SUPER-RESOLUTION MOSAICKING FROM DIGITAL SURVEILLANCE VIDEO CAPTURED BY UNMANNED AIRCRAFT SYSTEMS (UAS)

by

Aldo Camargo, B.S.E.E., M.S.S.E.

A Dissertation

Submitted to the Graduate Faculty

of the

University of North Dakota

in partial fulfillment of the requirements

for the degree of

Doctor of Philosophy

Grand Forks, North Dakota
August
2010

This dissertation, submitted by Aldo Camargo in partial fulfillment of the requirements for the Degree of Doctor of Philosophy Ph.D. from the University of North Dakota, has been read by the Faculty Advisory Committee under whom the work has been done and is hereby approved.

_____
Chairperson

_____

_____

_____

_____

This dissertation meets the standards for appearance, conforms to the style and format requirements of the Graduate School of the University of North Dakota, and is hereby approved.

_____
Dean of the Graduate School

_____
Date

PERMISSION

Title Super-resolution Mosaicking from Digital Surveillance Video Captured by Unmanned Aircraft Systems (UAS)

Department Electrical Engineering

Degree Doctor of Philosophy

In presenting this dissertation in partial fulfillment of the requirements for a graduate degree from the University of North Dakota, I agree that the library of this University shall make it freely available for inspection. I further agree that permission for extensive copying for scholarly purposes may be granted by the professor who supervised my thesis work or, in his absence, by the chairperson of the department or the dean of the Graduate School. It is understood that any copying or publication or other use of this thesis or part thereof for financial gain shall not be allowed without my written permission. It is also understood that due recognition shall be given to me and to the University of North Dakota in any scholarly use which may be made of any material in my thesis.

Signature _____

Date _____

TABLE OF CONTENTS

LIST OF FIGURES

Figure Page

LIST OF TABLES

Table Page

LIST OF ACRONYMS

LR ..... Low Resolution

SR..... Super-Resolution

SD ..... Steepest Descent

CG ..... Conjugate Gradient

LM..... Levenberg Marquardt

CPU..... Central Processing Unit

GPU..... Graphical Processing Unit

SVM..... Singular Value Decomposition

SIFT ..... Scale-Invariant Feature Transform

RANSAC ..... Random Sample Concensus

UAS..... Unmanned Aircraft System

UASE ..... Unmanned Aircraft Systems Engineering

TV ..... Total Variation

CV ..... Cross -Validation

GCV ..... Generalized Cross-Validation

GPS ..... Global Positioning System

IMU..... Inertial Measurement Unit

HMRF ..... Hubert Markov Random Field

ACKNOWLEDGMENTS

First at all, I would like to thank God for all his blessing that he always gave me and keep giving me.

I wish to thank Dr. Richard R. Schultz for all his support, understanding, teaching, mentoring and guidance in the development of my research and the completion of my Ph.D. studies at the University of North Dakota.

I wish to thank my father Walter, my mother Vilma, and brothers: Eduardo, Enzo, Wendy, Richard, Fernando and Giancarlo for their support, understanding and prays.

I also want to wish to thank all of the professors of the Department of Electrical Engineer at the University of North Dakota, for their teaching and time that they spent with me. Also, I would like to thank Dr. Jeremiah Neubert and Dr. William H. Semke from the Mechanical Engineering Department, and Dr. Ryan Zerr and Dr. Mike Minnotte of Mathematics Department, Dr. Mike Poellot from Atmospheric Sciences Department and Dr Ronald A. Fevig from the Space Studies Deparment at the University of North Dakota.

The author also would like to recognize the students of the School of Engineering and Mines for their proficiency and comradeship, especially the contributions of the Unmanned Aircraft Systems Engineering (UASE) Laboratory team.

This research was supported in part by the FY2006 Defense Experimental Program to Stimulate Competitive Research (DEPSCoR) program, Army Research

Office grant number 50441-CI-DPS, Computing and Information Sciences Division, "Real-Time Super-Resolution ATR of UAV-Based Reconnaissance and Surveillance Imagery," (Richard R. Schultz, Principal Investigator, active dates June 15, 2006, through June 14, 2010). This research was also supported in part by the Joint Unmanned Aircraft Systems Center of Excellence contract number FA4861-06-C-C006, "Unmanned Aerial System Remote Sense and Avoid System and Advanced Payload Analysis and Investigation," as well as the North Dakota Department of Commerce grant, "UND Center of Excellence for UAV and Simulation Applications."

To my wife Jenny, my daughter Angela Sofia and son Angelo Mathias.

ABSTRACT

Mosaicking refers to the stitching of one or more correlated images, forming a much larger image of a scene. Super-resolution mosaicking refers to methods for enhancing the resolution of the mosaic, which can be affected by different sources of noise, as well as other effects such as camera translation and rotation. Methods to compute super-resolution mosaics use a low-resolution mosaic as an input. The mosaic can be generated from a panoramic view of a scene, digital video, satellite terrain imagery, surveillance footage, or images from many other sources.

Unmanned Aircraft Systems (UAS) can be used for tracking and surveillance by exploiting the information captured by a digital imaging payload. Some of the most significant problems facing surveillance video captured by a small UAS aircraft (i.e., an airframe with a payload carrying capacity of less than 50 kilograms) include motion blur; the frame-to-frame movement induced by aircraft roll, wind gusts, and less than ideal atmospheric conditions; and the noise inherent within the image sensors. These effects have to be modeled to create a super-resolution mosaic from low-resolution UAS surveillance video frames, so that effective image analysis can be conducted. The goal of this dissertation is to perform super-resolution mosaicking of surveillance video captured by a UAS digital imaging payload, which involves recovering a high-resolution map of the region under surveillance using accurate camera and motion models with minimal computation for near-real-time operation.

This dissertation focuses on spatial domain methods based on image operators and iterated back-projection methods. We use a novel framework which does not require the construction of sparse matrices, efficient, robust, is independent (it constructs the super-resolution mosaic by itself from only video information), and is easy to implement. The results obtained in our simulations shows a great improvement of the resolution of the low resolution mosaic of up to 47.54 dB for synthetic images, and a great improvement in sharpness and visually details, for real UAS surveillance frame, in only ten iterations.

Steepest descent, conjugate gradient and Levenberg Marquardt are used to solve the nonlinear optimization problem involved in the computation of super-resolution mosaic. A comparision in computation time and improvement in the resolution is peformed. The algorithm used for Levenberg Marquardt avoid the computation of the inverse of the pseudo Hessian matrix by solving a linear square problem using singular value decomposition (SVD).

The use of the graphical processing unit (GPU) paradigm is used to speed up super-resolution mosaicking. Since, the registration step takes most of the time in feature based methods, we reduce this time by computing the SIFT features, matching and homography in the GPU. The remaining steps are performed over the CPUs (central processing unit).The speed up factor for the computation of the homography, used extensively in image registration and placement, is more than fifty times faster than using only CPUs.

CHAPTER 1

INTRODUCTION

This dissertation investigates sets of frames captured from UAS surveillance video, in which feature overlaps can be used to create a large image containing the entire view, with more resolution and details. The name for such techniques is *super-resolution mosaicking*. Using this, it is possible to extend the field of view beyond that of any single frame. The aim of this dissertation is to develop near-real-time, efficient, robust, independent, and automated frame super-resolution mosaiking with applications to UAS surveillance video.

An essential step required to construct the super-resolution mosaic is image registration. The SIFT (Scale Invariant Feature Transform)[92] together with the RANSAC (Random Sample Consensus) [93], are used to estimated the homography, which gives us the image registration between two consecutive frames. But, SIFT takes a great deal of computational resources making the image registration slow, so it becomes the bottleneck for the computation of near-real-time super-resolution mosaics. For that reason, the graphical processing unit (GPU) is used to compute the image registration, showing a considerable speed up.

Super-resolution mosaicking involves the understanding of both the generation of video mosaics and super-resolution reconstruction. Both of these areas have been studied by many researchers, and Chapter 2 will review the most important approaches. Most of these approaches have focused on small images (fewer number of pixels to process) and

synthetic images (images created with certain known parameters of motion, blur, and scaling). Conversely, this dissertation focuses on real video captured from a small UAS platform flown by the Unmanned Aircraft Systems Engineering (UASE) Laboratory at the University of North Dakota.

1.1 Overview of the Dissertation

Following this introduction, Chapter 2 presents some of the background theory necessary for a formal definition of the mosaicking and super-resolution problems. This chapter reviews the different approaches in the frequency and spatial domains.

Chapter 3 reviews the different stochastic and deterministic regularization techniques used for the numerical computations involved in super-resolution reconstruction. Throughout this dissertation, the concept of the inverse problem and especially of the ill-posed inverse problem will recur. For this reason, a brief review of this interesting subject is provided.

Chapter 4 details the construction of image and video mosaics. The use of MPEG I-frames are also detailed. Due to the fact that most of the real applications for video or image mosaicking for UAS also refer to the term "geo-referencing," Chapter 4 details the construction of geo-referenced mosaics based on the information of three different sensors: GPS (Global Positioning Unit), IMU (Inertial Measurement Unit), and video frames. The fusion of these data is done using the unscented Kalman Filter (UKF) because of its generally good performance for non-linear systems.

Chapter 5 details the construction of super-resolution mosaics by three different algorithms: steepest descent, conjugate gradient, and Levenberg Marquardt. All of these algorithms use a novel model to represent the super-resolution mosaic, where the

construction of the mosaic is represented by image operators. To solve the ill-posed inverse problem, a Huber prior is used. Also, the Lagrange multiplier is found using a robust method that does not require the construction of sparse matrices. The simulations were performed on both synthetic data, to be able to compute the PSNR qualitatively, and real frames captured by UAS to compare the results visually. Finally, a comparison between all three algorithms is shown; this comparison is based on visual quality and computation time.

