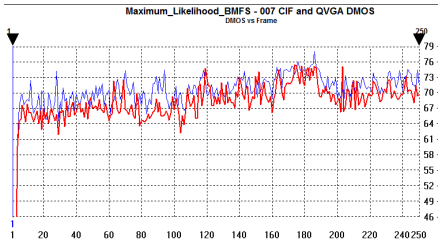
(b) Foreman Video

(c) News Video

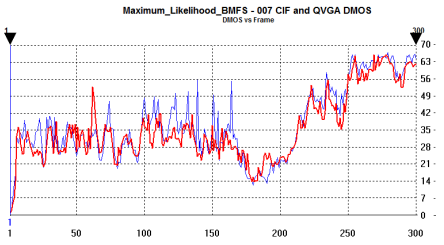
(d) Stefan Video

Figure 4.11: PSNR Comparison of Maximum Likelihood with Block Matching Full Search and Bicubic Interpolation

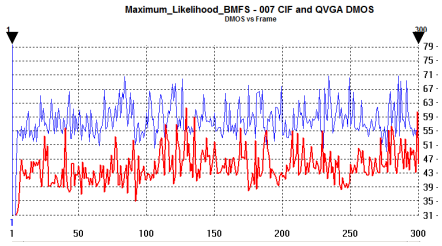
The DMOS results of the Maximum-Likelihood algorithm with Block Matching Full Search Motion Estimation can be found in the figures 4.12.



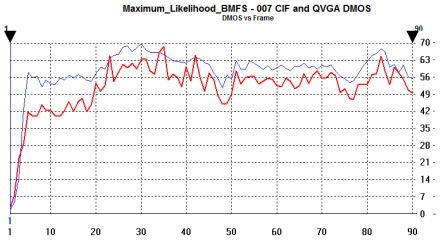
(a) Flower Video



(b) Foreman Video



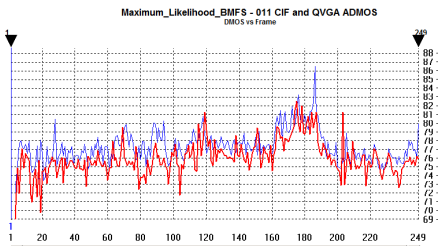
(c) News Video



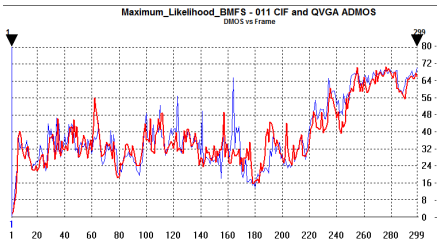
(d) Stefan Video

Figure 4.12: DMOS Comparison of Maximum Likelihood with Block Matching Full Search and Bicubic Interpolation

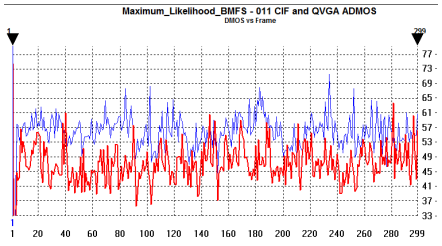
The ADMOS results of the Maximum-Likelihood algorithm with Block Matching Full Search Motion Estimation can be found in the figures 4.13.



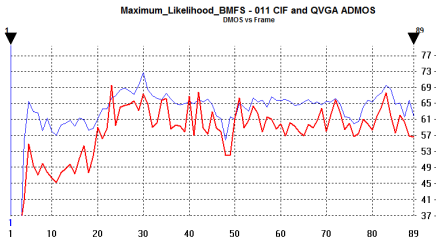
(a) Flower Video



(b) Foreman Video



(c) News Video



(d) Stefan Video

Figure 4.13: ADMOS Comparison of Maximum Likelihood with Block Matching Full Search and Bicubic Interpolation

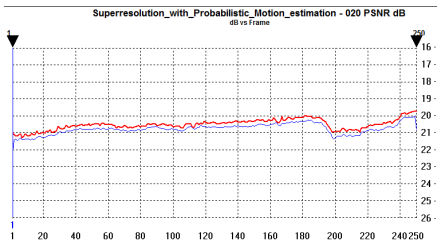
No matter which motion estimation algorithm is used, Maximum Likelihood algo-

rithm's performance decreases when scene cut occurs as it is shown in the figure C.13.

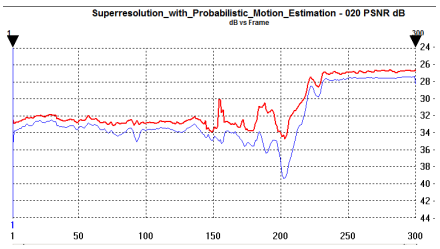
4.1.4 Super Resolution with Probabilistic Motion Estimation Algorithm

Super resolution with Probabilistic Motion Estimation algorithm falls behind the Bicubic Interpolation when PSNR, DMOS and ADMOS results are considered. Since the decimated videos are obtained using the equation 4.1 which includes blurring kernel, Super Resolution with Probabilistic Motion Estimation algorithm has also denoising characteristic since its algorithms is very similar with Bilateral Filter. Its results are worse than Bicubic Interpolation results, because output videos suffer from cartoonize effect as it is shown in figure C.9 and C.13. However, this algorithm is very resistant to scene cuts and local motion.

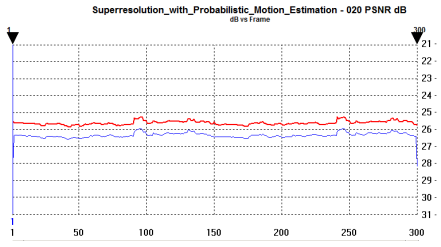
The PSNR results of the Super Resolution with Probabilistic Motion Estimation algorithm can be found in the figures 4.14. Three bi-directional consecutive frames are used in the experiments with two iterations. The search window size is selected as nine and the patch size is selected as three.



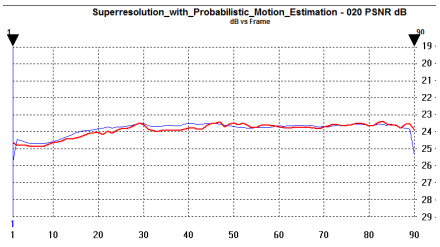
(a) Flower Video



(b) Foreman Video



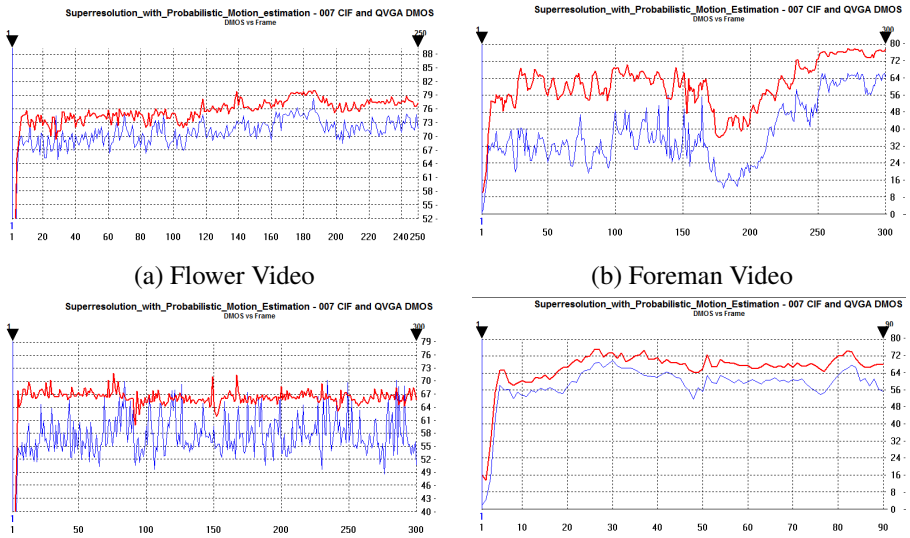
(c) News Video



(d) Stefan Video

Figure 4.14: PSNR Comparison of Super Resolution with Probabilistic Motion Estimation and Bicubic Interpolation

The DMOS results of Super Resolution with Probabilistic Motion Estimation algorithm can be found in the figures 4.15.



(a) Flower Video

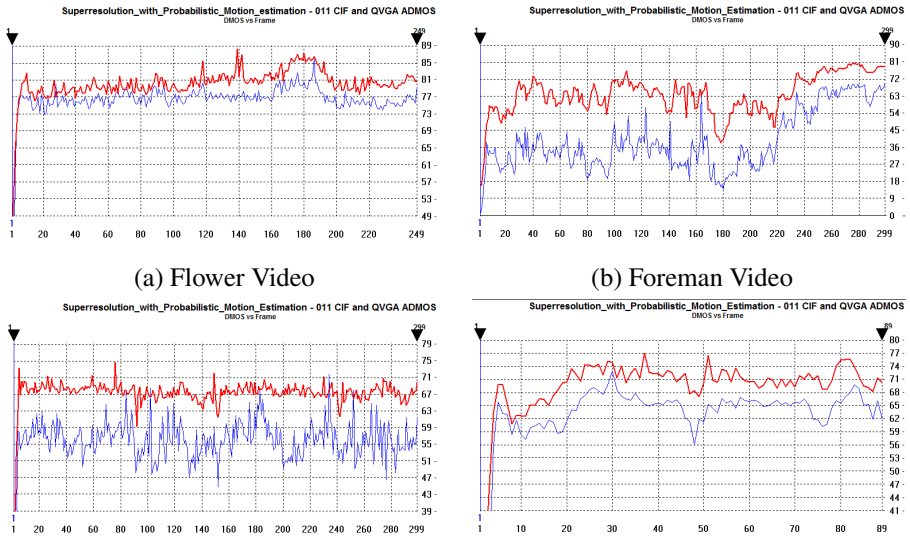
(b) Foreman Video

(c) News Video

(d) Stefan Video

Figure 4.15: DMOS Comparison of Super Resolution with Probabilistic Motion Estimation and Bicubic Interpolation

The ADMOS results of Super Resolution with Probabilistic Motion Estimation algorithm can be found in the figures 4.16.



(a) Flower Video

(b) Foreman Video

(c) News Video

(d) Stefan Video

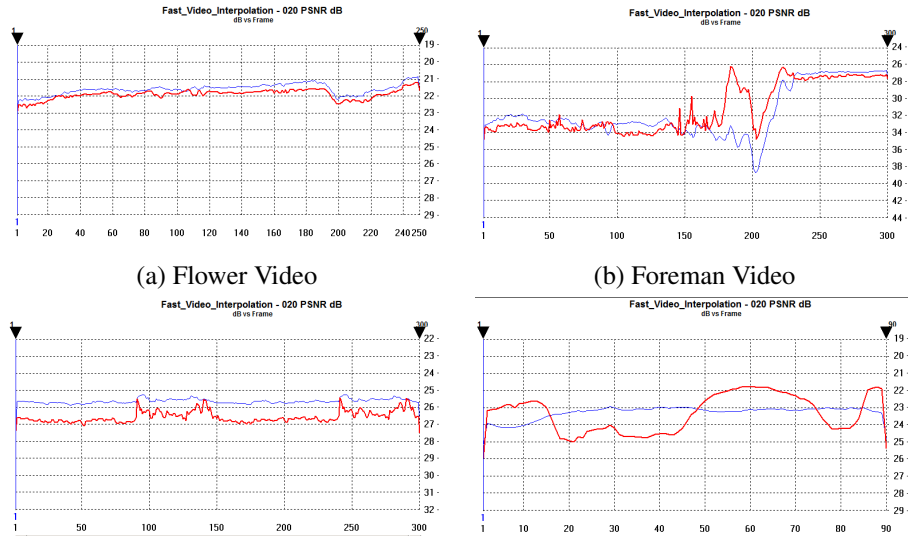
Figure 4.16: ADMOS Comparison of Super Resolution with Probabilistic Motion Estimation and Bicubic Interpolation

4.1.5 Fast Video Interpolation/Up-Sampling Using Linear Motion Model

This section will be introduced in three parts. Whereas first part shows thre result of Fast Video Interpolation without motion estimation technique, the remaining two parts illustrate the result of Maximum Likelihood algorithm which uses Projection-Based Motion Estimation and Block Matching Motion Estimation respectively. In the experiments, three bi-directional consecutive frames are used. The patch size of the algorithm is selected as three.

4.1.5.1 Fast Video Interpolation without Motion Estimation

The PSNR results of Fast Video Interpolation algorithm can be found in the figures 4.17.



(a) Flower Video

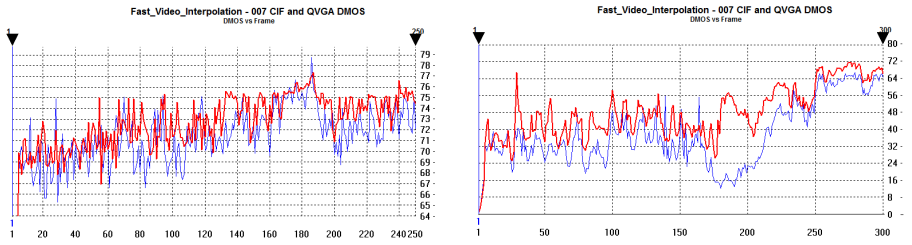
(b) Foreman Video

(c) News Video

(d) Stefan Video

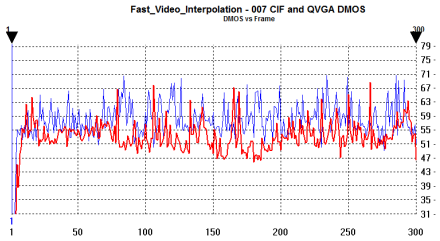
Figure 4.17: PSNR Comparison of Fast Video Interpolation without Motion Estimation and Bicubic Interpolation

The DMOS results of Fast Video Interpolation algorithm can be found in the figures 4.18.

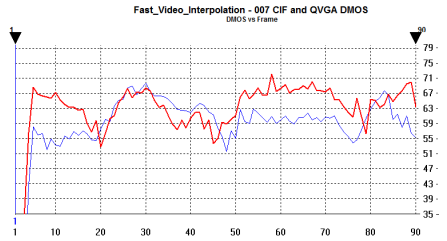


(a) Flower Video

(b) Foreman Video



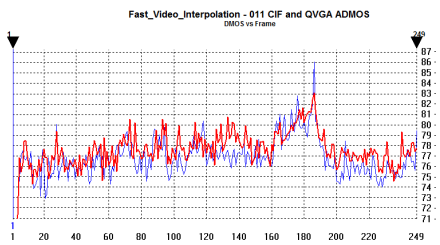
(c) News Video



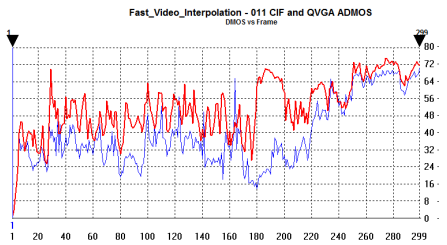
(d) Stefan Video

Figure 4.18: DMOS Comparison of Fast Video Interpolation without Motion Estimation and Bicubic Interpolation

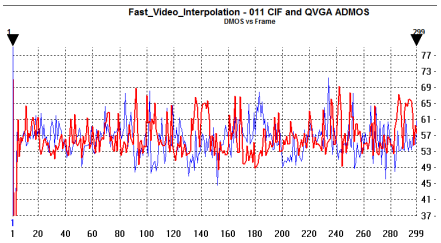
The ADMOS results of Fast Video Interpolation algorithm can be found in the figures 4.19.



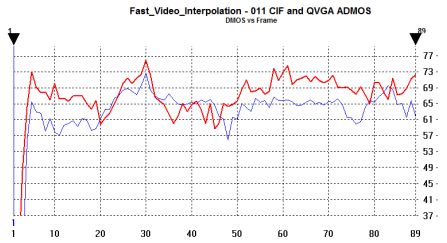
(a) Flower Video



(b) Foreman Video



(c) News Video



(d) Stefan Video

Figure 4.19: ADMOS Comparison of Fast Video Interpolation without Motion Estimation and Bicubic Interpolation

Fast Video Interpolation algorithm gives good result when the scene is stationary.

As it can be seen in PSNR and DMOS results of news video, it outgoes Bicubic Interpolation. However, if global or local motion exist between consecutive frames as it can be seen at foreman and stefan videos, the performance of the algorithm decreases. Even the algorithm claims that there is no need to register images, motion estimation algorithm is needed to eliminate the decrease of the performance.