Chapter 6 has three parts. The first part involves a short explanation of the GPU paradigm, and the second part explains the construction of video mosaics using MPEG I-frames implemented over GPU-CPU. The results demonstrate that it is possible to perform real-time video mosaicking with today's hardware. Finally, the last part explains the construction of super-resolution mosaicking using GPU-CPU and a comparison with the results using only CPU.

CHAPTER 2

BACKGROUND

2.1 Introduction

Super-resolution mosaicking involves many definitions and previous concepts to understand. This chapter provides some definitions of super-resolution and video mosaicking. An attempt is made to provide a perspective on how modern super-resolution reconstruction and image mosaicking techniques have evolved from the beginning to the present.

Section 2.2 provides a definition of super-resolution reconstruction and a high-level overview of the different approaches to solve it. Mainly, there are two approaches: 1) frequency-domain and 2) spatial-domain. Additionally, a brief definition of the ill-posed inverse problem and how regularization plays an important role is briefly described.

Section 2.3 explores the different techniques to construct a mosaic, the difference between static and dynamic mosaicking is shown, and how the accumulation of errors due to the projection model affects the construction of the mosaic.

2.2 Super-resolution reconstruction

Super-resolution reconstruction refers to methods for still image and video enhancement from multiple low-resolution, degraded observed images derived from an underlying scene [7], see Figure 1. The goal is to obtain a single image or video with better quality. There are two different categories of approaches: super-resolution in the

spatial-domain [8-12] and super-resolution in the frequency-domain [13,14], based on the motion estimation between consecutive frames. Frequency-domain super-resolution relies on motion vectors being comprised purely of horizontal and vertical displacements, which for real video data from UAS is almost never realistic. The frequency-domain is effective in making use of low-frequency components to register a set of images containing artifacts. The results with this type of approach generally have ringing effects.

Spatial-domain super-resolution methods use the image registration between frames by computing the feature correspondence in the spatial domain. The motion models can be global for the entire image or local for a set of corresponding feature vectors [62]. In order to understand the super-resolution problem, equation (2.1) represents the observation model that relates the original high-resolution (HR) image to the observed (i.e, low-resolution LR) images. Considering the desired HR image of size $L_1N_1 \text{x} L_2N_2$ written in lexicographical notation as the vector $\mathbf{x} = [x_1, x_2, ..., x_N]^T$, where $N = L_1N_1 \text{x} L_2N_2$. The down-sampling factors in the horizontal and vertical directions are represented by $L_1 \text{x} L_2$, respectively. Thus, each observed LR image is of size $N_1 \text{x} N_2$. Let the k^{th} LR image be denoted in lexicographic notation as $\mathbf{y}_k = [y_{k,1}, y_{k,2}, ..., x_M]^T$, for $k = 1,2,..., p$ and $M = N_1 \text{x} N_2$ [15]. Assuming $\mathbf{x}$ (HR image) remains constant during the acquisition of multiple LR images, the model is:

$$\mathbf{y}_k = \mathbf{D}\mathbf{B}_k\mathbf{M}_k\mathbf{x} + \boldsymbol{\eta}_k \text{ , for } 1 \le k \le p, \tag{2.1}$$

where $\mathbf{M}_k$ represents the warp matrix of size $L_1N_1L_2N_2 \text{x} L_1N_1L_2N_2$, $\mathbf{B}_k$ represents an $L_1N_1L_2N_2 \text{x} L_1N_1L_2N_2$ blur matrix, $\mathbf{D}$ is an $(N_1N_2)^2 \text{x} L_1N_1L_2N_2$ down-sampling matrix

and $\boldsymbol{\eta}_k$ represents a lexicographically-ordered noise vector. Figure 2 shows a block diagram for the observation model of equation (2.1).

The motion matrix represented by $\mathbf{M}_k$ may contain global or local translations, rotations, and so on. Since this information is unknown, it is necessary to estimate the scene motion for each frame with respect to one particular frame, called the reference frame. The warping process is defined in terms of LR pixel spacing; therefore is necessary to interpolate the pixels to represent them on the HR grid.

The blurring effect can be caused by many factors such as atmospheric blur, motion, and camera blur [16]. Most of the approaches model the LR sensor with a point spread function (PSF). This PSF is usually modeled as a spatial averaging operator or as a 2D – Gaussian.

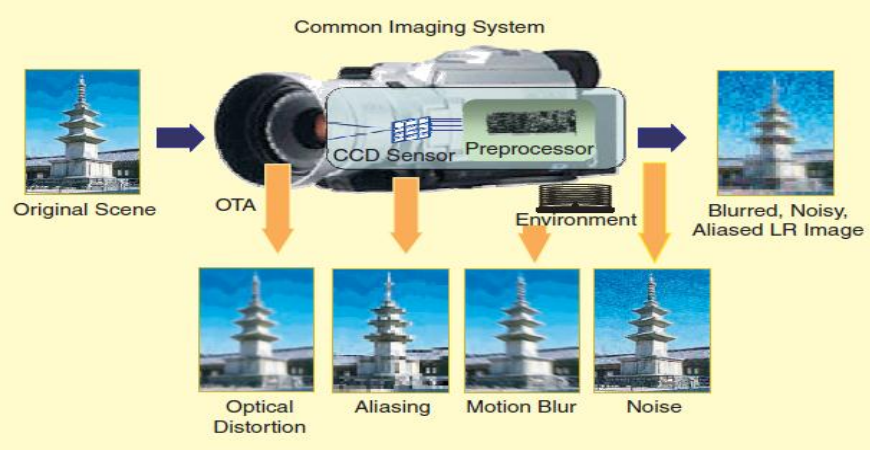


Figure 1. Different degradations of the HR image to create an LR image. Figure taken from [15].

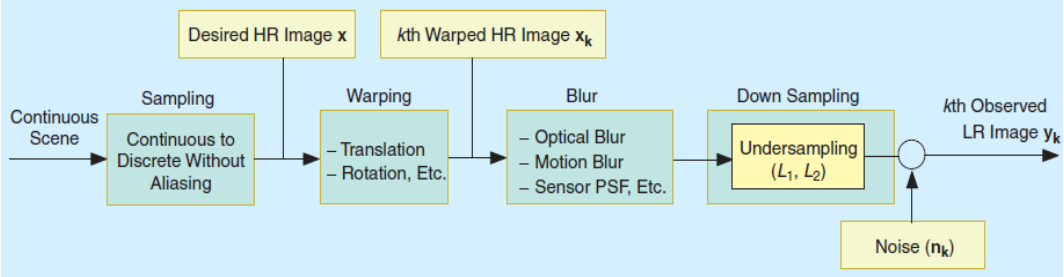


Figure 2. Block diagram of equation (2.1). Figure taken from [15].

The matrix $\mathbf{D}$ generates aliased LR images from the warped and blurred HR image. Figure 4 shows the effects of down-sampling and up-sampling over 9x9 and 3x3 images, respectively. In this dissertation, we assume that the blurring effects of the CCD are captured by the blur matrix $\mathbf{B}_k$, and therefore the CCD down-sampling process can be modeled by a simple periodic sampling of the high-resolution image. Thus, the corresponding up-sampling process is implemented as a zero-filling process.

Equation (2.1) can be represented as

$$\mathbf{y}_k = \mathbf{H}_k\mathbf{x} + \boldsymbol{\eta}_k \text{ , for } 1 \le k \le p, \tag{2.2}$$

where $\mathbf{H}_k$ represents the effect of the decimation, blurring and warping. This matrix $\mathbf{H}_k$ is of size $(N_1N_2)^2 \text{x} L_1N_1L_2N_2$.

Based on (2.2), the aim of SR reconstruction is to estimate the HR image $\mathbf{x}$ from the LR images $\mathbf{y}_k$ for $k = 1,..., p$. Therefore, SR is an inverse problem.

According to Hadamard [17] an inverse problem is consider well-posed when a solution:

1. exists for any data,
2. is unique, and
3. depends continuously on the data.