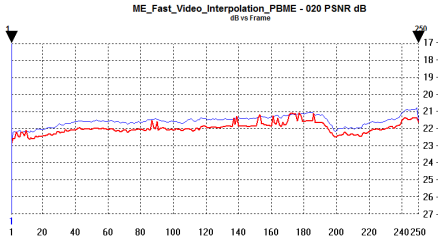
Nonetheless, if there is local motion between consecutive frames, this algorithm fails to estimate the missing pixels correctly as it can be seen on the hand of dancer in figure C.13. This is because of the fact that the weights shown in equation (3.43) are very small, therefore it is truncated to zero.

One of the main advantages of this algorithm is that it is resistant to scene cuts. When the scene is changed, this algorithm doesn't generate disturbing artifacts like motion blur as it can be seen in figure C.13. The other advantage of this algorithm is that although Maximum Likelihood algorithm generates jagged edges, this algorithm generate very straight edges as it can be seen in the figures in Appendix C.

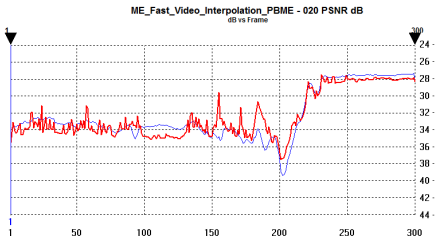
4.1.5.2 Fast Video Interpolation with Projection Based Motion Estimation

As it can be seen at the stefan video which has global motion, Fast Video Interpolation with Projection-Based Motion Estimation algorithm outgoes Fast Video Interpolation algorithm without motion estimation. However, when there is local motion between consecutive frames as it is shown in foreman video, this algorithm fails because of the fact that the weights shown in equation (3.43) are very small, therefore it is truncated to zero.

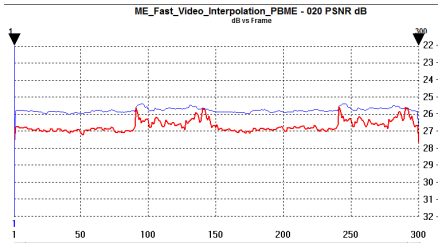
The PSNR results of Fast Video Interpolation algorithm with Projection Based Motion Estimation can be found in the figures 4.20.



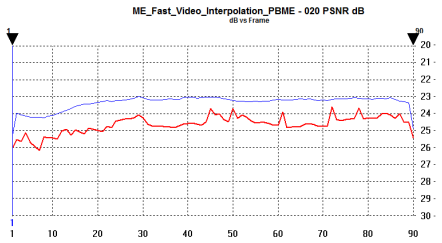
(a) Flower Video



(b) Foreman Video



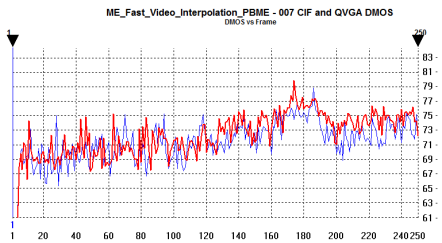
(c) News Video



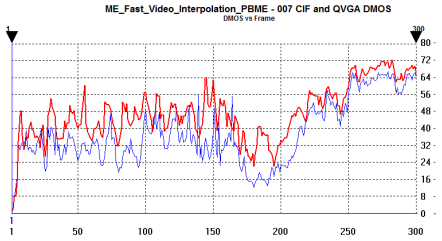
(d) Stefan Video

Figure 4.20: PSNR Comparison of Fast Video Interpolation with Projection-Based Motion Estimation and Bicubic Interpolation

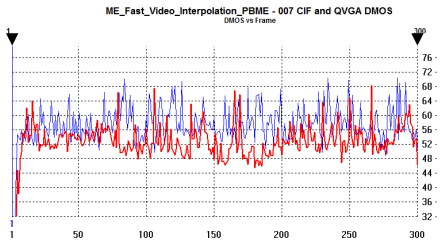
The DMOS results of Fast Video Interpolation algorithm with Projection Based Motion Estimation can be found in the figures 4.21.



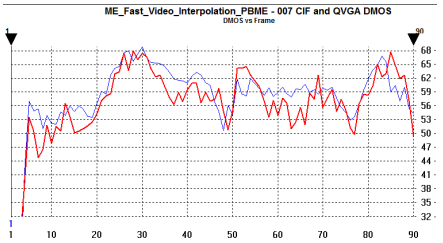
(a) Flower Video



(b) Foreman Video



(c) News Video

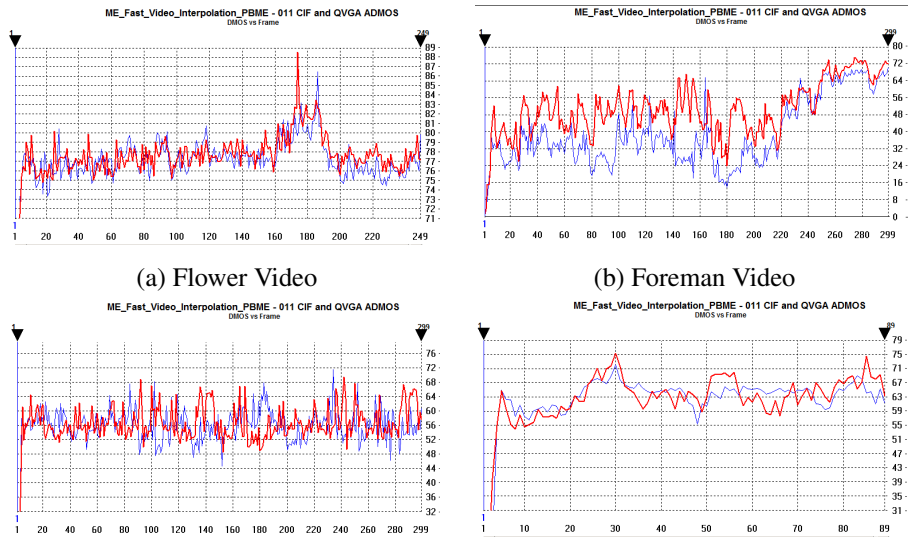


(d) Stefan Video

Figure 4.21: DMOS Comparison of Fast Video Interpolation with Projection-Based Motion Estimation and Bicubic Interpolation

The ADMOS results of Fast Video Interpolation algorithm with Projection Based

Motion Estimation can be found in the figures 4.22.



(a) Flower Video

(b) Foreman Video

(c) News Video

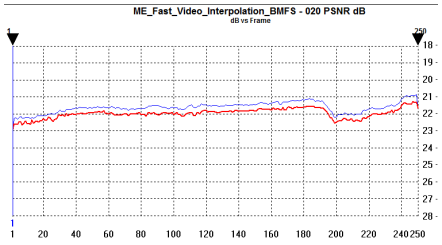
(d) Stefan Video

Figure 4.22: ADMOS Comparison of Fast Video Interpolation with Projection-Based Motion Estimation and Bicubic Interpolation

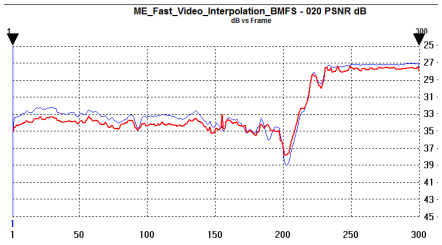
4.1.5.3 Fast Video Interpolation with Block Matching Full Search Motion Estimation

As it can be seen PSNR graph of foreman and news video; where there is local motion between consecutive frames, Fast Video Interpolation with Block Matching Full Search algorithm outgoes Fast Video Interpolation with Projection-Based Motion Estimation. The artifacts which are seen in the Fast Video Interpolation with Projection-Based Motion Estimation cannot be seen in this algorithm. It is because of the fact that all consecutive frames are properly aligned; therefore, the weights which is seen in equation (3.43) is estimated properly. However, all the Fast Video Interpolation algorithm gives worse result because of the nature of the algorithm. This algorithm is also very similar with Bilateral filter. The denoising characteristic of this video cause blur output images therefore DMOS and ADMOS result are worse than Bicubic Interpolation.

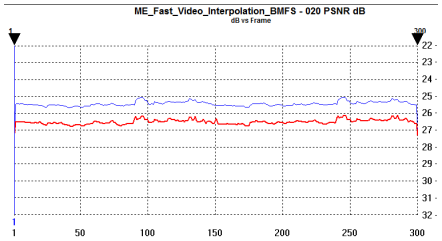
The PSNR results of Fast Video Interpolation algorithm with Block Matching Full Search Motion Estimation can be found in the figures 4.23.



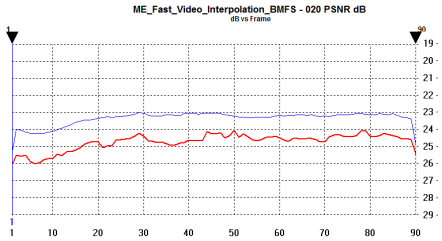
(a) Flower Video



(b) Foreman Video



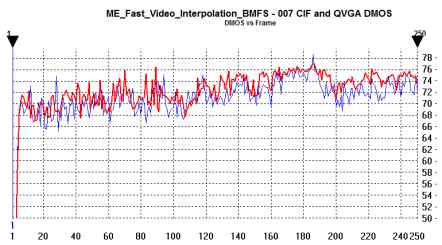
(c) News Video



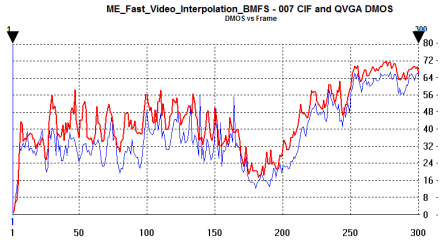
(d) Stefan Video

Figure 4.23: PSNR Comparison of Fast Video Interpolation with Block Matching Full Search and Bicubic Interpolation

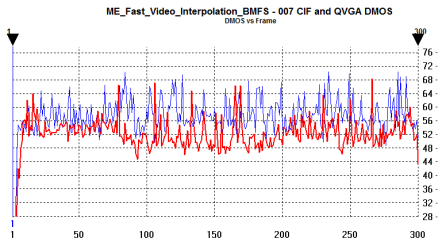
The DMOS results of Fast Video Interpolation algorithm with Block Matching Full Search Motion Estimation can be found in the figures 4.24.



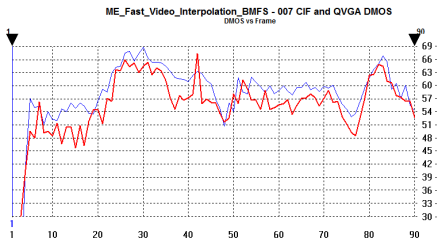
(a) Flower Video



(b) Foreman Video



(c) News Video

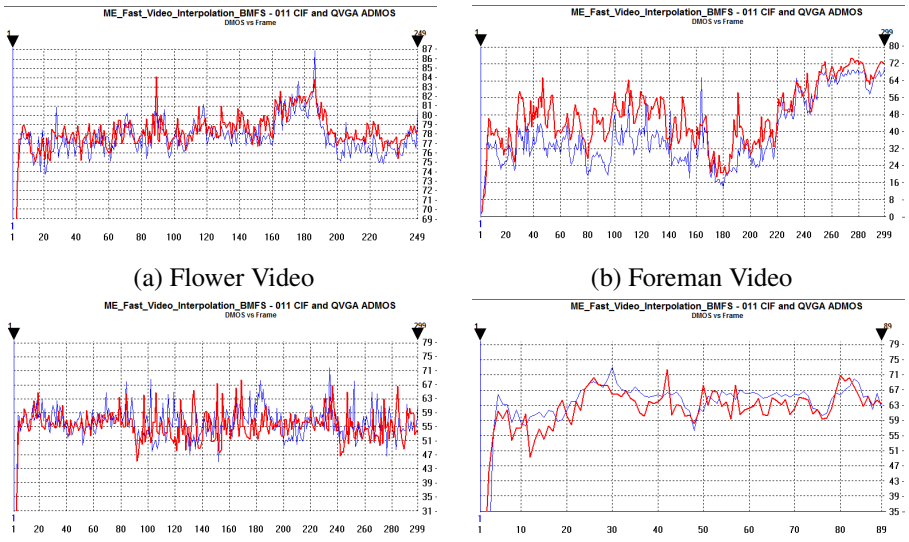


(d) Stefan Video

Figure 4.24: DMOS Comparison of Fast Video Interpolation with Block Matching Full Search and Bicubic Interpolation

The ADMOS results of Fast Video Interpolation algorithm with Block Matching Full

Search Motion Estimation can be found in the figures 4.25.



(a) Flower Video

(b) Foreman Video

(c) News Video

(d) Stefan Video

Figure 4.25: ADMOS Comparison of Fast Video Interpolation with Block Matching Full Search and Bicubic Interpolation

4.2 Comparison of Super Resolution Algorithms

In this section, performance-wise and complexity-wise results of analyzed algorithms will be presented. The performance-wise results of the analyzed algorithms are shown in tables 4.1, 4.2, 4.3 and 4.4.

Table 4.1: Comparison Table of Super Resolution Algorithms for Flower Video

	PSNR(dB)			DMOS			ADMOS		
	min	mean	max	min	mean	max	min	mean	max
Bicubic Interpolation	20,86	21,63	22,93	13,48	72,10	78,94	28,55	78,06	87,00
Fast Edge-Adaptive Interpolation	21,26	21,94	22,93	13,48	73,13	78,52	31,40	78,70	86,74
Iterative-Back Projection	21,68	22,87	23,57	13,48	65,43	74,54	25,58	73,50	80,30
Maximum Likelihood with PBME	21,28	22,04	22,93	13,48	70,20	77,06	29,61	77,00	83,77
Maximum Likelihood with BMFS	21,38	22,11	22,93	13,48	69,93	76,39	26,97	76,85	82,96
Super Resolution with PME	20,59	21,37	22,17	14,97	76,80	80,67	29,13	81,81	89,20
Fast Video Interpolation	21,24	21,94	22,93	13,48	73,53	77,57	31,40	78,73	84,01
Fast Video Interpolation with PBME	21,15	22,04	22,93	13,48	73,43	80,01	30,75	78,57	89,06
Fast Video Interpolation with BMFS	21,30	21,98	22,93	13,48	73,51	77,35	30,98	78,63	84,19

Table 4.2: Comparison Table of Super Resolution Algorithms for Foreman Video

	PSNR(dB)			DMOS			ADMOS		
	min	mean	max	min	mean	max	min	mean	max
Bicubic Interpolation	27,59	31,49	39,64	0,57	48,11	66,96	0,88	52,30	70,28
Fast Edge-Adaptive Interpolation	28,07	32,25	40,45	0,57	46,26	66,57	0,87	51,99	71,66
Iterative-Back Projection	28,54	32,51	40,05	0,57	35,81	60,77	0,88	41,61	66,18
Maximum Likelihood with PBME	27,85	30,86	37,26	0,57	47,31	66,15	0,88	53,20	70,34
Maximum Likelihood with BMFS	27,96	31,61	38,72	0,57	45,26	66,08	0,88	49,84	70,58
Super Resolution with PME	26,80	30,42	34,95	9,89	68,18	78,02	15,79	70,96	80,69
Fast Video Interpolation	27,09	31,22	35,82	0,57	56,75	71,62	0,88	60,72	75,01
Fast Video Interpolation with PBME	27,74	31,69	37,79	0,57	56,50	72,22	0,88	59,44	75,10
Fast Video Interpolation with BMFS	27,91	31,99	38,45	0,57	55,98	71,69	0,88	58,68	74,71

Table 4.3: Comparison Table of Super Resolution Algorithms for News Video

	PSNR(dB)			DMOS			ADMOS		
	min	mean	max	min	mean	max	min	mean	max
Bicubic Interpolation	26,02	26,42	28,27	11,76	59,67	71,02	12,76	57,56	72,17
Fast Edge-Adaptive Interpolation	27,15	27,48	28,27	11,37	48,04	67,84	12,76	50,36	66,17
Iterative-Back Projection	28,15	28,63	28,96	6,47	35,31	59,32	9,43	36,47	55,56
Maximum Likelihood with PBME	23,51	27,09	28,27	11,69	46,82	62,20	17,36	51,19	68,57
Maximum Likelihood with BMFS	26,01	27,43	28,27	9,65	46,1	61,85	12,19	49,55	75,22
Super Resolution with PME	25,33	25,69	25,92	19,53	67,19	72,57	28,88	68,18	75,12
Fast Video Interpolation	26,20	27,33	28,27	11,76	55,01	68,93	15,19	58,58	71,92
Fast Video Interpolation with PBME	26,20	27,33	28,27	11,76	55,01	68,93	15,19	58,41	70,07
Fast Video Interpolation with BMFS	27,09	27,48	28,27	11,76	54,26	68,94	16,13	56,59	68,76

Table 4.4: Comparison Table of Super Resolution Algorithms for Stefan Video

	PSNR(dB)			DMOS			ADMOS		
	min	mean	max	min	mean	max	min	mean	max
Bicubic Interpolation	23,51	23,86	25,78	1,65	61,16	69,77	2,29	65,40	73,08
Fast Edge-Adaptive Interpolation	24,55	25,05	26,56	2,51	58,82	69,10	3,61	63,73	74,66
Iterative-Back Projection	25,97	26,58	27,86	2,51	38,78	55,29	3,61	45,09	60,65
Maximum Likelihood with PBME	24,00	24,84	26,56	2,51	56,59	68,85	3,61	63,56	70,20
Maximum Likelihood with BMFS	24,51	25,18	26,56	2,51	55,53	68,75	4,20	60,37	69,92
Super Resolution with PME	23,43	23,91	24,93	13,39	68,45	75,16	19,22	71,22	77,14
Fast Video Interpolation	22,32	23,81	26,56	2,51	65,42	72,08	3,61	68,72	76,31
Fast Video Interpolation with PBME	24,13	25,14	26,71	2,51	59,88	68,88	3,61	66,34	76,31
Fast Video Interpolation with BMFS	24,59	25,22	26,56	2,51	58,39	68,27	7,03	64,23	72,37

The interpretations of the tables 4.1, 4.2, 4.3 and 4.4 are as follows.