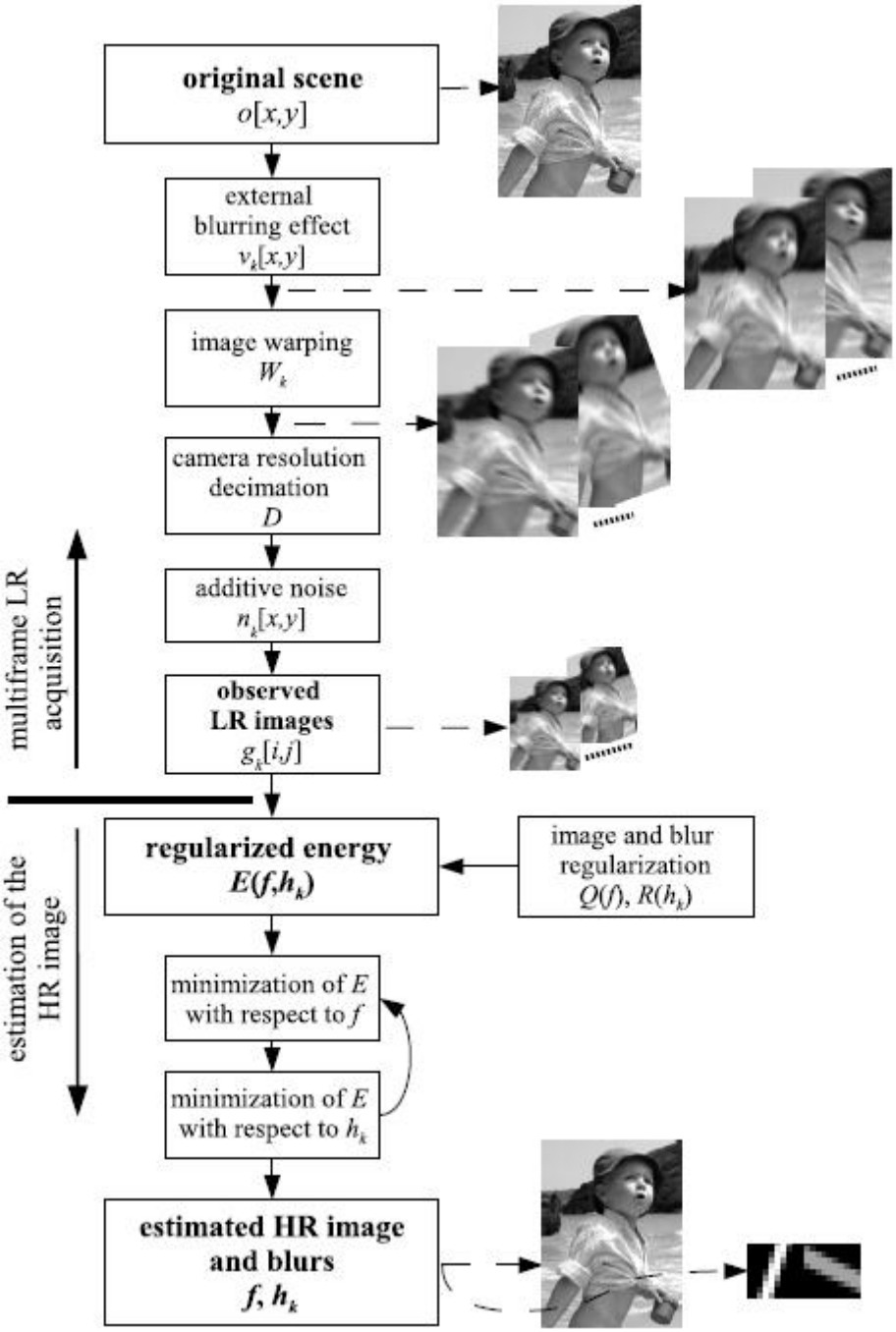


Figure 3. Low resolution and reconstruction flow. Figure taken from [45].

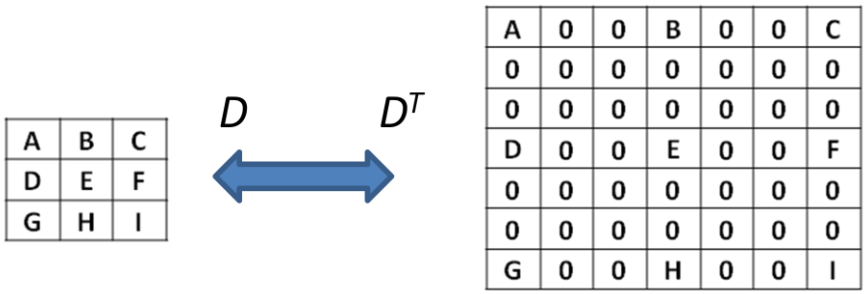


Figure 4. Down-sampling $\mathbf{D}$ and Up-sampling ($\mathbf{D}^T$) effects on a 9x9 and 3x3 image, respectively.

If any of these conditions are not satisfied, then the inverse problem is ill-posed. For the case of SR, the solution is not unique to (2.1), so SR is an ill-posed inverse problem.

Regularization refers to methods which are used to add additional information to compensate for the loss of information in the ill-posed problems. This additional information is typically referred to as *a priori* or prior information. This prior information cannot be derived from the observations or the observation process and must be known "before the fact." Normally, the prior information is chosen to represent desired characteristics of the solution, e.g., smoothness, total energy, edge preservation, etc. The role of this prior information is to reduce the space of solutions which are compatible with the observed data. A review of different regularization methods will be provided in Chapter 3.

2.2.1 Frequency-domain methods

These methods make explicit use of the aliasing that exists in each LR image to reconstruct an HR image [15,18]. The frequency-domain approach is based on the following three principles: i) the shifting property of the Fourier transform, ii) the

aliasing relationship between the continuous Fourier transform (CFT) of an original HR image and the discrete Fourier transform (DFT) of observed LR images, and iii) the assumption that the original HR image is bandlimited.

Tsai and Huang [18] rely on the motion being composed purely of horizontal and vertical displacements, Tom and Katsaggelos [20, 21] take a two-phase super-resolution approach, where the first step is to register, deblur, and de-noise the low-resolution images, and the second step is to interpolate and integrate them into a high-resolution image grid. Figure 5 shows one LR synthetic image data and the SR image result of the algorithm proposed by Tom and Katsagegelos.

Figure 5. Example of frequency-domain approach for super-resolution taken from Tom and Katsaggelos [20, 21]. Left: One of the four synthetic LR images. Right: SR image. There are several ringing artifacts, particularly along the image edges, but there is also a distinct improvement in the resolution of the image.

2.2.2 Spatial-domain methods

Spatial-domain methods actually perform better with additive noise, and a more natural treatment of the image point spread blur in cases where it cannot be approximated by a single convolution operation on the HR image [19].

The first spatial-domain methods were developed by Peleg [23], Keren [22], and Irani [24]. Peleg et al. [23] highlighted the use of subpixel motion to improve resolution. Keren [22] proposed a method to register two different images. This registration finds the translation and rotation within the plane of the image, but it generally fails with resampling and interpolation. Irani [24,25] used the same algorithm proposed by Keren, but proposed a more sophisticated method for super-resolution image recovery based on back-projection.

Later work by Zomet et al. [26] proposed the use of medians to deal with large outliers caused by the parallax of moving specularities. Projection onto Convex Sets (POCS), which is set-theoretic approach to super-resolution, was used by Stark and Oskoi [27] that utilizes a maximum likelihood (ML) framework and also prior information; Patti et al. [32], as well as Elad and Feuer [28,29,30] use Kalman filtering to pose the problem in an easy way to solve.

Figure 6 shows an example of the results using the method proposed by Zomet [26]. The left image is one of the LR images and the right image is the SR estimated image. Figure 7 shows four LR images from a set of 12 LR images and also the SR image using POCS.

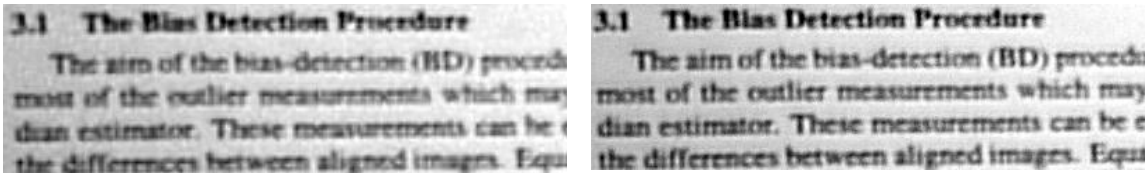


Figure 6. Example of spatial-domain approach for super-resolution taken from Zomet et al. [26]. Left: One of the input LR images. Right: SR image.

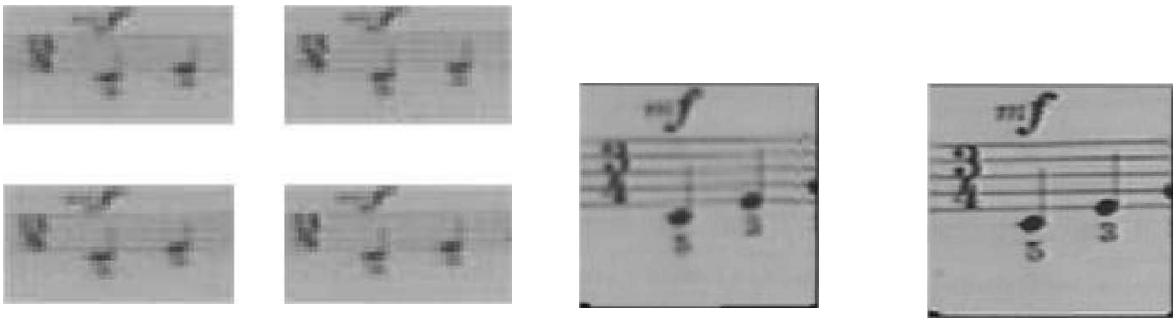


Figure 7. Example of POCS super-resolution taken from Patti et al. [32]. Left: Four of 12 low-resolution images. Middle: Interpolated approximation to the high-resolution image. Right: Super-resolution image found using POCS technique in [32].

2.2.3 Methods of Solution

One of the most important concerns in the solution of the ill-posed inverse super-resolution problem is the cost of the computation, and how quickly it converges to a unique optimal solution.

For the initial approaches based on frequency-domain least-squares (i.e., of format $\mathbf{Ax} = \mathbf{b}$), the super-resolution estimate is found using an iterative re-estimation process. However, the method proposed by Irani [24] generates different solutions depending on the initial guess.

The ML estimator is explored by Capel [31], where the SR image is estimated directly by using the pseudoinverse. Since this is another convex problem, the algorithm is guaranteed to converge to the same global optimum whatever the initial condition.

Maximum a *posteriori* (MAP) is one of the preferred methods. Some approaches can be re-interpreted as MAP because they use a regularized cost function whose terms can be matched to those of a posterior distribution over a high-resolution image, as the regularization term can be viewed as a type of image prior. If a prior over the high-resolution image is chosen so that the log prior distribution is convex in the image pixels

and the basic ML solution itself is also convex, then the MAP solution will have a unique optimal super-resolution image.

A popular form of convex regularizer is a quadratic function of the image pixels, $\|\mathbf{Ax}\|_2^2$, for some matrix $\mathbf{A}$ and image pixel vector $\mathbf{x}$. If the objective function is taken as the exponential argument, it can be manipulated to give the probabilistic interpretation, because a term with the form $\exp\left\{-\frac{1}{2}\|Ax\|_2^2\right\}$ is proportional to a zero-mean Gaussian prior over $\mathbf{x}$ with covariance $\mathbf{A}^T\mathbf{A}$.