- Accuracy of the motion estimation affects the performance of the multi-frame super resolution algorithms.
- Iterative Back Projection algorithm is the outgoing algorithms since it is a single frame algorithm which takes blurring into account.
- Fast Edge-Adaptive algorithm is one of the best algorithm which gives higher

quality images when compared with Bicubic Interpolation unless video has texture region.

- When Maximum Likelihood and Fast Video Interpolation algorithm are compared, it is seen that while Fast Video Interpolation outgoes Maximum Likelihood algorithm for PSNR results, Maximum Likelihood algorithm outgoes Fast Video Interpolation algorithm for DMOS and ADMOS results. It is because of the fact that while Maximum Likelihood algorithm applies deblurring to output image. Therefore, DMOS and ADMOS results Maximum Likelihood algorithm is higher than Fast Video Interpolation.

The resolutions in the tables 4.5 and 4.6 indicates output resolutions. Scale factor is two for all the experiments.

Table 4.5: Execution time of super resolution algorithms for different resolutions on INTEL architecture whose details are listed in Appendix D

	CIF	SD	HD	FHD	UHD
Bicubic Interpolation Native	35 msec	148 msec	353 msec	799 msec	3223 msec
Bicubic Interpolation Opencv	1 msec	1 msec	2 msec	5 msec	18 msec
Fast Edge-Adaptive Interpolation	7 msec	31 msec	73 msec	161 msec	656 msec
Iterative-Back Projection	259 msec	1031 msec	2846 msec	6451 msec	21357 msec
Maximum Likelihood with PBME	256 msec	994 msec	2664 msec	6002 msec	20651 msec
Maximum Likelihood with BMFS	542 msec	2427 msec	8983 msec	20730 msec	54194 msec
Super Resolution with PME	17275 msec	76706	182821	411213	NA
Fast Video Interpolation	1161 msec	4859 msec	11507 msec	26020 msec	105541 msec
Fast Video Interpolation with PBME	1161 msec	4881 msec	12052 msec	26219 msec	105843 msec
Fast Video Interpolation with BMFS	1408 msec	6191 msec	17434 msec	39922 msec	135337 msec

Table 4.6: Execution time of super resolution algorithms for different resolutions on ARM architecture whose details are listed in Appendix D

	CIF	SD	HD	FHD	UHD
Bicubic Interpolation Native	750 msec	3140 msec	7670 msec	16740 msec	66970 msec
Bicubic Interpolation Opencv	10 msec	20 msec	40 msec	80 msec	270 msec
Fast Edge-Adaptive Interpolation	60 msec	270 msec	610 msec	1400 msec	5510 msec
Iterative-Back Projection	2680 msec	10880 msec	29360 msec	66420 msec	223780 msec
Maximum Likelihood with PBME	2450 msec	9840 msec	25990 msec	58740 msec	201940 msec
Maximum Likelihood with BMFS	4900 msec	21830 msec	78290 msec	180280 msec	480340 msec
Super Resolution with PME	481230 msec	NA	NA	NA	NA
Fast Video Interpolation	12040 msec	51260 msec	118510 msec	270220 msec	1087590 msec
Fast Video Interpolation with PBME	11150 msec	47210 msec	117990 msec	273730 msec	1096820 msec
Fast Video Interpolation with BMFS	14280 msec	62690 msec	171570 msec	398390 msec	1383840 msec

The interpretations of the tables 4.5 and 4.6 are as follows.

- Multi-frame super resolution algorithms needs more computational power when compared with single frame super resolution algorithms. This computational cost is the consequence of the motion between consecutive frames and necessity of processing many input frames to obtain one output frame. Unless a powerful dedicated device or advanced GPU with many cores are used, multi-frame algorithms are not suitable for real time implementation.

- There is a huge difference in Bicubic Interpolation between OPENCV implementation and the native C++ implementation. It is because of the fact that OPENCV uses SSE, SSE2, SSE3 and SSSE3 for INTEL architecture and NEON for ARM architecture. If the algorithm can be made vectorize, these API's make the implementation fast enough to be applicable to real time. However, since super resolution algorithm cannot be made vectorize, it is hard to use these API's. Moreover, since these API's are platform specific, it is very hard to implement all the algorithms for INTEL Platform and ARM platform separately.

4.3 Discussion

In this chapter, PSNR, DMOS and ADMOS of analyzed algorithms are presented. The results show that motion estimation to detect local motion is a must in order to obtain artifact-free high resolution output image even if modern super resolution algorithms are used. Since computational load of estimating local motion is high, it is not proper for real time implementation. When execution time and performance of the algorithms are considered, Fast Edge Adaptive Algorithm is the most suitable algorithm to be implemented for real time.

CHAPTER 5

PARALLEL PROGRAMMING

Since super resolution algorithms' computational complexity is too high, parallel programming is needed to speed up the algorithms. In order to run the instructions in parallel for multiple data, there are many APIs to accomplish this task. If the application is running on INTEL processor, SSE can be used; if the application is running on ARM processor, the algorithm can be speeded up via NEON if the algorithms can be made vectorize. If the platform has NVIDIA GPU; CUDA API can be used to speed up the algorithm even if the algorithm cannot be made vectorize. The only prerequisite to implement the algorithm in CUDA is that algorithm should be suitable for parallel programming. In other words, there should be a loop which has no interdependency between different iterations. In case of absence of any advanced GPU in the platform, programmer should learn Verilog or VHDL to run the computational costly algorithms on FPGA.

Since there are too many APIs, which depend on architecture, to speed up the algorithms; it is very hard to manage the software and adopt new modifications to program. In order to eliminate this difficulty, OPENCL was released in 2008 by Khronos Group, an independent standards consortium which consists of many companies such as 3Dlabs, ATI, Discreet, Evans & Sutherland, Intel, NVIDIA, SGI and Sun Microsystems. OPENCL blends them to create a hardware independent software development environment. It supports single or multiple device systems consisting of CPUs,GPUs, FPGA and potentially other future devices. OpenCL provides parallel computing using task-based and data-based parallelism. It currently supports CPUs that include x86 and ARM, and it has been adopted into graphics

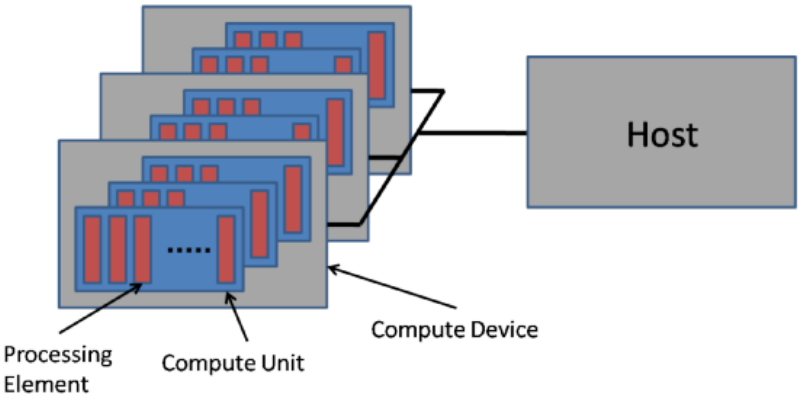


Figure 5.1: OPENCL Platform Model [16]

card drivers by AMD, Apple, Intel, and NVIDIA.[15][18] Also, Altera and ARM has released OPENCL SDK to be used in their FPGAs and Graphic Processing Unit respectively.[17][19]

In this chapter, first OPENCL will be introduced briefly. After that, OPENCL support of mmrgLibrary, whose details can be found in Appendix B, will be explained.

5.1 Introduction to OPENCL

OPENCL platform model is defined as a host and connected OPENCL supported devices as it is shown in figure 5.1.

While host can be any CPU such as ARM or X86; OPENCL devices can be GPU, DSP, FPGA or a multi-core CPU. An OPENCL device consists of a collection of one or more compute units (cores). A compute units or cores is composed of one or more processing elements (threads). Processing elements execute the instructions specified in the OPENCL kernel as Single Instruction Multiple data principle. In other words, every threads are executing same code blocks in parallel. Since compute units typically contain many more processing elements than application processors (host), they can compute at a much higher rate than application processors.

Host device controls the flow of the program. Every intensive data processing task will be directed to compute device by copying data to memory of compute device

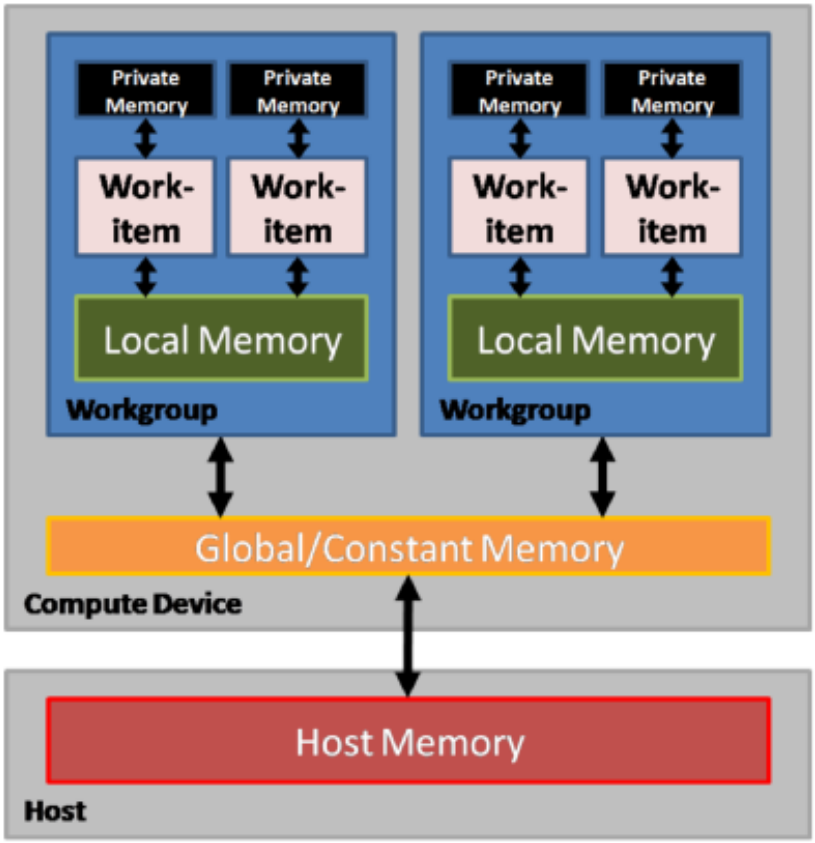


Figure 5.2: OPENCL Memory Model [16]

from memory of host device. The copied data will be processed in parallel by many threads which are created by compute unit. The processed data will be copied back to host memory again. The figure 5.2 shows the regions of the accessible memory by host and compute devices.

Host device determines the instructions of the compute device by introducing kernel. Kernels are the basic unit of executable code that runs on one or more OpenCL devices. Kernels are similar to a C function that can be data- or task-parallel. Every processing element execute the same kernel by using different data. The corresponding data is determined by ID of the processing element. The simple example can be found in figure 5.3.

In figure 5.3, the Scalar C function runs in CPU serially. In other words, result[n] will be calculated after result[n-1] is calculated. On the contrary, result[n], result[n-

Scalar C Function	Data-Parallel Function

`void square(int n, const float *a, float *result)`
`{`
`  int i;`
`  for (i=0; i<n; i++)`
`  result[i] = a[i]*a[i];`
`}`
	`kernel void dp_square`
	`  (global const float *a, global float *result)`
	`{`
	`  int id= get_global_id(0);`
	`  result[id] = a[id]*a[id];`
	`}`
	`// dp_square execute over "n" work-items`

Figure 5.3: Simple Example of Scalar Versus Parallel Implementation [16]

1]..., result[0] will be calculated in parallel in the Data-Parallel function by the aid of the parallel architecture. Therefore, the super resolution algorithms which has loop without interdependencies between iterations can be implemented by parallel programming.

5.2 mmrgLibrary OPENCL Support

In order to use OPENCL support of mmrgLibrary, the driver of the OPENCL device should be installed to the computer. Since INTEL processor whose details are listed in Appendix D, INTEL SDK for OPENCL was installed to the computer. INTEL SDK is using INTEL HD Graphics 5500 as compute device.

OPENCL usage of mmrgLibrary can be controlled via the configuration file as it is shown in the figure B.2. OPENCL support is enabled by setting OPENCL variable to "USED" or OPENCL support is disabled by setting OPENCL variable to "NOT_USED".

All the OPENCL kernels are located at the **ocl_kernels** directory under **src** path of mmrgLibrary. For now, there are only two kernel files which are used in OPENCL implementation of Fast Edge-Adaptive Interpolation.

As it is shown in figure 5.4, OPENCL implementation of Fast Edge-Adaptive Interpolation is called by passing OPENCL enumeration as input parameter.

When figure B.6 and 5.4 are compared, the only difference between these two figures is the parameter passing as function argument. Whenever the OPENCL function is called from the application layer, EN_MMRG_USED_API_OPENCL enumeration must be used.

```
void ResizeImageExampleOpencl()
{
    int nInputImageWidth = 176;
    int nInputImageHeight = 144;
    int nInputImageNumberofChannel = 3;
    mmrgPixelValue nInputImagePixelValue(255,255,255);
    mmrgImage cInputImage(nInputImageWidth,nInputImageHeight,nInputImageNumberofChannel,nInputImagePixelValue);


    int nOutputImageWidth = 352;
    int nOutputImageHeight = 288;
    int nOutputImageNumberofChannel = 3;
    mmrgImage cOutputImage(nOutputImageWidth,nOutputImageHeight,nOutputImageNumberofChannel);

    mmrgResize::GetInstance(EN_MMRG_RESIZE_TYPE_EDGE_ADAPTIVE_INTERPOLATION, EN_MMRG_USED_API_OPENCL)->imResize(cInputImage, cOutputImage);

}
```

Figure 5.4: mmrgLibrary Resize Image Example by using OPENCL implementation of Fast Edge-Adaptive Interpolation

Table 5.1: Execution time of Serial and Parallel Fast Edge-Adaptive Interpolation for different resolutions on INTEL architecture whose details are listed in Appendix D

	CIF	SD	HD	FHD	UHD
Fast Edge-Adaptive Interpolation Serial	7 msec	31 msec	73 msec	161 msec	656 msec
Fast Edge-Adaptive Interpolation Parallel	2 msec	9 msec	21 msec	47 msec	189 msec

Execution time comparison of serial and parallel implementation is shown in table 5.1.

As it can be seen in table 5.1, parallel implementation is four times faster than the serial implementation.

CHAPTER 6

CONCLUSION AND FUTURE WORKS

6.1 Conclusion

In the light of the information given in this thesis, it can be deduced that DMOS and ADMOS are better quality metrics to simulate the human visual system when they are compared with PSNR. Since DMOS and ADMOS consider characteristic of the input video, it should be preferred to compare super resolution algorithms in order to obtain more reliable results.