Schultz and Stevenson [33,34] look at video sequences with frames related by dense corrrespondence found using a hierarchical block-matching algorithm. They use the Huber Markov Random Field (HMRF) as a prior to regularize the super-resolution image recovery. The Huber function is quadratic for small values of input, but linear for larger values, so it penalizes edges less severely than a Gaussian prior. This Huber function models the statistics of real images more closely than a purely quadratic function, because real images contain edges. Therefore, they have much heavier-tailed first-derivative distributions than can be modeled by a Gaussian. Figure 8 shows one of the LR images on the left and the SR image on the right using the Schultz and Stevenson [33,34] method.

Figure 8. Example of MAP with Huber Markov Random Field (HMRF) prior extracted from Schultz and Stevenson [33,34]. Left: One of input LR images. Right: SR image.

The total variation (TV) prior and a related technique called the bilateral filter was used by Farsiu et al. [35, 36, 37,38]. They introduced a regularization term called Bilateral-TV, which is inexpensive to implement and also preserves edges. Furthermore, they explore several ways to formulate quick solutions by working with L_1 norms, rather than the more common L_2 norms to solve the super-resolution problem. Figure 9 shows one of their results using the TV prior and the bilateral filter from [35, 36, 37,38].

Capel and Zisserman [39,40] compare the back–projection model of Irani and Peleg to simple spatial-domain ML approaches, and show that these perform much less well on a text image sequence than the HMRF method and the Total Variation (TV) estimator. Also, they consider super-resolution as a second step after image mosaicking, where the image registration (using a homography with eight degrees of freedom) is carried out in the mosaicking process.

Figure 9. Example of TV prior and bilateral filter from Farsiu et al. [35, 36, 37, 38]. Left: One of the input LR images. Right: SR image.

Baker and Kanade [41,42] analyze the sources of noise and poor reconstruction in the ML case, by considering various forms of the PSF and their limitations. Their proposed method works by partitioning the low-resolution space into a set of classes, each of which has a separate prior model. For example, if a face is detected in the low resolution image set, a face-specific prior over the super-resolution image will be used. The classification of the low-resolution images is made using a pyramid of multi-scale Gaussian and Laplacian images, which were built up from training data.

Generalized cross-validation (GCV) was proposed Nguyen in his dissertation [43]. GCV is used to compute the regularization factor used in the solution of the ill-posed super-resolution problem. GCV works well for overdetermined, underdetermined, and square systems. GCV is simple cross-validation applied to the original system after it

has undergone a unitary transformation. GCV is also known to be less sensitive to large outliers than cross-validation [43].

Šroubek and Flusser [45, 46] developed an alternating minimization scheme based on a maximum *a posteriori* (MAP) blind deconvolution with a prior distribution of blurs derived from the multichannel framework and a prior distribution of original images. This method combines the benefits of edge preserving denoising techniques and the one-step subspace eigenvector-based method (EVAM) reconstruction method. Figure 10 shows one of the LR images on the left and the SR image with the estimated PSF on the right using the deconvolution method proposed by Šroubek and Flusser [45, 46].

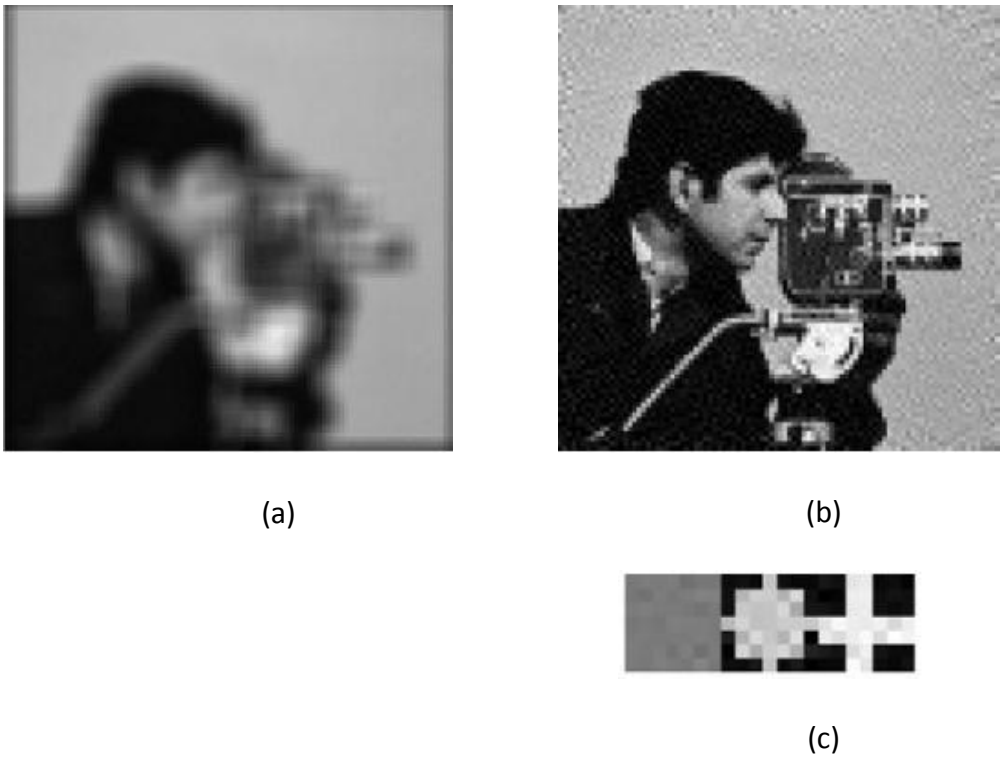


Figure 10. Estimation of the cameraman image and blurs taken from [45]: (a) Degraded image. (b) Result from the blind deconvolution algorithm [45], (c) Estimated PSF.

Tian and Ma [47] proposed a Markov Chain Monte Carlo (MCMC) algorithm with outlier-sensitive bilateral filtering. The idea of MCMC is to generate N samples

with $p(\mathbf{X}/\mathbf{Y})$. The number of samples has to be large enough to guarantee the convergence of MCMC. They use a bilateral filter to reduce the noise effect within the pixels. Figure 11 shows an example of the computation of SR using the MCMC.

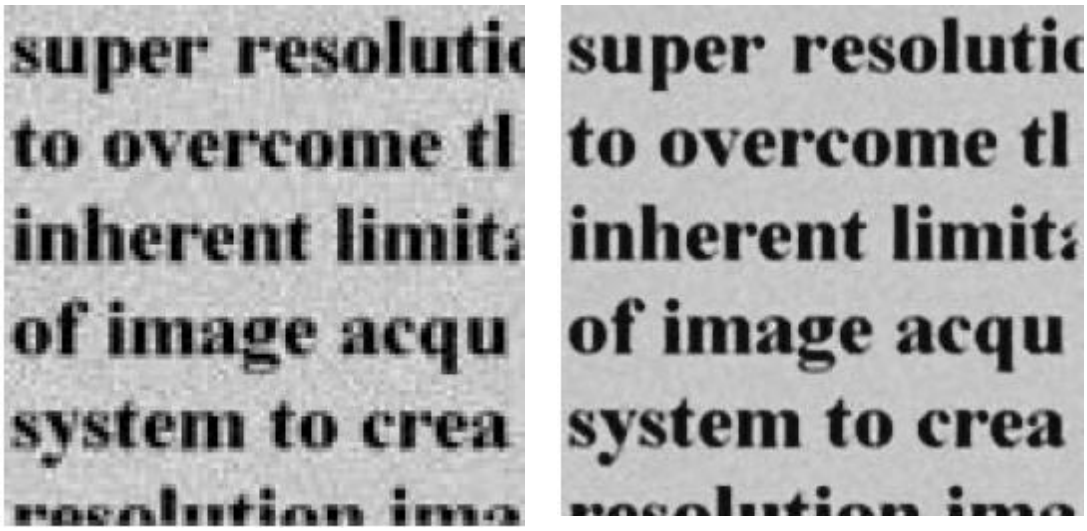


Figure 11. Super-resolution using MCMC and the bilateral filter taken from [47]. Left: One of the low resolution images. Right: Super-resolution of the text image using the MCMC and the outlier-sensitive bilateral method.

Pickup [19] studies the different effects of the geometric and photometric registration challenges related to super-resolution. Also, she proposed a model that finds both the blur and the super-resolution image. This model is Bayesian based and leads to a direct method of optimizing the super-resolution image pixel values, resulting in better SR images. Furthermore, she introduces a texture-based prior for super-resolution using MAP. Figures 12 and 13 show some results obtained with the method proposed by Pickup [19].

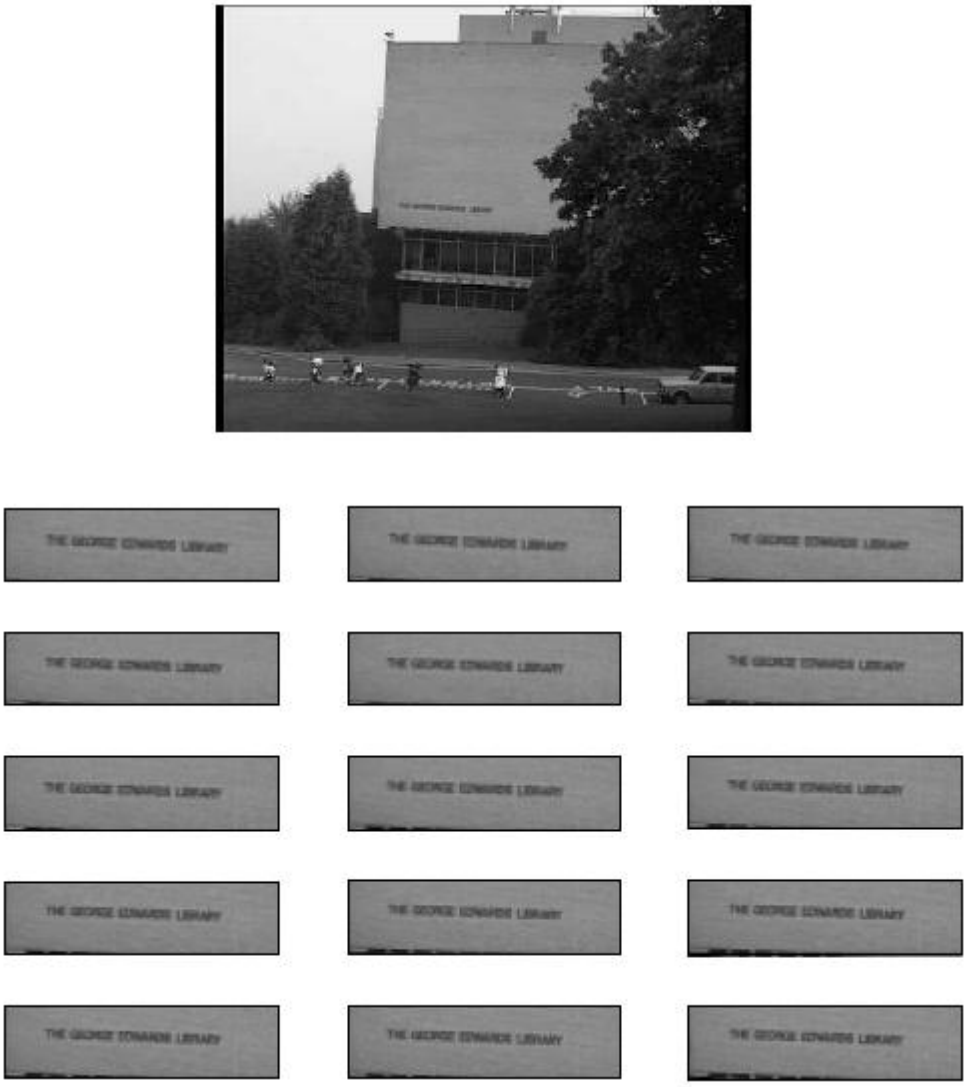


Figure 12. Super-resolution algorithm proposed by Pickup [19]: Top: One of the 30 original low-resolution frames. Bottom: Every second input from the sequence, showing a cropped region of interest.

Figure 13. Super-resolution algorithm proposed by Pickup [19]. Left column shows the super-resolution images computed using a standard MAP approach, where the geometric and photometric registration parameters are estimated and frozen before the high-resolution pixel values are optimized. The right column shows the results using the proposed MAP approach of [19], where both the pixels and registration values are found simultaneously.

2.3 Image Mosaicking

Image mosaicking is the alignment (i.e., stitching) of multiple images into larger compositions which represent portions of a 3D scene [31]. For the construction of the mosaic, the camera needs to take different views by panning, tilting, or zooming. In order to build the mosaic, it is necessary that images be warped, using computed homographies, into a common coordinate frame, and combined to form a single image. The basic steps to construct a mosaic are 1) registration, 2) reprojection, and 3) blending. For registration, this finds the homography between consecutive frames, in the case of digital video. To find the homography, it is necessary to find robust features that will then be matched with the similar features in the next frame or images. Reprojection warps all the frames to a simple coordinate system. To do that, it is necessary to choose a

frame as a reference frame. Blending consist of eliminating the vignetting parallax effects. Figure 14 illustrates these three steps to create a mosaic.

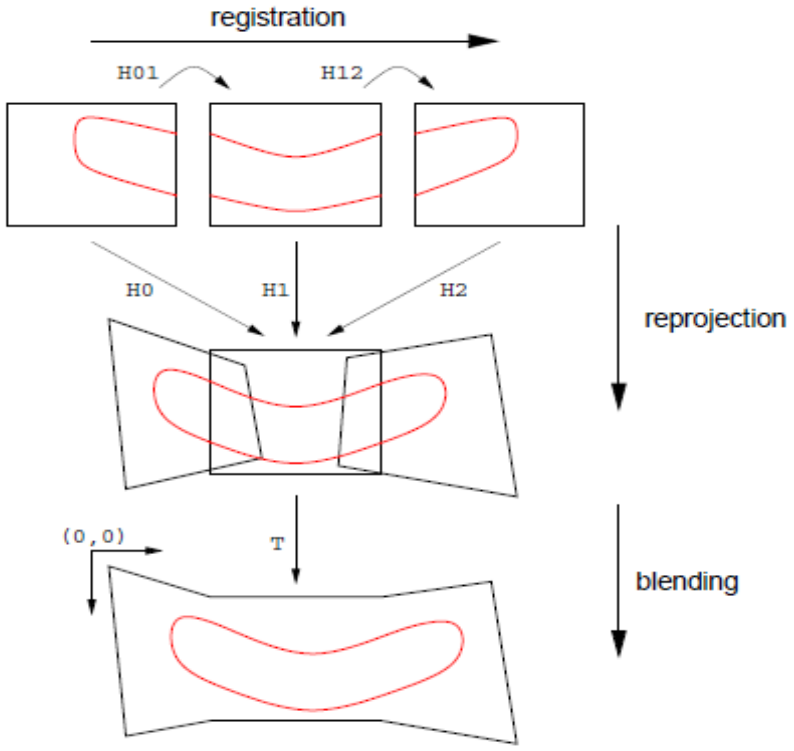


Figure 14. Basic steps to construct a mosaic, taken from [31]. 1) Registration: consists of finding the homography between consecutive frames. 2) Reprojection: consists of warping all the frames into a common coordinate system. 3) Blending: consists of eliminating parallax effects.

Irani et al. [48, 49, 50] reviews image mosaicking and its many applications: video compression, video enhancement, and enhanced visualization, as well as other applications in video indexing, search, and manipulation. Furthermore, she constructs and analyzes two different types of mosaic: 1) static mosaic, where the input video is usually segmented into contiguous scene subsequences, and the mosaic is constructed for each scene subsequence; and 2) dynamic mosaic, where the mosaic captures the dynamic

changes in the scene. Figure 15 shows both static and dynamic mosaics taken from [48, 49, 50].

Peleg et al. [51,52] consider mosaics composed of strips extracted from the input images. The strips are chosen such that the direction of optical flow is orthogonal to the axis of the strip. By doing this, and with a suitable blending, it is possible to make approximate mosaics for situations including camera translation. Figure 16 shows one example of the mosaic using the method proposed by Peleg [51,52].

Kan and Szeliski [53] proposed constructing a mosaic composed of a hemisphere of an image to represent the view in every direction at a particular point in the world. They construct the mosaic at many points, and match the image features across the mosaics to perform a wide-baseline 3D scene reconstruction. Szeliski [54] constructs mosaic using 2D transformations and depth information. The intention is to use the creation of the mosaic to recover a full 3D model, which has many applications including 3D model acquisition for inverse CAD, model acquisition for computer animation and special effects, virtual reality, etc. Figure 17 shows an example of the construction of mosaics based on depth information for virtual reality proposed by Kan and Szeliski [53].

Figure 15. . Static and dynamic mosaicking taken from [48]. Top: Construction of the static mosaic using the temporal median of a baseball game sequence. Bottom: Construction of a dynamic mosaic of a baseball sequence.

Figure 16. Panoramic mosaic using manifold projection [51].

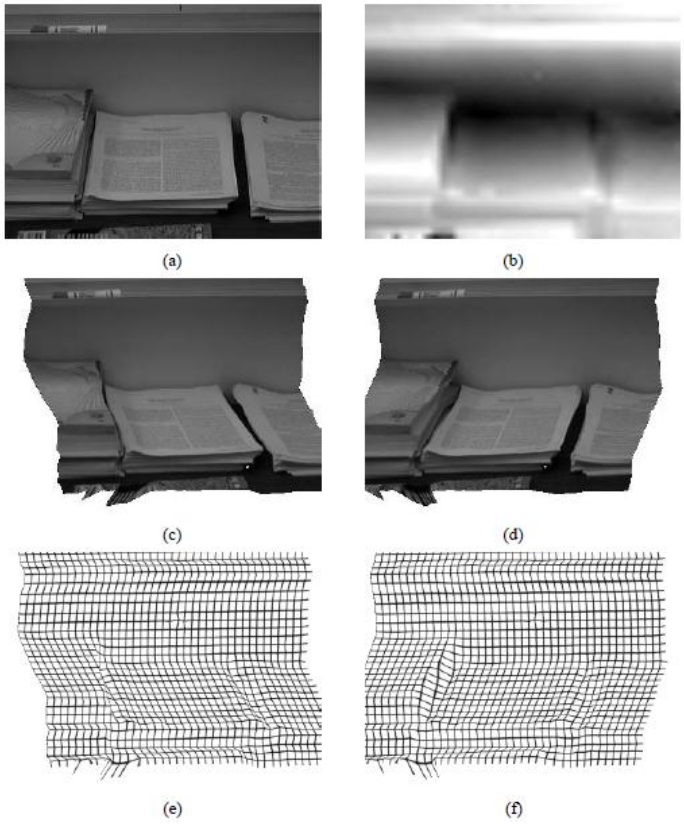


Figure 17. Depth recovery example taken from [54]. Table with a stack of papers (a) as an input image taken by moving the camera up and over the scene. (b) The resulting depth-map as intensity-coded range values. (c-d) show the original intensity image texture mapped onto the surface. (e-f) show a set of grid-lines overlayed on the recovered surface.