The accuracy of motion estimation algorithms is very critical for the performance of multi-frame super resolution algorithms. The motion compensation algorithms which will be used in the registration part of multi-frame algorithms should compensate all the motion in order to obtain high picture quality. Even if the modern super resolution algorithms penalizes the outliers between consecutive frames, it is observed that the quality of the output image suffers from various artifacts if the accuracy of motion estimation is not good enough.

Although modern super resolution algorithms produce artifact-free edges, the output image suffers from blurriness since these algorithms originate from Bilateral filter. Although Bilateral filter is edge-preserving filter, it loses the high frequency components since it is a de-noising filter. Nevertheless, the complexity of these algorithms are so high that it is hard to implement for a real time implementation. On the other hand, Maximum Likelihood algorithm produces high picture quality output if Block Matching Full Search Motion Estimation algorithm is used. However, since Block Matching Full Search Motion Estimation algorithm's complexity is too high,

it is hard to implement for real time implementation. Nonetheless, since Maximum Likelihood algorithm does not penalize the outliers between consecutive frames, the output suffers from the motion and scene-cuts artifacts.

Fast Edge Adaptive Interpolation is the best application if performance/complexity ratios of the algorithms are considered. It is very fast algorithm and it can be applicable to parallel programming. The performance is one of the best among all the analyzed algorithms. Therefore, this algorithm is selected to be implemented on Arçelik Linux Television. However, since the algorithm complexity is too high to give FHD and UHD output, parallel programming is needed to implement the algorithm in real time. In order to accomplish this task, OPENCL API was used to speed up the algorithm on INTEL on-board GPU as a first step. In future, embedded parallel implementation of this algorithm will be implemented.

6.2 Future Works

As a future plan, OPENCL will be used on our Arcelik Televisions. Our televisions have MALI 760 GPU which supports OPENCL and OPENGL. Fast Edge-Adaptive Algorithm and other new algorithms will be implemented on television environment. In parallel with this task, Altera FPGA will be bought and all the super resolution algorithms which are suitable for parallel implementation including Block Matching Motion Estimation are implemented by using OPENCL SDK of ALTERA.

REFERENCES

[1] *Objective Measurements and Subjective Assessments*, Application Note of Tektronix

[2] *Understanding PQR, DMOS, and PSNR Measurements,* Application Note of Tektronix

[3] H. G. Musmann, P. Pirsch, J. Grallert, *Advances in picture coding,* Proc. IEEE, vol. 73, no. 4, pp. 523-548, April 1985

[4] S. Alliney, C. Morandi, *Digital Image Registration using Projections,* IEEE Trans. On Pattern Analysis and Machine Intelligence, Vol PAMI-8 No 2, March 1986

[5] C. Tomasi and R. Manduchi, *Bilateral filtering for gray and color images,* in Proc. of the sixth international Conference of Computer Vision, pp. 839-846, 1998.

[6] Xiong Changzhen, Chen Licong, Pang Yigui, *An Adaptive Bilateral Filtering Algorithm and Its Application in Edge Detection,* International Conference on Measuring Technology and Mechatronics Automation, 2010.

[7] Lin Sun, Oscar C. Au, Ruobing Zou, Wei Dai, Xing Wen, Sijin Li and Jiali Li, *Adaptive Bilateral Filter Considering Local Characteristics,* Sixth International Conference on Image and Graphics, 2011

[8] R. G. Keys, *Cubic convolution interpolation for digital image processing,* IEEE Trans. On Acoustics, Speech and Signal Processing, vol. 29, no. 6, pp. 1153-1160, December 1981

[9] Mei-Juan Chen , Chin-Hui Huang , Wen-Li Lee, *A fast edge-oriented algorithm for image interpolation,* Image and Vision Computing, v.23 n.9, p.791-798, September, 2005

[10] Michal Irani, Shmuel Peleg, *Improving Resolution by Image Registration,* CVGIP, Graphical Models and Image Processing Vol 53 no. 3 pp 231-239 May 1991

[11] M. Elad, S. Farsiu, D. Robinson and P. Milanfar, *Robust shift and add approach to superresolution,* in Proc. SPIE Conf. Applications of Digital Signal and Image Processing, pp. 121-130, August 2003

[12] M. Protter, M. Elad, *Superresolution with probabilistic motion estimation,* IEEE Trans. On Image Processing, vol.18, no. 8, pp. 1899 – 1904, August 2009

[13] Kwok-Wai Hung and Wan-Chi Siu, *Fast video interpolation/upsampling using linear motion model,* in Proc. IEEE Int. Conf. Image Processing (ICIP 2011), pp. 1341-1344, September 2011

[14] Adrian Kaehler, Gary Bradski, *Learning OpenCV: Computer Vision in C++ with the OpenCV Library*

[15] *Intel SDK for OpenCL Applications 2015 Release Note*

[16] Aaftab Munshi, Benedict R. Gaster, Timothy G. Mattson, James Fung, Dan Ginsburg, *OpenCL Programming Guide*

[17] *Altera SDK for OpenCL Programming Guide*

[18] *AMD Training Guide : Introduction to OpenCL Programming*

[19] *Mali -T600 Series GPU OpenCL Developer Guide*

APPENDIX A

ALGORITHMS

Algorithm 1 Fast Edge-Adaptive Interpolation

1: **procedure** IMRESIZE(LR, HR)
2: $Scale\_x \leftarrow LR\_Width/HR\_Width$
3: $Scale\_y \leftarrow LR\_Height/HR\_Height$
4: **while** $HR\_y < HR\_Height$ **do**
5: $LR\_y \leftarrow HR\_y \times Scale\_y$
6: **while** $HR\_x < HR\_Width$ **do**
7: $LR\_x \leftarrow HR\_x \times Scale\_x$
8: $CalculateMissingPixel(HR, HR\_x, HR\_y, LR, LR\_x, LR\_y)$
9: $HR\_x \leftarrow HR\_x + 1$
10: **end while**
11: $HR\_y \leftarrow HR\_y + 1$
12: **end while**
13: $HR\_y, HR\_x, Diff1, Diff2, Diff3, Diff4 \leftarrow 0$
14: **while** $HR\_y < HR\_Height$ **do**
15: **while** $HR\_x < HR\_Width$ **do**
16: **if** $EDGE\_MAP(HR\_x, HR\_y)$ is $true$ **then**
17: $CalculateEdgePixel(HR, HR\_x, HR\_y, EDGE\_MAP)$
18: **end if**
19: $HR\_x \leftarrow HR\_x + 1$
20: **end while**
21: $HR\_y \leftarrow HR\_y + 1$
22: **end while**
23: **end procedure**

Algorithm 2 CalculateMissingPixel Method

procedure CALCULATEMISSINGPIXEL(HR,HR_x,HR_y,LR,LR_x,LR_y)
  if HR_x is even $\&$ HR_y is even **then**
    $HR(HR\_x, HR\_y) \leftarrow LR(LR\_x, LR\_y)$
  else if HR_x is odd $\&$ HR_y is even **then**
    if $|LR(LR\_x + 1, LR\_y) - LR(LR\_x, LR\_y)| < Threshold$ **then**
      $HR(HR\_x, HR\_y) \leftarrow (LR(LR\_x + 1, LR\_y) + LR(LR\_x, LR\_y))/2$
    else
      $EDGE\_MAP(HR\_x, HR\_y) = true$ ▷ MARK AS EDGE
    end if
  else if HR_x is even $\&$ HR_y is odd **then**
    if $|LR(LR\_x, LR\_y + 1) - LR(LR\_x, LR\_y)| < Threshold$ **then**
      $HR(HR\_x, HR\_y) \leftarrow (LR(LR\_x, LR\_y + 1) + LR(LR\_x, LR\_y))/2$
    else
      $EDGE\_MAP(HR\_x, HR\_y) = true$ ▷ MARK AS EDGE
    end if
  else if HR_x is odd $\&$ HR_y is odd **then**
    if $|LR(LR\_x + 1, LR\_y + 1) - LR(LR\_x, LR\_y)| < Threshold$ **then**
      $HR(HR\_x, HR\_y) \leftarrow (LR(LR\_x + 1, LR\_y + 1) + LR(LR\_x, LR\_y))/2$
    else if $|LR(LR\_x, LR\_y + 1) - LR(LR\_x + 1, LR\_y)| < Threshold$ **then**
      $HR(HR\_x, HR\_y) \leftarrow (LR(LR\_x, LR\_y + 1) + LR(LR\_x + 1, LR\_y))/2$
    else
      $EDGE\_MAP(HR\_x, HR\_y) = true$ ▷ MARK AS EDGE
    end if
  end if
end procedure

Algorithm 3 CalculateEdgePixel Method

procedure CALCULATEEDGEPIXEL(HR,HR_x,HR_y,$EDGE\_MAP$)
    $Y_{2i,2j}$ edge pixel will be calculated which is shown in Figure 3.1c
    if $EDGE\_MAP(HR\_x-1, HR\_y)$ is $false$ & $EDGE\_MAP(HR\_x+1, HR\_y)$ is $false$ **then**
        $Diff1 \leftarrow (HR(HR\_x-1, HR\_y) + HR(HR\_x+1, HR\_y))/2$
    else
        $Diff1 \leftarrow 255$
    end if
    if $EDGE\_MAP(HR\_x+1, HR\_y-1)$ is $false$ & $EDGE\_MAP(HR\_x-1, HR\_y+1)$ is $false$ **then**
        $Diff2 \leftarrow (HR(HR\_x+1, HR\_y-1)+HR(HR\_x-1, HR\_y+1))/2$
    else
        $Diff2 \leftarrow 255$
    end if
    if $EDGE\_MAP(HR\_x, HR\_y-1)$ is $false$ & $EDGE\_MAP(HR\_x, HR\_y+1)$ is $false$ **then**
        $Diff3 \leftarrow (HR(HR\_x, HR\_y-1) + HR(HR\_x, HR\_y+1))/2$
    else
        $Diff3 \leftarrow 255$
    end if
    if $EDGE\_MAP(HR\_x-1, HR\_y-1)$ is $false$ & $EDGE\_MAP(HR\_x+1, HR\_y+1)$ is $false$ **then**
        $Diff4 \leftarrow (HR(HR\_x-1, HR\_y-1)+HR(HR\_x+1, HR\_y+1))/2$
    else
        $Diff4 \leftarrow 255$
    end if

Algorithm 3 CalculateEdgePixel Method (Continued)

23: $MinDiff \leftarrow min(Diff1, Diff2, Diff3, Diff4)$

24: **if** $MinDiff = Diff1$ **then**

25: $HR(HR\_x, HR\_y) \leftarrow (HR(HR\_x - 1, HR\_y) + HR(HR\_x + 1, HR\_y))/2$

26: **else if** $MinDiff = Diff2$ **then**

27: $HR(HR\_x, HR\_y) \leftarrow (HR(HR\_x + 1, HR\_y - 1) + HR(HR\_x - 1, HR\_y + 1))/2$

28: **else if** $MinDiff = Diff3$ **then**

29: $HR(HR\_x, HR\_y) \leftarrow (HR(HR\_x, HR\_y-1)+HR(HR\_x, HR\_y+1))/2$

30: **else if** $MinDiff = Diff4$ **then**

31: $HR(HR\_x, HR\_y) \leftarrow (HR(HR\_x - 1, HR\_y - 1) + HR(HR\_x + 1, HR\_y + 1))/2$

32: **end if**

33: **end procedure**

Algorithm 4 Fast Video Interpolation

1: **procedure** IMRESIZE($LR[3]$, HR) ▷ 3 LR Input Images, HR Output Image
2: $LR\_y \leftarrow 0$ ▷ Low Resolution Image Vertical Position
3: $LR\_x \leftarrow 0$ ▷ Low Resolution Image Horizontal Position
4: $HR\_y \leftarrow 0$ ▷ High Resolution Image Vertical Position
5: $HR\_x \leftarrow 0$ ▷ High Resolution Image Horizontal Position
6: $Scale\_x \leftarrow LR\_Width/HR\_Width$
7: $Scale\_y \leftarrow LR\_Height/HR\_Height$
8: $NeighborSetSize \leftarrow 1$ ▷ Half of Search Window Size
9: $PatchSize \leftarrow 1$ ▷ Half of the Patch Window Size(3x3)
10: $VARIANCE1 \leftarrow 2000$ ▷ Equals to 2 * σ_1^2
11: $VARIANCE2 \leftarrow 60$ ▷ Equals to 2 * σ_2^2
12: $PatchPixelNumber \leftarrow (2 * PatchSize + 1)^2$
13: $numerator \leftarrow 0$
14: $denumerator \leftarrow 0$
15: $patch1\_error \leftarrow 0$
16: $patch2\_error \leftarrow 0$
17: **while** $HR\_y < HR\_Height$ **do**
18: $LR\_y \leftarrow HR\_y \times Scale\_y$
19: **while** $HR\_x < HR\_Width$ **do**
20: $LR\_x \leftarrow HR\_x \times Scale\_x$
21: $ImageIndex \leftarrow 0$
22: **while** $ImageIndex < 3$ **do** ▷ Since there are 3 input LR images
23: $Neighbor\_x \leftarrow -NeighborSetSize$
24: **while** $Neighbor\_x \leq NeighborSetSize$ **do**
25: $Neighbor\_y \leftarrow -NeighborSetSize$
26: **while** $Neighbor\_y \leq NeighborSetSize$ **do**
27: $Shift\_x \leftarrow -PatchSize$
28: **while** $Shift\_x \leq PatchSize$ **do**
29: **while** $Shift\_y \leq PatchSize$ **do**

Algorithm 4 Fast Video Interpolation (Continued)

30: $patch1\_error \leftarrow patch1\_error + |LR[ImageIndex](LR\_x + Neighbor\_x + Shift\_x, LR\_y + Neighbor\_y + Shift\_y) - LR[1](LR\_x + Shift\_x, LR\_y + Shift\_y)|$

31: $patch2\_error \leftarrow patch2\_error + |LR[ImageIndex](LR\_x + Neighbor\_x + Shift\_x, LR\_y + Neighbor\_y + Shift\_y) - LR[2 - ImageIndex](LR\_x - Neighbor\_x + Shift\_x, LR\_y - Neighbor\_y + Shift\_y)|$

32: $Shift\_y \leftarrow Shift\_y + 1$

33: **end while**

34: $Shift\_x \leftarrow Shift\_x + 1$

35: **end while**

36: $patch1\_error \leftarrow patch1\_error/nPatchPixelNumber$

37: $patch2\_error \leftarrow patch2\_error/nPatchPixelNumber$

38: $patch1\_error \leftarrow (patch1\_error)^2$

39: $patch2\_error \leftarrow (patch2\_error)^2$

40: $numerator \leftarrow numerator + LR[ImageIndex](LR\_x + Neighbor\_x, LR\_y + Neighbor\_y) * e^{-(patch1\_error)/VARIANCE1} * e^{-(patch2\_error)/VARIANCE2}$

41: $denumerator \leftarrow denumerator + e^{-(patch1\_error)/VARIANCE1} * e^{-(patch2\_error)/VARIANCE2}$

42: $patch1\_error \leftarrow 0$

43: $patch2\_error \leftarrow 0$

44: $Neighbor\_y \leftarrow Neighbor\_y + 1$

45: **end while**

46: $Neighbor\_x \leftarrow Neighbor\_x + 1$

47: **end while**

48: $Neighbor\_x \leftarrow -NeighborSetSize$

49: **while** $Neighbor\_x \leq NeighborSetSize$ **do**

50: $Neighbor\_y \leftarrow -NeighborSetSize$

51: **while** $Neighbor\_y \leq NeighborSetSize$ **do**

Algorithm 4 Fast Video Interpolation (Continued)

$Shift\_x \leftarrow -PatchSize$
while $Shift\_x \leq PatchSize$ **do**
while $Shift\_y \leq PatchSize$ **do**
$patch2\_error \leftarrow patch2\_error + |LR[1](LR\_x + Neighbor\_x + Shift\_x, LR\_y + Neighbor\_y + Shift\_y) - LR[1](LR\_x - Neighbor\_x + Shift\_x, LR\_y - Neighbor\_y + Shift\_y)|$
$Shift\_y \leftarrow Shift\_y + 1$
end while
$Shift\_x \leftarrow Shift\_x + 1$
end while
$patch2\_error \leftarrow patch2\_error / nPatchPixelNumber$
$patch2\_error \leftarrow (patch2\_error)^2$
$numerator \leftarrow numerator + LR[ImageIndex](LR\_x + Neighbor\_x, LR\_y + Neighbor\_y) * e^{-(patch2\_error)/VARIANCE2}$
$denumerator \leftarrow denumerator + e^{-(patch2\_error)/VARIANCE2}$
$patch2\_error \leftarrow 0$
$Neighbor\_y \leftarrow Neighbor\_y + 1$
end while
$Neighbor\_x \leftarrow Neighbor\_x + 1$
end while
$HR(HR\_x, HR\_y) \leftarrow numerator / denumerator$
$numerator \leftarrow 0$
$denumerator \leftarrow 0$
$ImageIndex \leftarrow ImageIndex + 1$
end while
$HR\_x \leftarrow HR\_x + 1$
end while
$HR\_y \leftarrow HR\_y + 1$
end while
end procedure

APPENDIX B

INTRODUCTION TO MMRGLIBRARY

In the scope of this thesis, mmrgLibrary, which is developed by me, will be introduced. This is a computer vision library which consists of many third party libraries, such as OPENCV, FFMPEG, X264, X265, DirectFB, Posix Thread and OPENCL; moreover, customized algorithms which are implemented by taking reference of published articles. This API is capable of running both in X86 linux platform and ARM linux platform. If toolchain is provided for other platforms, the API can be built again easily and it can be used in other platforms by the aid of CMake.