Brown and Lowe [55] proposed a method to construct mosaic panoramas without the help of human input. They use SIFT (Scale Invariant Features Transform) to select the features within the images that are then matched using the RANSAC algorithm. They use a probabilistic model to verity the match. Bundle adjustment based on the Levenberg Marquardt algorithm is then used eliminate the accumulation of errors. Finally, Multi-band blending is used. Figure 18 shows one of the results of the mosaic construction using the method proposed by Brown and Lowe [55].

Capel [31] proposed a novel algorithm for an efficient matching of features across multiple views which are related by projective transformations. Also, he proposed a new method to reduce the effect of the projective distortion for two and *N*-view cases. Figure 19 shows the pre-image point X, which generates interest points x_1, x_2, and x_3 in three different views. The distances d_1, d_2, and d_3 are to be minimized with respect to the homographies H_1, H_2, and H_3 and the point X.

Figure 18. Final mosaic taken from [55]. This mosaic was constructed using 80 images matched using SIFT (Scale Invariant Feature Transform), rendered in spherical coordinates, and blended using the multi-band technique.

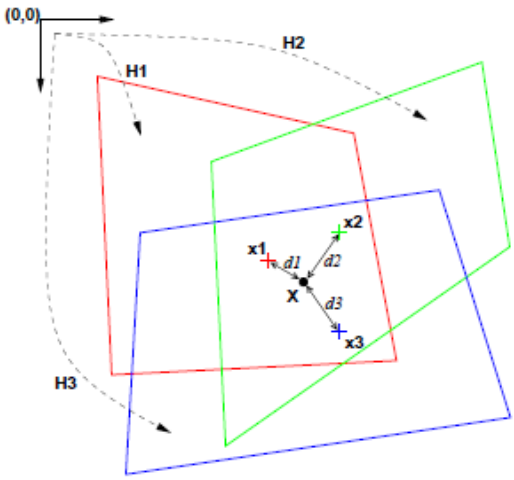


Figure 19. Solution to the problem of error accumulation proposed by Capel [31].

Figure 20 shows a comparison of the close-up views between the region of interest (red box) and the real region extracted from a single frame in the sequence. It is easy to see a clear mismatch between the first and the last frames in the sequence, caused by the accumulation of error in the construction of the mosaic.

Figure 21 shows the result of the mosaic construction after refinement of the homographies by bundle-adjustment using the Levenberg Marquardt algorithm. The mismatch presented in Figure 20 has been removed.

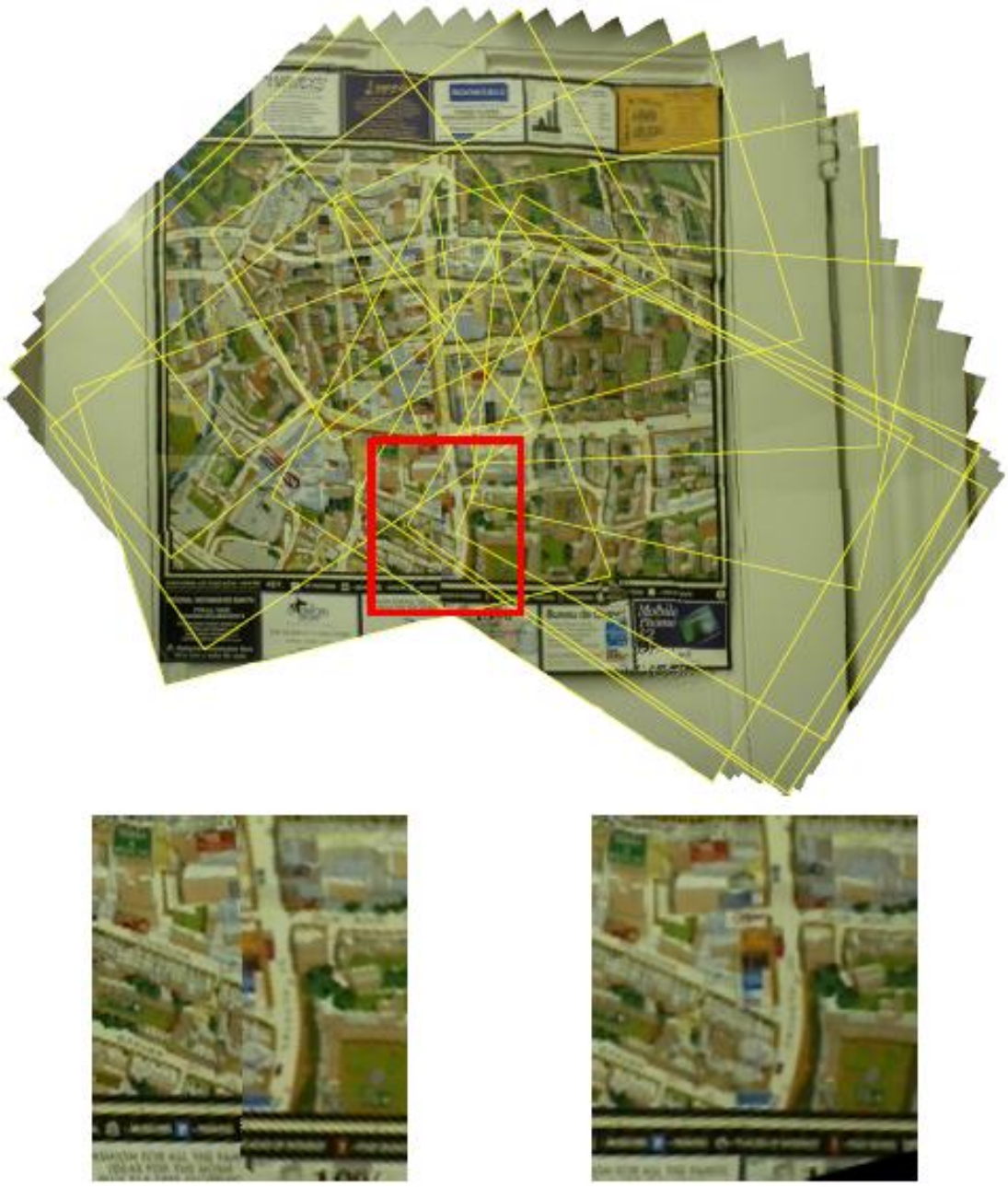


Figure 20. Top: A mosaic image obtained from [31]. The outlier of every 5th frame is overlaid. Left: A close-up view of the region of interested (red box). Right: The corresponding region extracted from a single frame.

Figure 21. Top: A mosaic image after refinement of the homography by bundle-adjustment, obtained from [31]. Left: A close-up view of the region of interest (red box of the Figure 20). Right: The corresponding region extracted from a single frame.

2.4 Conclusion

This section presented a summary of the most recent and important approaches super-resolution reconstruction and image mosaicking. Scenarios for super-resolution frequency-domain and spatial-domain reconstruction are. The spatial-domain methods

are the most appropriate for real UAS video frames, but they require great deal of computational resources. Most of the spatial-domain approaches require the construction of a sparse matrix to represent equation (2.2), so the problem is converted in the solution of a non-linear sparse problem.

The motion estimation is a key component for super-resolution reconstruction and image mosaicking. Most of the first approaches for super-resolution use direct methods to find the motion vectors. These methods, like block matching, often fail at object edges or are susceptible to parallax effects (optical flow). Conversely, feature-based methods are: 1) invariant to a wide range of photometric and geometric transformations of the image; 2) robustness to outliers, because by using RANSAC (Random Sampling Algorithm and Consensus), the outliers are rejected and not taking into consideration to find the homography. The advantage of direct methods over feature-based methods is computational efficiency, because a carefully implementation of them has proved successful in real-time tracking applications.

CHAPTER 3

STOCHASTIC AND DETERMINISTIC REGULARIZATION FOR SUPER-RESOLUTION

3.1 Introduction

Mathematical background about regularization is presented in this chapter. Section 3.2 explains the ill-posed and ill-conditioned inverse problems. Different approaches for regularization are shown in Section 3.3.

There has been much research to solve linear ill-posed inverse problems stably, especially the Fredholm integral equations of the first kind [43]. These equations can be expressed as:

$$\int_{\Omega} h(s,t)x(t)dt = f(s), s \in \Omega \subset R^q, \tag{3.1}$$

with $h(.,.) \in L^2(\Omega x \Omega)$. If $h(s,t)$ is translation invariant, then $h(s,t) = h(s-t)$ and (3.1) becomes a convolution equation with kernel $h(s)$:

$$f(s) = \int_{\Omega} h(s-t)x(t)dt \tag{3.2}$$

$$= h(s) * x(s) \tag{3.3}$$

In this case, * denotes the convolution operator. Let $H : L^2(\Omega) \to L^2(\Omega)$ be the linear convolution operator $Hx = h * x$. Then H is a compact operator with the singular value expansion,

$$H\upsilon_j = \sigma_j u_j, H^* u_j = \sigma_j \upsilon_j, \qquad j = 1,2,..., \tag{3.4}$$

where H^* is the adjoint operator, σ_j is a non-increasing sequence of positive singular values u_j, υ_j are the corresponding singular functions. Now, if we expand the right-hand side, such that

$$f = \sum \eta_j u_j, \qquad \eta_j = (u_j, f), \tag{3.5}$$

with $(.,.)$ representing the inner product, the solution $x = H^{-1} f$ converges only if f satisfies the Picard condition.

In practice, the right-hand side of f contains noise, (i.e., from the camera sensors, for the case of super-resolution) and modeling errors. This problem is considered an ill-conditioned inverse problem, since a small change in f can result in a wild oscillation approximation to x

3.2 Ill-posed and Ill-conditioned Inverse Problems

According to Keller [58], an inverse problem is defined as: "We call two problems *inverses* of one another if the formulation of each involves all or part of the solution of the other. Often, for historical reasons, one of the two problems has been studied extensively for some time, while the others have never been studied and not so well understood. In such cases, the former is called a *direct problem*, while the latter is the *inverse problem.* "

Borman [7] provides a historical solution of the heat equation. The problem is, given an initial temperature distribution at time $t = t_0$, determine the evolution of the temperature profile for times $t > t_0$. Consider, however, the following: assume that the temperature profile at time $t = t_f > t_0$ is provided. The challenge is to determine the original temperature profile at the earlier time $t = t_0$. This is the inverse problem of the

direct heat equation. It turns out, however, that while the direct problem is easily solved, the inverse problem is not.