One of the main reason of creating this library is that it makes developing an algorithm simple for developers. It makes the developing environment compact so that every complicated task can be done just by using one command. When this complicated command is called, the customer do not need to know the details about this command.

Although OPENCV library includes many algorithms, it was first created and optimized for INTEL platform since it is developed by INTEL.[14] Then it can support arm platforms for developer boards such as Raspberry Pi, Pandaboard or Beaglebone etc. These developers board mostly have operating system inside it and all the packages are provided with the operating system. The foremost example for this case is showing the image on the screen. When imshow function is called in OPENCV; GTK and QT graphic libraries should be installed in your system in order to see the image on the screen. However, when we consider an embedded device without operating system, such as linux televisions, GTK and QT are not installed. Therefore, when the developer calls imshow function provided by OPENCV, the system will crash or OPENCV library cannot be compiled since the libraries cannot be found. Instead of using GTK and QT, the television platform uses another graphic library, called Di-

rectFB. The other reason of creating this library is that it should also be worked for television platform or any embedded device without operating system.

In this chapter; first of all, mmrgLibrary structure will be explained and the usage of mmrgLibray will be introduced. Secondly, third party libraries, which are used in mmrgLibrary, will be introduced briefly. After that, mmrgLibrary class diagram will be presented and finally, an example will be given for resizing a raw video.

B.1 Structure of mmrgLibrary

mmrgLibrary is C++ shared library which is created by compiling all the source codes located in its source directory. It also includes third party libraries externally by linking them into mmrgLibrary. Therefore, the developer, who would like to write a code by using mmrgLibrary, can use the methods of all the third party libraries such as OPENCV, FFMPEG, X264, X265, DirectFB, Posix Thread and OPENCL. The source directory and build directory can be found in figure B.1.

As it can be seen in figure B.1, there are two main directory in mmrgLibrary folder; namely, **Trunk** and **Build**. **Trunk** directory is the source directory of mmrgLibrary API. All the source codes, header files and third party libraries are located at this folder. It is critically note that when this API is compiled, nothing will be changed in this folder. All the output files will be created in the **Build** directory.

Except the third party libraries located in the **3rdparty** directory, two important tools are used to create this library. One of them is **CMake**, the other one is the **Git**.

Git is a widely-used version control system which is developed by Linux kernel developers. It stores every version when it is committed to the system. By the aid of **Git**, developer can revert their modifications back. Therefore, developer can write the code more safely.

CMake is the cross-platform, open-source build system. It is a family of tools designed to build, test and package software. It is used to control the software compilation process using simple platform and compiler independent configuration files. It generates native makefiles and workspaces that can be used in the compiler environ-

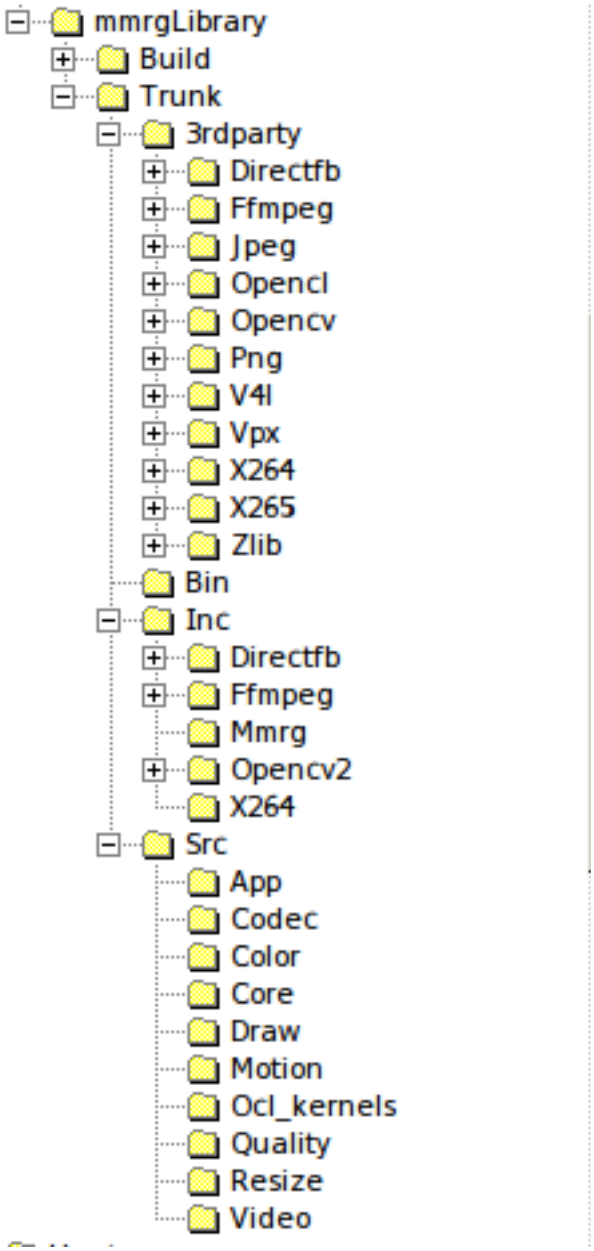


Figure B.1: View of mmrgLibrary Source and Build Directory

ment of developer choice. While working with a project with many modules, writing native makefiles can be very challenging task. CMake makes this process very easy and adaptable. When a developer would like to port a new module with many source codes, he/she needs to change just a couple of line to achieve this. Moreover, target platform can be changed by toggling parameters in the mmrgLibrary configuration file by the aid of CMake. As it can be seen in the figure B.2, if a developer would like to change target platform from X86 to arm, **PLATFORM** parameter should be set to ARM. The modules of third party libraries which is used in mmrgLibrary can also be modified by using this file. If a developer would like to remove **ocl** module of OpenCV, he/she needs to erase this line to disable OPENCL support of OPENCV.

After arranging the configuration file depend on the target platform, the developer should open the terminal program in linux and enter **mmrgLibrary/build** directory. All the output files will be extracted here so that it can be given to customer. Therefore, customer will not see the source code of mmrgLibrary. In order to build the API, **compile.sh** script, which is located in **build** directory, should be called. After the compilation finishes, **build_output** folder will be created in **build** directory. All the shared libraries, header files and binaries can be found under this folder as it can be seen in the figure B.3.

Since API is compiled for x86 platform, **x86** folder is created in the **build_output** directory. As it can be seen in the figure B.3; the modules, which are enabled in mmrgLibrary Cmake Configuration file for FFMPEG and OPENCV, are copied to build directory so that it can be used in customer platform.

B.2 Class Diagram of mmrgLibrary

As it can be seen in figure B.1, there are nine modules located under the **src** directory. These modules have different responsibilities. All modules are used in the scope of this thesis and they will be explained briefly. After that, **resize** module class diagram will be shown.

App module is presented ready-to-use solutions from the application layer. Customer should call a method of this modules in order to achieve a task.

```
set(PLATFORM "X86")                          # X86   |   ARM
set(OPENCV "USED")                           # USED  |   NOT_USED
set(DIRECT_FRAME_BUFFER "NOT_USED")          # USED  |   NOT_USED
set(FFMPEG "USED")                           # USED  |   NOT_USED
set(X264 "USED")                             # USED  |   NOT_USED
set(OPENCL "USED")                           # USED  |   NOT_USED

set(TARGET_NAME "SUPERNOVA")             # SUPERNOVA |


# ------------------  OPENCV CONFIGURATION  ------------------
set(OPENCV_MAJOR_NUMBER 2)
set(OPENCV_MINOR_NUMBER1 4)
set(OPENCV_MINOR_NUMBER2 9)

set(OPENCV_MODULES      core
                        imgproc
                        highgui
                        objdetect
                        features2d
                        flann
                        ml
                        calib3d
                        video
                        photo
                        superres
                        ocl)

# ------------------  OPENCV CONFIGURATION  ------------------



# ------------------  FFMPEG CONFIGURATION  ------------------
set(FFMPEG_MODULES      avcodec
                        avdevice
                        avfilter
                        avutil
                        avformat
                        postproc
                        swresample
                        swscale)

# ------------------  FFMPEG CONFIGURATION  ------------------


# ------------------  X264 CONFIGURATION  --------------------
set(X264_MODULES        x264)

# ------------------  X264 CONFIGURATION  --------------------


# ------------------  DIRECTFB CONFIGURATION  ----------------
set(DIRECTFB_MAJOR_NUMBER 1)
set(DIRECTFB_MINOR_NUMBER1 4)
set(DIRECTFB_MINOR_NUMBER2 2)

set(DIRECTFB_MODULES        directfb)

# ------------------  DIRECTFB CONFIGURATION  ----------------
```

Figure B.2: mmrgLibrary CMake Configuration File

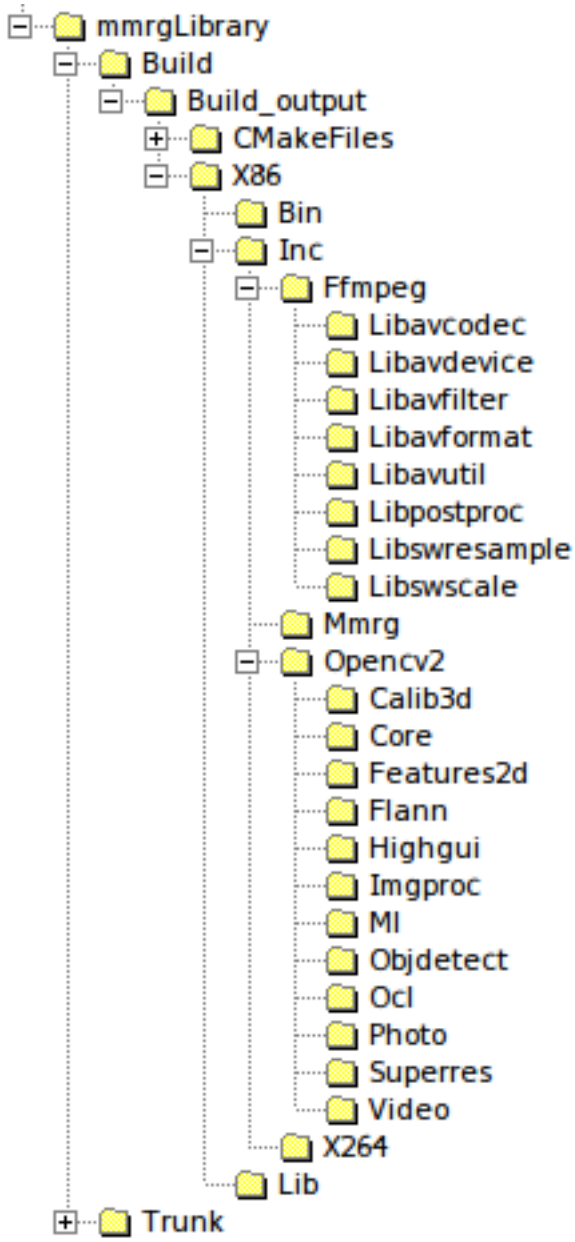


Figure B.3: mmrgLibrary CMake Configuration File

Codec module is presented to save and load image from file system and showing image to screen. As it is stated before, both GTK, QT graphic library and DirectFB graphic library can be used depend on the platform. In our television, we are using DirectFB library.

Color module is presented for color-wise operations such as converting color space, splitting channels or merging channels.

Core module is the basis of the mmrgLibrary API. Most important class of this API is **mmrgImage** container class. All the class are creating object of this class to process an image. The other class of this module is **mmrgFactory** class. Since mmrgLibrary is using Factory Design Pattern, all the classes, which create an instance of mmrgLibrary class, should call a method of mmrgFactory.

Draw module is introduced to draw a shape or text on an image.

Motion module is presented to estimate motion vectors between two frames by using several algorithms. Also, this module is used to shift the image in horizontal and vertical directions.

Quality module is introduced to measure the quality performance of images.

Resize module is the widely used module in this thesis. It is presented to resize image by using many interpolation algorithms and super resolution algorithms. The class diagram of this module can be found in figure B.4.

Video module is used to capture frame from camera or any raw video. It can also be used to save video as raw data.

Since all the modules have same design architecture, only **resize** module's class diagram will be shown in this chapter. As it can be seen in figure B.4, there is a abstract class, called mmrgResize, at the top of the diagram. Any method, which would like to use resize function in this API, must call this class. This abstract class is responsible to assign a proper class so that it provides what caller method requests.

Figure B.4: mmrgLibrary Class Diagram of Resize module

B.3 mmrgResize Example

if the caller method would like to use Bicubic Interpolation which is implemented by OpenCV, it needs to call the function as it can be seen in figure B.5.

```
void ResizeImageExample()
{
    int nInputImageWidth = 176;
    int nInputImageHeight = 144;
    int nInputImageNumberofChannel = 3;
    mmrgPixelValue nInputImagePixelValue(255,255,255);
    mmrgImage cInputImage(nInputImageWidth,nInputImageHeight,nInputImageNumberofChannel,nInputImagePixelValue);


    int nOutputImageWidth = 352;
    int nOutputImageHeight = 288;
    int nOutputImageNumberofChannel = 3;
    mmrgImage cOutputImage(nOutputImageWidth,nOutputImageHeight,nOutputImageNumberofChannel);

    mmrgResize::GetInstance(EN_MMRG_RESIZE_TYPE_BICUBIC, EN_MMRG_USED_API_OPENCV)->imResize(cInputImage, cOutputImage);

}
```

Figure B.5: mmrgLibrary Resize Image Example by using Bicubic Interpolation

If the caller method would like to use Edge Adaptive Interpolation, which is implemented by using Native C++, it needs to call the function as it can be seen in figure B.6.

```
void ResizeImageExample2()
{
    int nInputImageWidth = 176;
    int nInputImageHeight = 144;
    int nInputImageNumberofChannel = 3;
    mmrgPixelValue nInputImagePixelValue(255,255,255);
    mmrgImage cInputImage(nInputImageWidth,nInputImageHeight,nInputImageNumberofChannel,nInputImagePixelValue);


    int nOutputImageWidth = 352;
    int nOutputImageHeight = 288;
    int nOutputImageNumberofChannel = 3;
    mmrgImage cOutputImage(nOutputImageWidth,nOutputImageHeight,nOutputImageNumberofChannel);

    mmrgResize::GetInstance(EN_MMRG_RESIZE_TYPE_EDGE_ADAPTIVE_INTERPOLATION, EN_MMRG_USED_API_NATIVE)->imResize(cInputImage, cOutputImage);

}
```

Figure B.6: mmrgLibrary Resize Image Example by using Fast Edge-Adaptive Interpolation

By the aid of the **App** module, some of the solutions are ready to use instead of writing many lines of codes. In order to upscale a raw video which has a chroma sub-sampling 4:2:0, the block of codes, which is shown in figure B.7, makes the experiment much more easy and adaptable to new algorithms. This block of code will

- read the frames from input raw video
- split the channels of the frames, respectively luminance and two chrominance

- upscale luminance channel by using the selected algorithm
- upscale two chroma algorithm by using nearest neighbour algorithm
- merge upscaled channels into output image
- write output frame into output file

```
void ResizeYUVVideoApp(int argc, char* argv[])
{
    mmrgVideoCapture_NativeRaw_Conf eInputConf;
    eInputConf.m_nImageHeight = 288;
    eInputConf.m_nImageWidth = 352;
    eInputConf.m_eChromaSubSampling = EN_MMRG_CHROMA_SUBSAMPLING_4_2_0;
    eInputConf.m_eFileCommandMode = EN_MMRG_FILE_COMMAND_MODE_READ;
    std::string strVideoName(argv[1]);
    eInputConf.m_strVideoName = strVideoName;

    mmrgVideoCapture_NativeRaw_Conf eOutputConf;
    eOutputConf.m_nImageHeight = 576;
    eOutputConf.m_nImageWidth = 704;
    eOutputConf.m_eChromaSubSampling = EN_MMRG_CHROMA_SUBSAMPLING_4_2_0;
    eOutputConf.m_eFileCommandMode = EN_MMRG_FILE_COMMAND_MODE_WRITE;
    std::string strVideoName2(argv[2]);
    eOutputConf.m_strVideoName = strVideoName2;

    mmrgResizeApp::GetInstance()->RecordResizedYUVFile(eInputConf, eOutputConf, EN_MMRG_RESIZE_TYPE_FAST_VIDEO_INTERPOLATION);
} « end ResizeYUVVideoApp »
```

Figure B.7: mmrgLibrary Resize Raw Video Example by using Fast Video Interpolation

In the experiments whose results are presented in the next chapter, the block of code which is shown in figure B.7 will be used.