According to Hadamard's requirements, solving the direct heat equation meets all three requirements, so it is well-posed. But, for the case of inverse problem, that of determining the initial temperature distribution given the final temperature distribution, turns out to be highly problematic. The problems are intimately related to the irrecoverable loss of information. This irrecoverable loss of information does not present significant difficulties for the direct problem. In particular, the loss of information implies that there exist a multiple of initial temperature distributions which could give rise to an observed temperature distribution at time $t > t_0$. Therefore, since the inverse problem fails to have a unique solution (Hadamard's second requirement), it is an ill-posed problem.

3.3 Regularization

Regularization is a term which refers to methods that utilize additional information to compensate for the information loss in the ill-posed problems. This additional information is typically referred as *a priori* or prior information, and adds prior knowledge about the desired estimate to make the ill-posed problem well-posed. Tikhonov [59] was the pioneer in introducing deterministic theory of regularized solutions to ill-posed problems. Tikhonov regularization is a deterministic technique which restricts the solution space, using a metric to distinguish between possible solutions.

3.3.1 Tikhonov Regularization

In the Tikhonov approach, a family of approximate solutions to the inverse problem is constructed, with the family of solutions controlled by a nonnegative real-valued *regularization parameter*. Recall equation (2.2) from Chapter 2, which represents the super-resolution problem. Equation (3.6) can be rewritten as (3.7), representing a more general equation for all the images or frames:

$$\mathbf{y}_k = \mathbf{H}_k\mathbf{x} + \boldsymbol{\eta}_k \text{ , for } 1 \le k \le p, \tag{3.6}$$

$$\begin{bmatrix} y_1 \\ . \\ . \\ . \\ y_p \end{bmatrix} = \begin{bmatrix} \mathbf{H}_1 \\ . \\ . \\ . \\ \mathbf{H}_p \end{bmatrix} \mathbf{x} + \begin{bmatrix} \boldsymbol{\eta}_1 \\ . \\ . \\ . \\ \boldsymbol{\eta}_p \end{bmatrix} \tag{3.6.a}$$

$$\mathbf{Y} = \mathbf{H}\mathbf{x} + \boldsymbol{\eta} \tag{3.7}$$

In order to obtain a reasonable estimate for $\mathbf{x}$, we need to regularize that equation. For noisy, over-determined systems we search for solutions to fit the noisy data, such that

$$\min_x \left\|\mathbf{H}\mathbf{x} - \mathbf{Y}\right\|_2^2 + \lambda\left\|L\mathbf{x}\right\|_2^2 \tag{3.8}$$

where L is a regularization operator, λ is related to the Lagrange multiplier, and $||.||_2$ represents the Euclidean (L_2) norm. The first term of (3.8) ensures that the estimated solution has small residuals, and the second term ensures "*well-behaved*" solutions.

The Lagrange multiplier allows for a balance between the two requirements. If λ is too large, the regularized system is too far from the original equation. But, if it is

too small, the system behaves as an ill-conditioned problem. Figure 22 illustrates this behavior of the Lagrange multiplier [63]. For the case of an under-regularized problem, the solution is overwhelmed with noise and registration artifacts. But, for the case of an over-regularized problem, the solution smooths out the final output. The matrix L can also include prior knowledge of the problem, e.g., degree of smoothness [60].

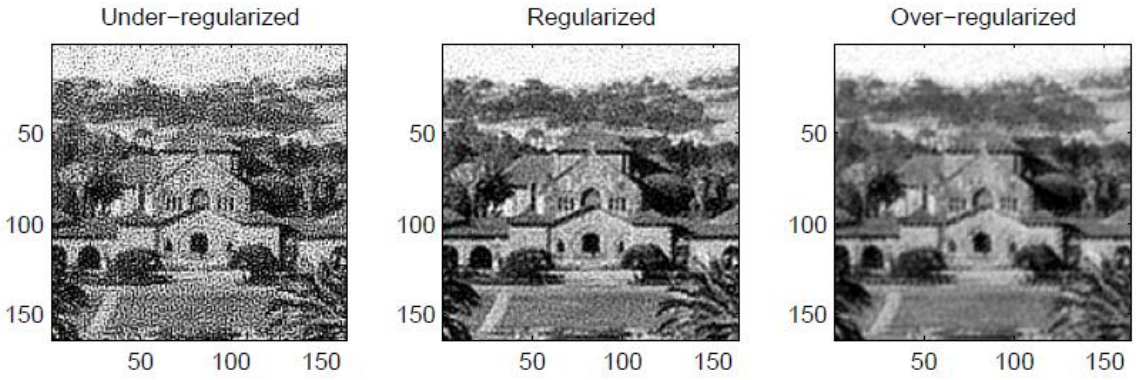


Figure 22. The importance of the Lagrange multiplier in the regularization to solve super-resolution [43].

Now, taking the derivatives of (3.8) and setting them to 0, we obtain

$$\mathbf{x}(\lambda) = (\mathbf{H}^T\mathbf{H} + \lambda L^T L)^{-1}\mathbf{H}^T\mathbf{Y} \tag{3.9}$$

A common assumption is that the images are primarily smooth. Tikhonov proposed a generic stabilizer based on the m^{th} order Sobolev norm [61], which conveys the assumption of function continuity. $\mathbf{H}$ is also called the linear compact injective operator between Hilbert spaces U and F. The solution of $\mathbf{x}$ and data $\mathbf{Y}$ belongs to U and F, respectively. In the context of low-level vision problems, the first-order Tikhonov stabilizer is called a thin plate [62]. Both elements are "stretched" across the data, and their minimum states provide the estimates.

3.3.2 Total Variation (TV) Regularization

Let Ω_L be a subset of $\Omega \subset R^2$, and define $\mathbf{Y}$ as a real function over Ω_L. Also, assuming that the high resolution images are those whose domain is Ω, we have

$$\hat{\mathbf{x}} = \arg\min_x \left\{ \int |\nabla \mathbf{x}| + \sum_{k=1}^{N} \frac{\lambda_k}{2} \left[\left\| \mathbf{Y} - \mathbf{H}\mathbf{x} \right\|_2^2 - \sigma_k^2 \right] \right\}, \tag{3.10}$$

where σ_k is the variance of the white noise with zero mean, and λ_k represents the Lagrange multipliers for every low-resolution image.

The model expressed by (3.10) solves a more general problem of super-resolution using the total variation (TV) norm as the regularizing function, allowing homogeneous Newmann boundary conditions.

3.3.3 Cross-Validation (CV)

The idea of cross-validation (CV) to choose the Lagrange multiplier λ from the data is simple. To estimate λ, the data is divided into two sets: one set is used to construct an approximate solution based on λ, and the other is used to measure the error of that approximation [63]. For example, the validation error by using the j^{th} pixel value as the validation set is

$$CV_j(\lambda) = \left\| h_j \mathbf{x}_j(\lambda) - y_j \right\|_2^2. \tag{3.11}$$

The optimal regularization parameter λ_{CV} minimizes the total validation error:

$$\lambda_{CV} = \arg\min_{\lambda} \sum_{j-1}^{K} CV(\lambda) \tag{3.12}$$

3.3.4 Generalized Cross-Validation (GCV)

Generalized Cross-Validation (GCV) is simply CV applied to the original system after it has undergone a unitary transformation. Also, it is known to be more robust to outliers than CV [64]. For overdetermined systems, it has been shown that the asymptotically optimum regularization parameter according to GCV is given by [65]:

$$\lambda_{GCV} = \arg\min_{\lambda} \frac{\left\| \left(\mathbf{H}\mathbf{H}^T + \lambda \mathbf{I}\right)^{-1} \mathbf{Y} \right\|_2}{tr((\mathbf{H}\mathbf{H}^T + \lambda \mathbf{I})^{-1})} \tag{3.13}$$

GCV is used for calculating regularization parameters for Tikhonov-regularized overdetermined and underdetermined least squares problems [43].

3.3.5 Bilateral-TV

Based on the TV (Total Variation) criterion and the bilateral filter [65,66], the bilateral TV is based on both of these methods and is found by

$$\lambda_{BTV}(\mathbf{X}) = \sum_{l=0}^{P} \sum_{m=0}^{P} \alpha^{m+l} \left\| \mathbf{X} - S_x^l S_y^m \mathbf{X} \right\|_{L1}, \tag{3.14}$$

where the matrices (operators) S_x^l and S_y^m shift $\mathbf{x}$ by l and m pixels in the x and y directions, respectively. The scalar α is a weight between 0 and 1. The parameter "*P*" defines the size of the corresponding bilateral filter kernel [67].The BTV regularization preserves edges and is less computationally expensive than Tikhonov regularization.