APPENDIX C

VISUAL RESULTS

(a) 200th HR Frame

(b) 199th LR Frame

(c) 200th LR Frame

(d) 201st LR Frame

Figure C.1: Flower Video 200th HR Frame with three bi-directional LR Frames

(a) 155th HR Frame

(b) 154th LR Frame

(c) 155th LR Frame

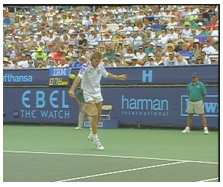
(d) 156th LR Frame

Figure C.2: Foreman Video 155th HR Frame with three bi-directional LR Frames

(a) 91st HR Frame

(b) 90th LR Frame (c) 91th LR Frame (d) 92th LR Frame

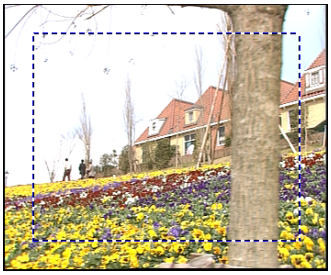
Figure C.3: News Video 91st HR Frame with three bi-directional LR Frames



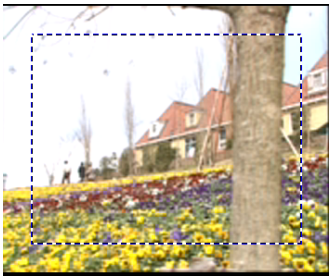
(a) 22nd HR Frame

(b) 21st LR Frame (c) 22nd LR Frame (d) 23rd LR Frame

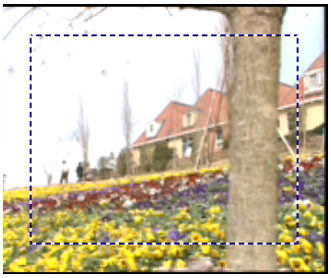
Figure C.4: Stefan Video 22nd HR Frame with three bi-directional LR Frames



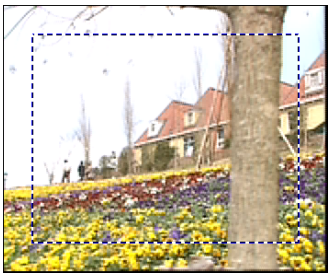
(a) Original Frame



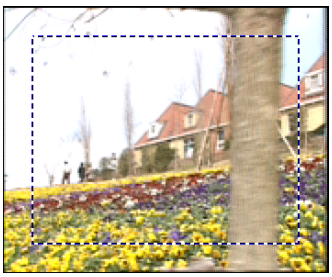
(b) Bicubic Interpolation



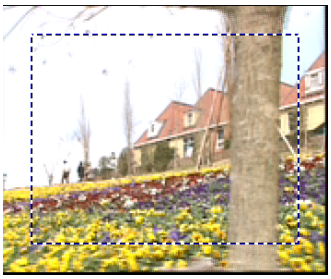
(c) Fast Edge-Adaptive Interpolation



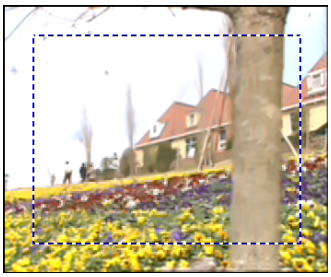
(d) Iterative Back Projection



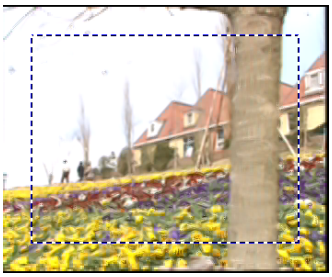
(e) Maximum Likelihood with PBME



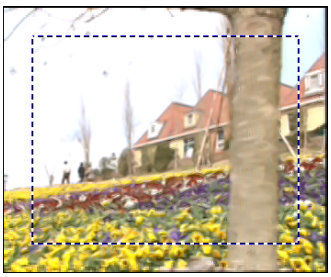
(f) Maximum Likelihood with BMFS



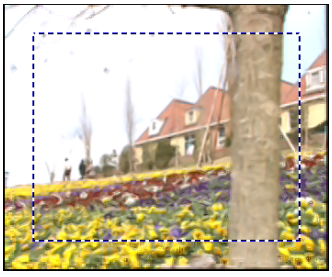
(g) Super Resolution with PME



(h) Fast Video Interpolation



(i) Fast Video Interpolation with PBME



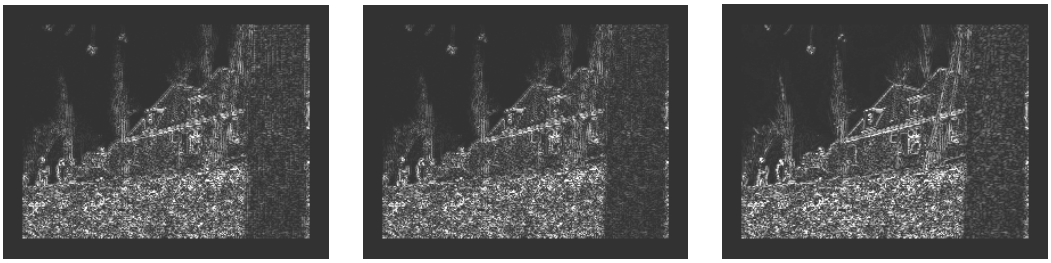
(j) Fast Video Interpolation with BMFS

Figure C.5: Flower Video 200th Upscaled HR Frame

(a) Bicubic Interpolation

(b) Fast Edge-Adaptive Interpolation

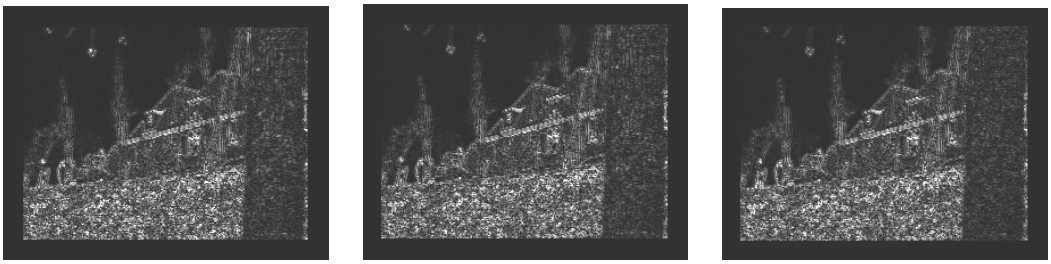
(c) Iterative Back Projection



(d) Maximum Likelihood with PBME

(e) Maximum Likelihood with BMFS

(f) Super Resolution with PME

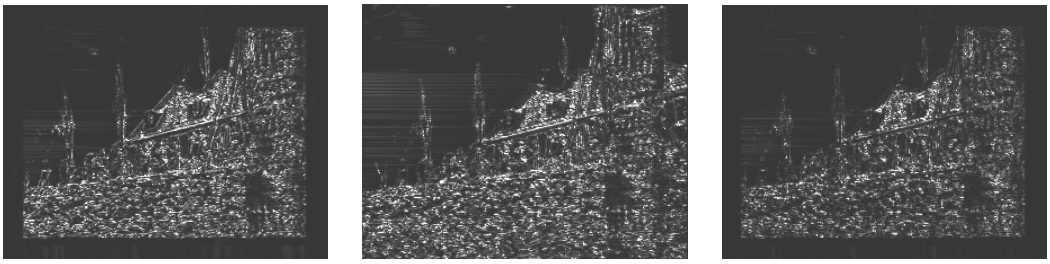


(g) Fast Video Interpolation

(h) Fast Video Interpolation with PBME

(i) Fast Video Interpolation with BMFS

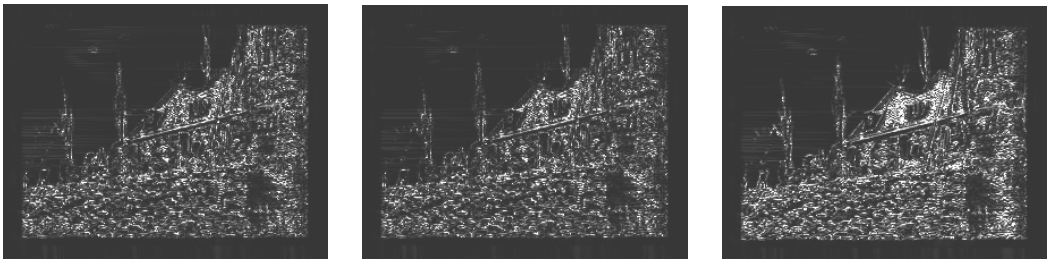
Figure C.6: PSNR Difference Map for Flower Video 200th Upscaled HR Frame



(a) Bicubic Interpolation

(b) Fast Edge-Adaptive Interpolation

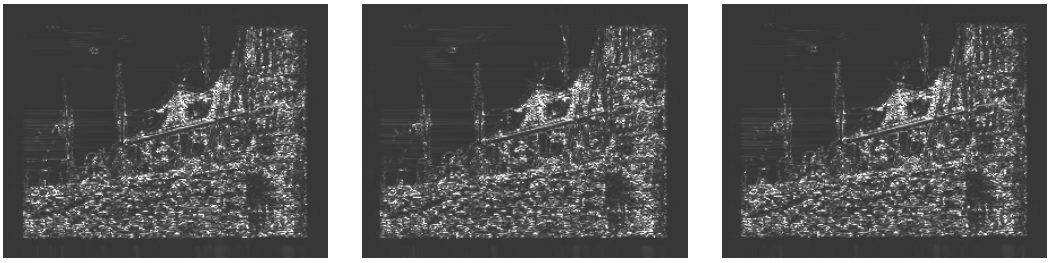
(c) Iterative Back Projection



(d) Maximum Likelihood with PBME

(e) Maximum Likelihood with BMFS

(f) Super Resolution with PME

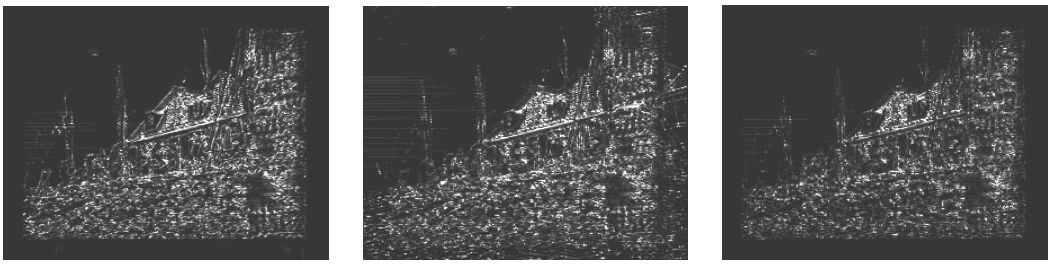


(g) Fast Video Interpolation

(h) Fast Video Interpolation with PBME

(i) Fast Video Interpolation with BMFS

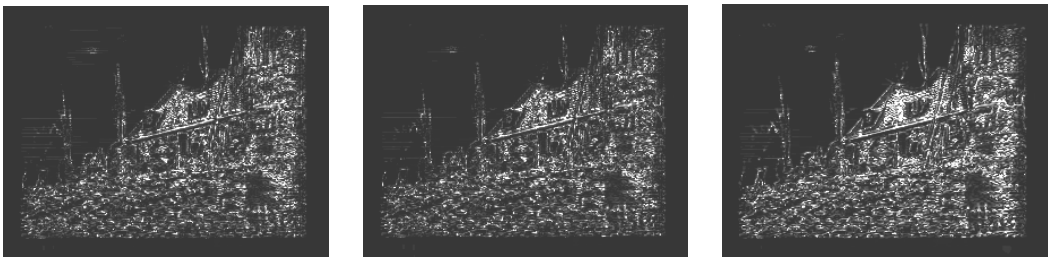
Figure C.7: DMOS Difference Map for Flower Video 200th Upscaled HR Frame



(a) Bicubic Interpolation

(b) Fast Edge-Adaptive Interpolation

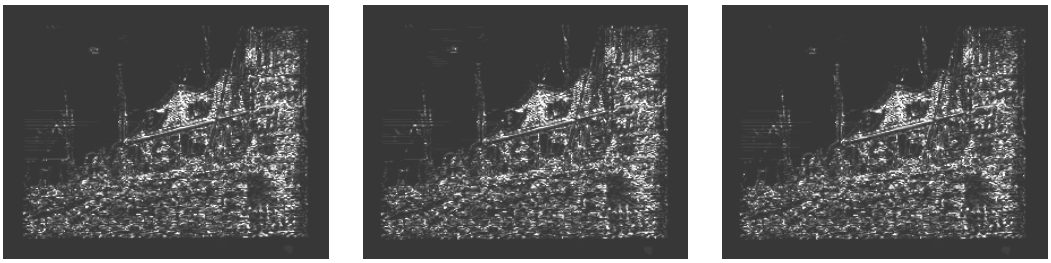
(c) Iterative Back Projection



(d) Maximum Likelihood with PBME

(e) Maximum Likelihood with BMFS

(f) Super Resolution with PME



(g) Fast Video Interpolation

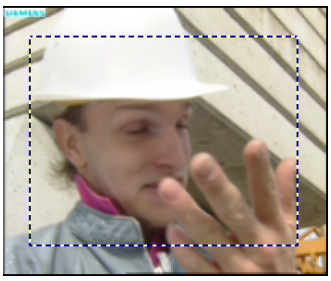
(h) Fast Video Interpolation with PBME

(i) Fast Video Interpolation with BMFS

Figure C.8: ADMOS Difference Map for Flower Video 200th Upscaled HR Frame

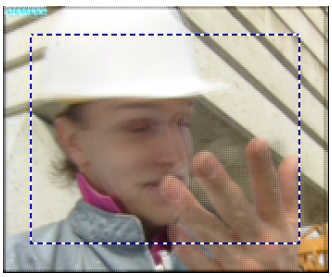
(a) Original Frame

(b) Bicubic Interpolation

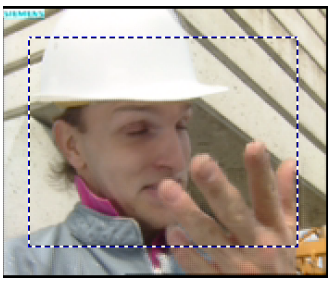


(c) Fast Edge-Adaptive Interpolation

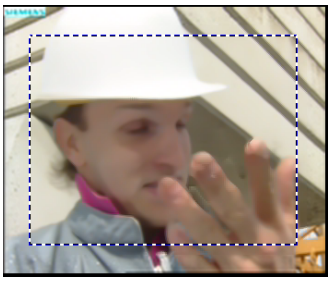
(d) Iterative Back Projection



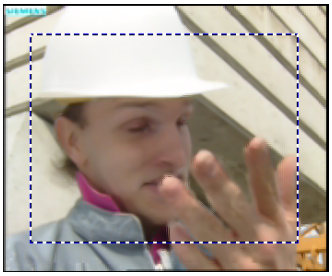
(e) Maximum Likelihood with PBME



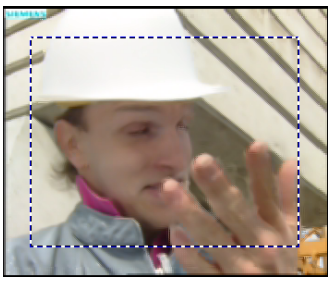
(f) Maximum Likelihood with BMFS



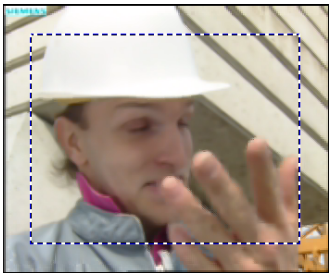
(g) Super Resolution with PME



(h) Fast Video Interpolation

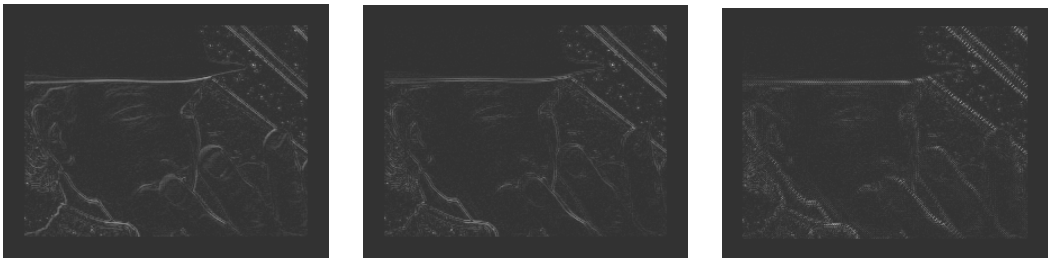


(i) Fast Video Interpolation with PBME



(j) Fast Video Interpolation with BMFS

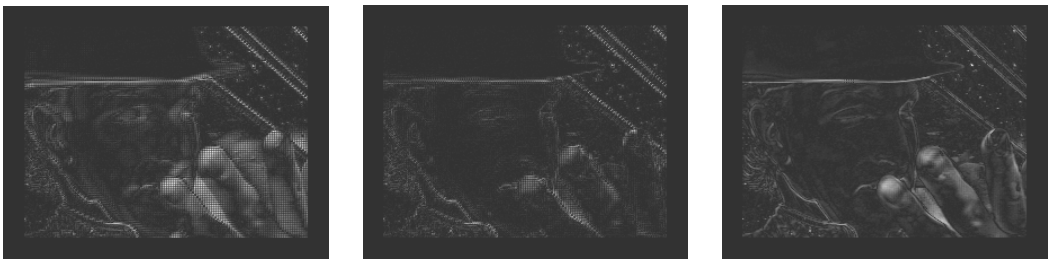
Figure C.9: Foreman Video 155th Upscaled HR Frame



(a) Bicubic Interpolation

(b) Fast Edge-Adaptive Interpolation

(c) Iterative Back Projection



(d) Maximum Likelihood with PBME

(e) Maximum Likelihood with BMFS

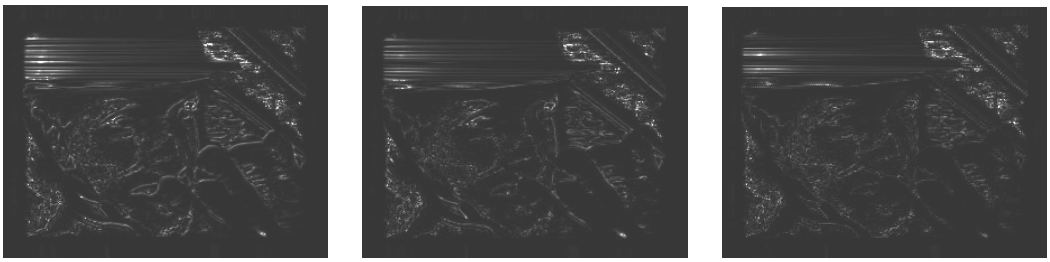
(f) Super Resolution with PME

(g) Fast Video Interpolation

(h) Fast Video Interpolation with PBME

(i) Fast Video Interpolation with BMFS

Figure C.10: PSNR Difference Map for Foreman Video 155th Upscaled HR Frame



(a) Bicubic Interpolation

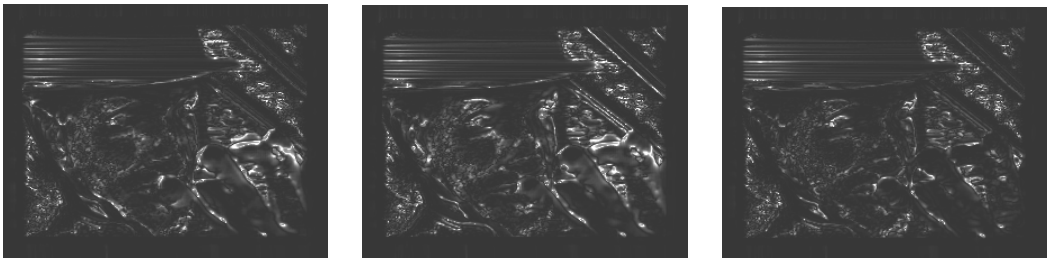
(b) Fast Edge-Adaptive Interpolation

(c) Iterative Back Projection

(d) Maximum Likelihood with PBME

(e) Maximum Likelihood with BMFS

(f) Super Resolution with PME

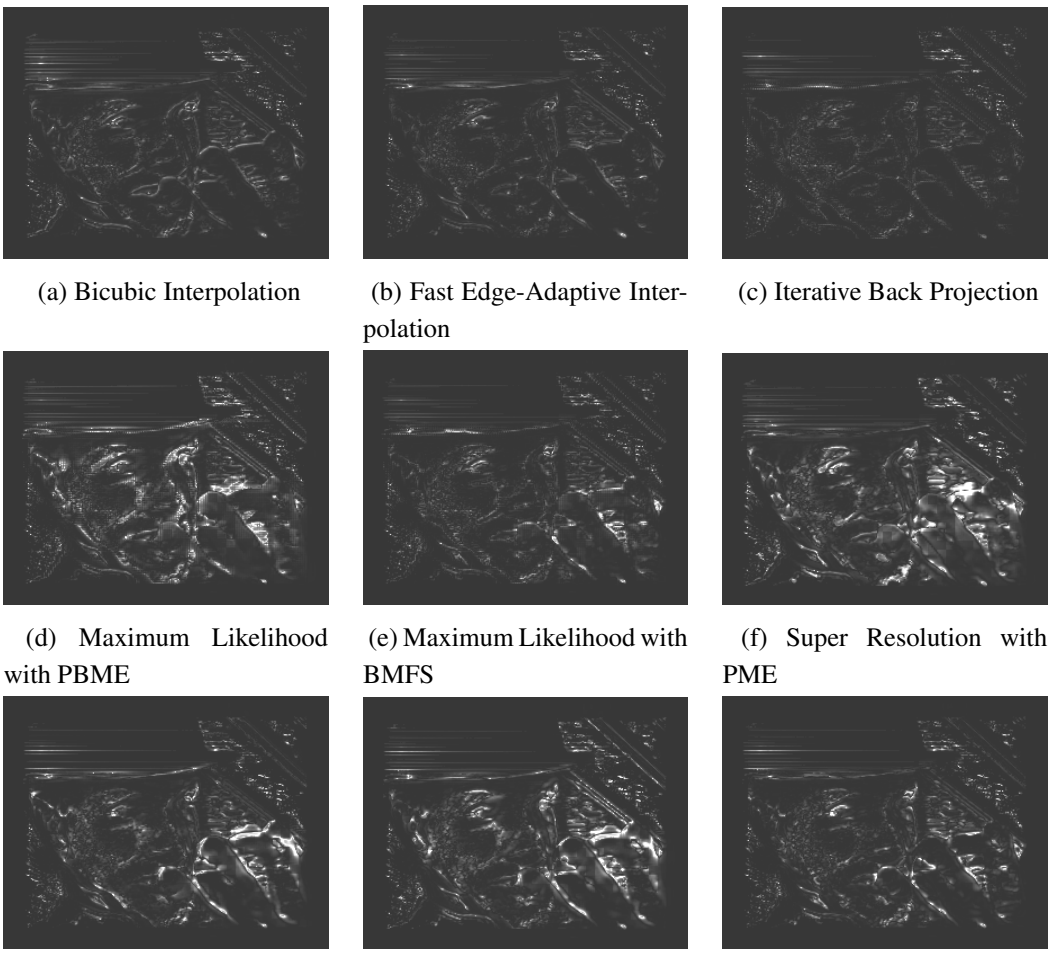


(g) Fast Video Interpolation

(h) Fast Video Interpolation with PBME

(i) Fast Video Interpolation with BMFS

Figure C.11: DMOS Difference Map for Foreman Video 155th Upscaled HR Frame



(a) Bicubic Interpolation

(b) Fast Edge-Adaptive Interpolation

(c) Iterative Back Projection

(d) Maximum Likelihood with PBME

(e) Maximum Likelihood with BMFS

(f) Super Resolution with PME

(g) Fast Video Interpolation

(h) Fast Video Interpolation with PBME

(i) Fast Video Interpolation with BMFS

Figure C.12: ADMOS Difference Map for Foreman Video 155th Upscaled HR Frame

(a) Original Frame

(b) Bicubic Interpolation

(c) Fast Edge-Adaptive Interpolation

(d) Iterative Back Projection

(e) Maximum Likelihood with PBME

(f) Maximum Likelihood with BMFS

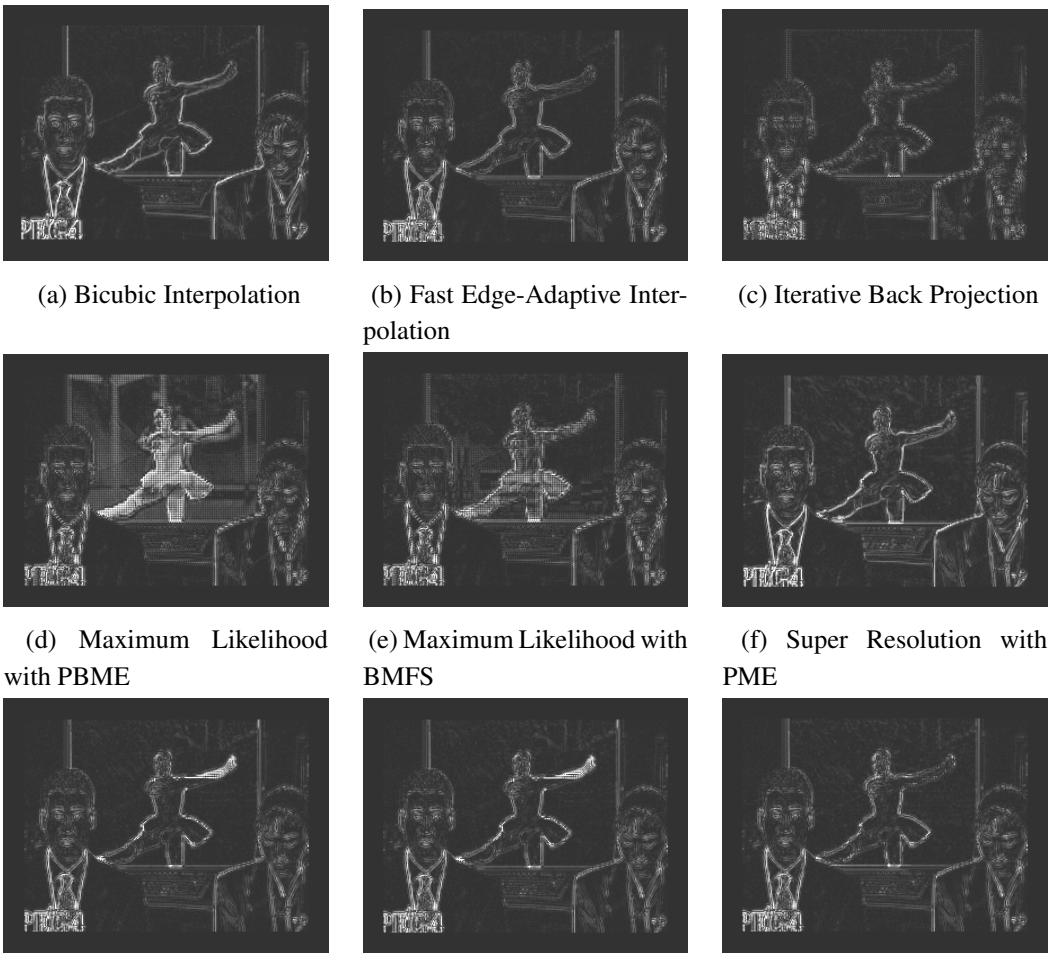
(g) Super Resolution with PME

(h) Fast Video Interpolation

(i) Fast Video Interpolation with PBME

(j) Fast Video Interpolation with BMFS

Figure C.13: Foreman Video 91st Upscaled HR Frame



(a) Bicubic Interpolation

(b) Fast Edge-Adaptive Interpolation

(c) Iterative Back Projection

(d) Maximum Likelihood with PBME

(e) Maximum Likelihood with BMFS

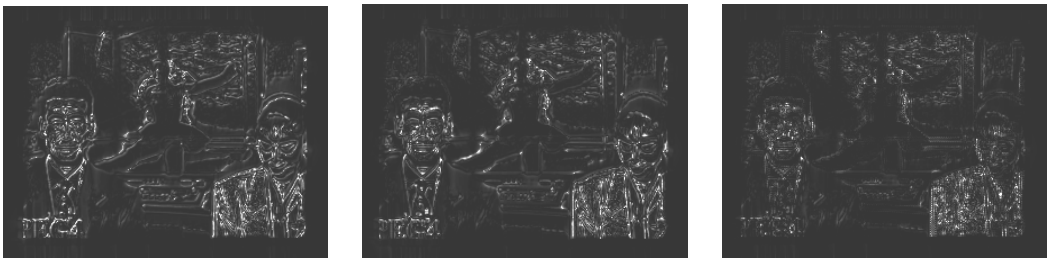
(f) Super Resolution with PME

(g) Fast Video Interpolation

(h) Fast Video Interpolation with PBME

(i) Fast Video Interpolation with BMFS

Figure C.14: PSNR Difference Map for News Video 91st Upscaled HR Frame



(a) Bicubic Interpolation

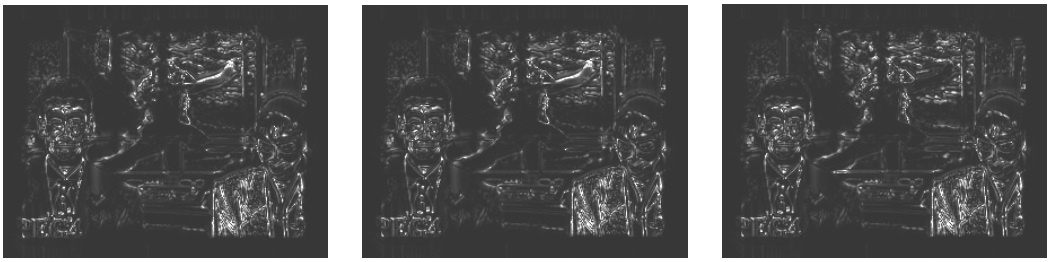
(b) Fast Edge-Adaptive Interpolation

(c) Iterative Back Projection

(d) Maximum Likelihood with PBME

(e) Maximum Likelihood with BMFS

(f) Super Resolution with PME

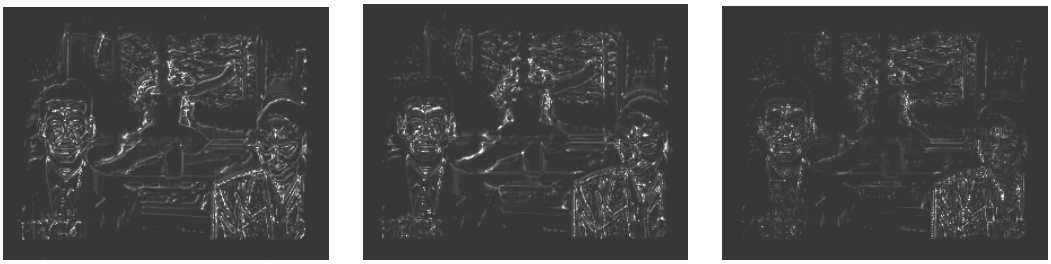


(g) Fast Video Interpolation

(h) Fast Video Interpolation with PBME

(i) Fast Video Interpolation with BMFS

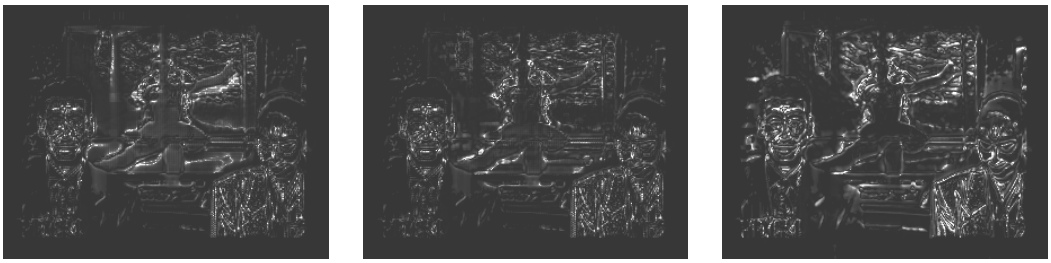
Figure C.15: DMOS Difference Map for News Video 91st Upscaled HR Frame



(a) Bicubic Interpolation

(b) Fast Edge-Adaptive Interpolation

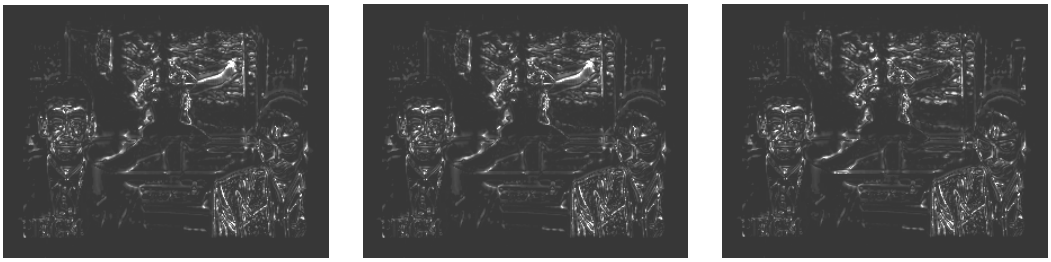
(c) Iterative Back Projection



(d) Maximum Likelihood with PBME

(e) Maximum Likelihood with BMFS

(f) Super Resolution with PME

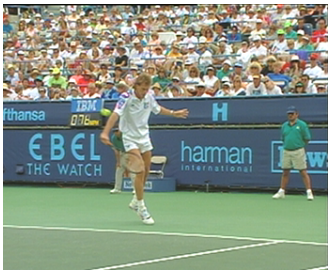


(g) Fast Video Interpolation

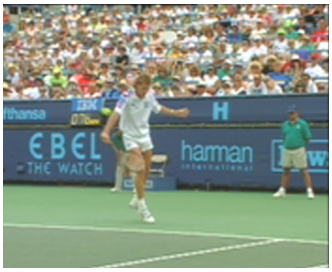
(h) Fast Video Interpolation with PBME

(i) Fast Video Interpolation with BMFS

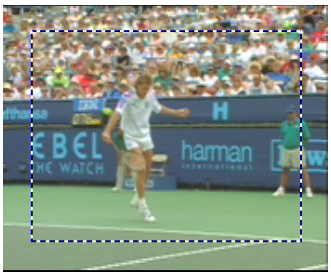
Figure C.16: ADMOS Difference Map for News Video 91st Upscaled HR Frame



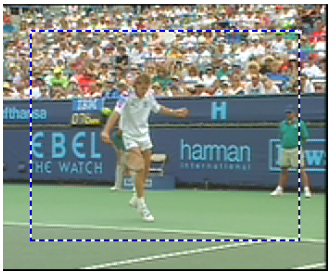
(a) Original Frame



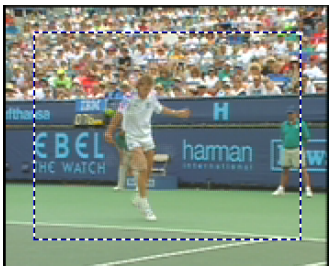
(b) Bicubic Interpolation



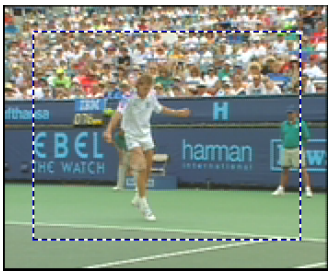
(c) Fast Edge-Adaptive Interpolation



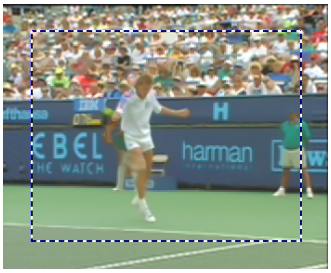
(d) Iterative Back Projection



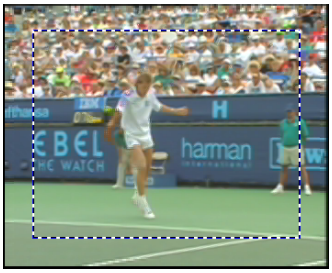
(e) Maximum Likelihood with PBME



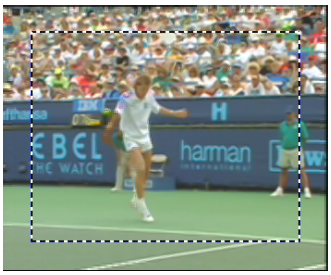
(f) Maximum Likelihood with BMFS



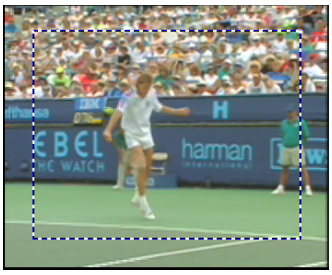
(g) Super Resolution with PME



(h) Fast Video Interpolation



(i) Fast Video Interpolation with PBME

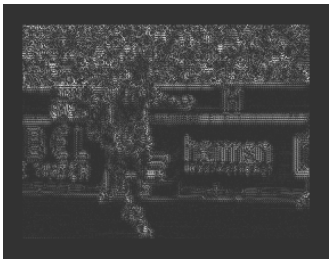


(j) Fast Video Interpolation with BMFS

Figure C.17: Stefan Video 22nd Upscaled HR Frame

(a) Bicubic Interpolation

(b) Fast Edge-Adaptive Interpolation



(c) Iterative Back Projection

(d) Maximum Likelihood with PBME

(e) Maximum Likelihood with BMFS

(f) Super Resolution with PME

(g) Fast Video Interpolation

(h) Fast Video Interpolation with PBME

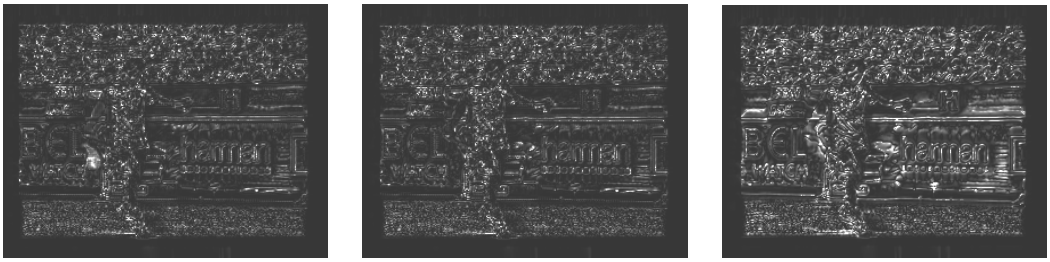
(i) Fast Video Interpolation with BMFS

Figure C.18: PSNR Difference Map for Stefan Video 22nd Upscaled HR Frame

(a) Bicubic Interpolation

(b) Fast Edge-Adaptive Interpolation

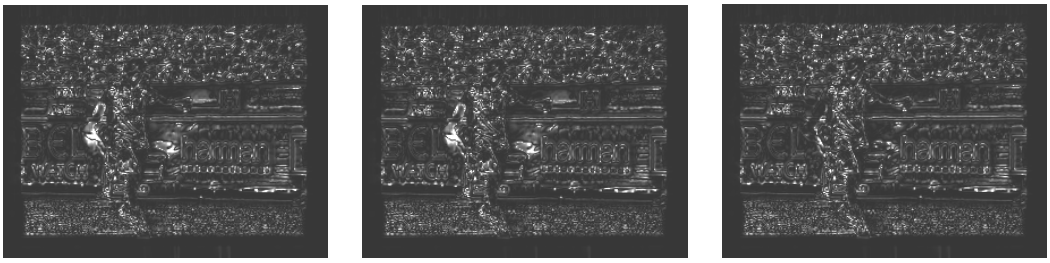
(c) Iterative Back Projection



(d) Maximum Likelihood with PBME

(e) Maximum Likelihood with BMFS

(f) Super Resolution with PME



(g) Fast Video Interpolation

(h) Fast Video Interpolation with PBME

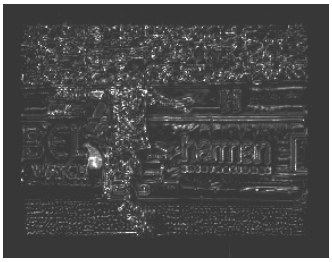
(i) Fast Video Interpolation with BMFS

Figure C.19: DMOS Difference Map for Stefan Video 22nd Upscaled HR Frame

(a) Bicubic Interpolation

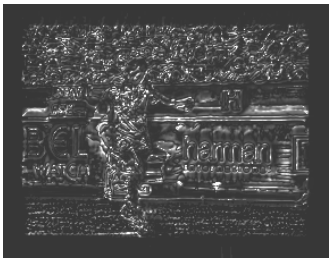
(b) Fast Edge-Adaptive Interpolation

(c) Iterative Back Projection



(d) Maximum Likelihood with PBME

(e) Maximum Likelihood with BMFS



(f) Super Resolution with PME

(g) Fast Video Interpolation

(h) Fast Video Interpolation with PBME

(i) Fast Video Interpolation with BMFS

Figure C.20: ADMOS Difference Map for Stefan Video 22nd Upscaled HR Frame

APPENDIX D

CPU DETAILS

The details of the CPU which is used in PC environment are as follows.

$ cat /proc/cpuinfo
processor : 0
vendor_id : GenuineIntel
cpu family : 6
model : 61
model name : Intel(R) Core(TM) i7-5600U CPU @ 2.60GHz
stepping : 4
microcode : 0x21
cpu MHz : 2000.578
cache size : 4096 KB
physical id : 0
siblings : 4
core id : 0
cpu cores : 2
apicid : 0
initial apicid : 0
fpu : yes
fpu_exception : yes
cpuid level : 20
wp : yes
flags : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush dts acpi mmx fxsr sse sse2 ss ht tm pbe syscall nx pdpe1gb rdtscp lm con-

stant_tsc arch_perfmon pebs bts rep_good nopl xtopology nonstop_tsc aperfmperf eagerfpu pni pclmulqdq dtes64 monitor ds_cpl vmx smx est tm2 ssse3 fma cx16 xtpr pdcm pcid sse4_1 sse4_2 x2apic movbe popcnt tsc_deadline_timer aes xsave avx f16c rdrand lahf_lm abm 3dnowprefetch ida arat epb pln pts dtherm tpr_shadow vnmi flexpriority ept vpid fsgsbase tsc_adjust bmi1 hle avx2 smep bmi2 erms invpcid rtm rdseed adx smap xsaveopt
bugs :
bogomips : 5187.85
clflush size : 64
cache_alignment : 64
address sizes : 39 bits physical, 48 bits virtual
power management:

processor : 1
vendor_id : GenuineIntel
cpu family : 6
model : 61
model name : Intel(R) Core(TM) i7-5600U CPU @ 2.60GHz
stepping : 4
microcode : 0x21
cpu MHz : 1477.125
cache size : 4096 KB
physical id : 0
siblings : 4
core id : 0
cpu cores : 2
apicid : 1
initial apicid : 1
fpu : yes
fpu_exception : yes
cpuid level : 20
wp : yes
flags : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36

clflush dts acpi mmx fxsr sse sse2 ss ht tm pbe syscall nx pdpe1gb rdtscp lm constant_tsc arch_perfmon pebs bts rep_good nopl xtopology nonstop_tsc aperfmperf eagerfpu pni pclmulqdq dtes64 monitor ds_cpl vmx smx est tm2 ssse3 fma cx16 xtpr pdcm pcid sse4_1 sse4_2 x2apic movbe popcnt tsc_deadline_timer aes xsave avx f16c rdrand lahf_lm abm 3dnowprefetch ida arat epb pln pts dtherm tpr_shadow vnmi flexpriority ept vpid fsgsbase tsc_adjust bmi1 hle avx2 smep bmi2 erms invpcid rtm rdseed adx smap xsaveopt
bugs :
bogomips : 5187.85
clflush size : 64
cache_alignment : 64
address sizes : 39 bits physical, 48 bits virtual
power management:

processor : 2
vendor_id : GenuineIntel
cpu family : 6
model : 61
model name : Intel(R) Core(TM) i7-5600U CPU @ 2.60GHz
stepping : 4
microcode : 0x21
cpu MHz : 2624.273
cache size : 4096 KB
physical id : 0
siblings : 4
core id : 1
cpu cores : 2
apicid : 2
initial apicid : 2
fpu : yes
fpu_exception : yes
cpuid level : 20
wp : yes

flags : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush dts acpi mmx fxsr sse sse2 ss ht tm pbe syscall nx pdpe1gb rdtscp lm constant_tsc arch_perfmon pebs bts rep_good nopl xtopology nonstop_tsc aperfmperf eagerfpu pni pclmulqdq dtes64 monitor ds_cpl vmx smx est tm2 ssse3 fma cx16 xtpr pdcm pcid sse4_1 sse4_2 x2apic movbe popcnt tsc_deadline_timer aes xsave avx f16c rdrand lahf_lm abm 3dnowprefetch ida arat epb pln pts dtherm tpr_shadow vnmi flexpriority ept vpid fsgsbase tsc_adjust bmi1 hle avx2 smep bmi2 erms invpcid rtm rdseed adx smap xsaveopt
bugs :
bogomips : 5187.85
clflush size : 64
cache_alignment : 64
address sizes : 39 bits physical, 48 bits virtual
power management:

processor : 3
vendor_id : GenuineIntel
cpu family : 6
model : 61
model name : Intel(R) Core(TM) i7-5600U CPU @ 2.60GHz
stepping : 4
microcode : 0x21
cpu MHz : 2886.304
cache size : 4096 KB
physical id : 0
siblings : 4
core id : 1
cpu cores : 2
apicid : 3
initial apicid : 3
fpu : yes
fpu_exception : yes
cpuid level : 20

wp : yes
flags : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush dts acpi mmx fxsr sse sse2 ss ht tm pbe syscall nx pdpe1gb rdtscp lm constant_tsc arch_perfmon pebs bts rep_good nopl xtopology nonstop_tsc aperfmperf eagerfpu pni pclmulqdq dtes64 monitor ds_cpl vmx smx est tm2 ssse3 fma cx16 xtpr pdcm pcid sse4_1 sse4_2 x2apic movbe popcnt tsc_deadline_timer aes xsave avx f16c rdrand lahf_lm abm 3dnowprefetch ida arat epb pln pts dtherm tpr_shadow vnmi flexpriority ept vpid fsgsbase tsc_adjust bmi1 hle avx2 smep bmi2 erms invpcid rtm rdseed adx smap xsaveopt
bugs :
bogomips : 5187.85
clflush size : 64
cache_alignment : 64
address sizes : 39 bits physical, 48 bits virtual
power management:

The details of the CPU which is used in Television environment are as follows.

$ cat /proc/cpuinfo
processor : 0
model name : ARMv7 Processor rev 1 (v7l)
BogoMIPS : 2002.94
Features : swp half thumb fastmult vfp edsp neon vfpv3 tls vfpv4 idiva
idivt
CPU implementer : 0x41
CPU architecture: 7
CPU variant : 0x0
CPU part : 0xc0d
CPU revision : 1

processor : 1 model name : ARMv7 Processor rev 1 (v7l)
BogoMIPS : 2002.94
Features : swp half thumb fastmult vfp edsp neon vfpv3 tls vfpv4 idiva

idivt
CPU implementer : 0x41
CPU architecture: 7
CPU variant : 0x0
CPU part : 0xc0d
CPU revision : 1

processor : 2
model name : ARMv7 Processor rev 1 (v7l)
BogoMIPS : 2002.94
Features : swp half thumb fastmult vfp edsp neon vfpv3 tls vfpv4 idiva
idivt
CPU implementer : 0x41
CPU architecture: 7
CPU variant : 0x0
CPU part : 0xc0d
CPU revision : 1

processor : 3
model name : ARMv7 Processor rev 1 (v7l)
BogoMIPS : 2002.94
Features : swp half thumb fastmult vfp edsp neon vfpv3 tls vfpv4 idiva
idivt
CPU implementer : 0x41
CPU architecture: 7
CPU variant : 0x0
CPU part : 0xc0d
CPU revision : 1

Hardware : monaco
Revision : 0020
Serial : 0000000000000000