3.3.6 Huber Prior

The Huber function is used as a simple prior for image super-resolution, which benefits from penalizing edges less severely than Gaussian image priors. The form of the prior is

$$p(x) = \frac{1}{Z} \exp\left\{ -\nu \sum_{g \in D(x)} \rho(g, \alpha) \right\}, \tag{3.15}$$

where D is a set of gradient estimates, given by $D(x)$ [19]. The parameter v is a prior strength somewhat similar to a variance term, Z is the normalization constant, and α is a parameter of the Huber function specifying the gradient value at which the penalty switches from being quadratic to linear:

$$\rho(g,\alpha) = \begin{cases} x^2 & ,if\ |x| \le \alpha \\ 2\alpha|x| - \alpha^2, & otherwise \end{cases} \tag{3.16}$$

Figure 23 shows different Huber functions and their corresponding distributions, note that the value of α determines the behavior of the Huber function, and controls the overall shape of the edge-preserving function. v controls the behavior of distributions:

$$p(x) = \frac{1}{Z}\exp\{-v\rho(x,\alpha)\} \tag{3.17}$$

By integrating (3.16) Z can be expressed as

$$Z = \frac{1}{v\alpha}\exp\{-2v\alpha^2\} + \left(\frac{\pi}{v}\right)^{\frac{1}{2}} erf\{\alpha v\}. \tag{3.18}$$

One important feature of why Huber prior is one of the most prior used is because makes the problem convex, therefore most of the optimization algorithms can converge to a local minima.

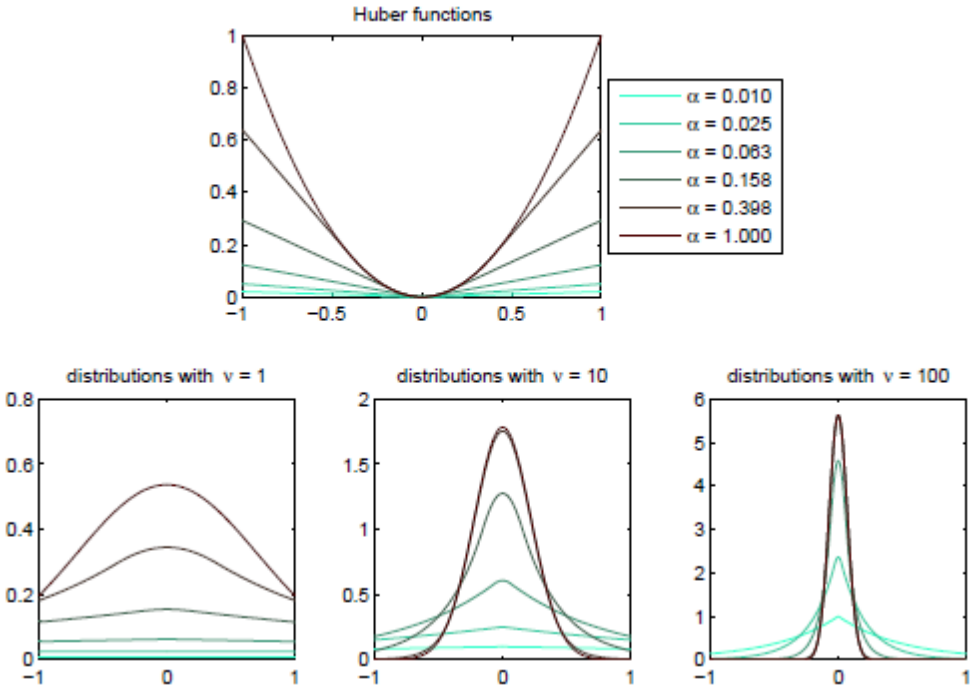


Figure 23. Huber function and corresponding distributions [19]. Top: Several functions corresponding to a set of logarithmically-spaced values. Bottom: Three sets for ν = 1, ν = 10, and ν = 100, each of them using the set of Huber functions.

3.3.7 Spatially Adaptive Prior

The objects in most images have edges with coherently varying pixel intensities. The pixel-scale intensity differences alone are not sufficient to characterize objects of multiple scales. Thus, continuous texture information within a larger scale should be used to discriminate information from singularities or noise. This is the basis of the Spatial Adaptive (SA) prior model. SA uses a large nonlocal neighborhood *N* to incorporate geometrical configuration information [68]. The SA prior can be formalized as follows:

$$U_{SA}(f) = \sum_j U_j(f) = \sum_j \sum_{b \in N_j} (w_{bj}(fb - fj)^2 / |N_{rj}|) \tag{3.18}$$

$$w_{bj} = \begin{cases} 1, & if \quad dis_{bj} \le \delta \\ 0, & otherwise. \end{cases} \tag{3.19}$$

$$dis_j = \left\| n_b(f) - n_j(f) \right\|_E^2 = \sum_l (f_{l \in n_b} - f_{l \in n_j})^2 \tag{3.20}$$

$$n_b(f) = \{ f_l : \quad l \in n_b \} \tag{3.21}$$

$$n_j(f) = \{ f_l : \quad l \in n_j \} \tag{3.22}$$

In this case, U_{SA} is the energy function for the SA prior, w_{bj} represents the classification of the neighbor pixels in the search neighborhood N_j, $|N_{rj}|$ is the number of neighbor pixels with nonzero w_{bj} in the neighborhood N_j, and δ is the threshold parameter. The value of the distance dis_{bj} is determined by a distance measurement between the two translated neighborhoods n_b and n_j, respectively.

3.4 Conclusions

The problem of super-resolution has the form of the Fredholm integral equation of the first kind. This chapter explains why super-resolution is an inverse ill-posed problem and presents the different regularization techniques to solve it. The reason why Huber prior is preferred in super-resolution reconstruction is also explained.

CHAPTER 4

MOSAICKING AND GEO-REFERENCING

4.1 Introduction

This chapter describes in detail the process of constructing of a video dynamic mosaic and also its geo-referencing from image coordinates to world coordinates. Section 4.2 explains the construction of the image mosaic based on the computation of the homography between consecutive images. This homography is computed using SIFT, because of its robustness. Examples using real data from frames obtained in flight tests of the Unmanned Aircraft Systems Engineering (UASE) Laboratory at the University of North Dakota are shown. Section 4.3 explains the construction of a video mosaic using MPEG video, and results using real data are shown. Finally, Section 4.4 explains the construction of a geo-referenced mosaic based on the Unscented Kalman Filter (UKF).

4.2 Image Mosaicking

There are three general steps for the construction of an image mosaic: (1) registration, (2) reprojection, and (3) blending. In the following sections these steps are described in more detail. Image mosaicking is the alignment of multiple images into a larger composition which represents portions of a 3D scene [31]. The mosaic method used in this dissertation is concerned with images that can be registered by a planar homography: views of a planar scene from a camera that has a rotation and a translation.

4.2.1 Registration

Registration is a fundamental task in digital video processing, especially for mosaicking and super-resolution, where we need sub-pixel accuracy. The need to register images has arisen in many practical problems: (1) integrating information taken from different sensors, (2) finding changes in images taken at different times and/or in different conditions, (3) inferring 3D information from images, and (4) object and target recognition. Registration methods can be viewed as different combinations of choices for the following four components: (a) a feature space, (b) a search space, (c) a search strategy, and (d) similarity metric.

The *feature space* use a sparse set of corresponding image features (e.g., points, or lines) to estimate the image-to-image mapping. The *search space* is the class of transformations that are capable of aligning the images. The *search strategy* decides how to choose the next transformation from this space, which will be tested in the search for the optimal transformation. The *similarity metric* determines the relative merit for each test.

This dissertation uses feature-based registration and planar homography, which is the mapping that arises in the perspective image of planes. There are two important situations where the image-to-image mapping is exactly captured by a planar homography: images of a plane viewed by a camera rotating about its optic center and/or zooming, which is the typical case for UAS surveillance imaging. These two situations are illustrated in Figures 24 and 25. Furthermore, the homography is appropriate for this dissertation due to a camera viewing a distant scene, such as is the case for UAS

surveillance imaging. For all cases, it is assumed that the images are obtained by a perspective pin-hole camera.

A point is represented by homogeneous coordinates, so that point (x, y) is represented as $(x, y, 1)$. However, the point (x_1, x_2, x_3) in homogeneous coordinates corresponds to the inhomogeneous point $(x_1 / x_3, x_2 / x_3)$. Under a planar homography (called also plane projective, collineation, or projectivity), those points are mapped as [69]:

$$\begin{pmatrix} x_1' \\ x_2' \\ x_3' \end{pmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} \tag{4.1}$$

$$\mathbf{x}' = \mathbf{H}\mathbf{x} \tag{4.2}$$

The matrix $\mathbf{H}$ is called homogeneous, because this matrix can be multiplied by a factor (scale) without altering the projective transformation. There are eight independent ratios among the nine elements of $\mathbf{H}$.

There are many methods to find the homography, which are grouped in two ways: (1) direct correlation methods and (2) feature-based methods. As was mentioned before, this dissertation will use feature-based methods to find the registration parameters.

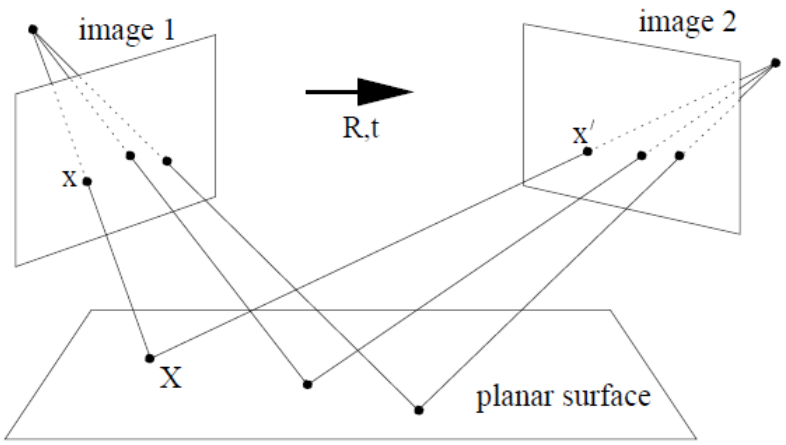


Figure 24. Images of planes. There is a planar homography between two images of a plane taken from different viewpoints, related by a rotation $\mathbf{R}$ and translation t. The scene point X is projected to point x and x' in image 1 and image 2, respectively. These points are related by $x' = \mathbf{H}x$.

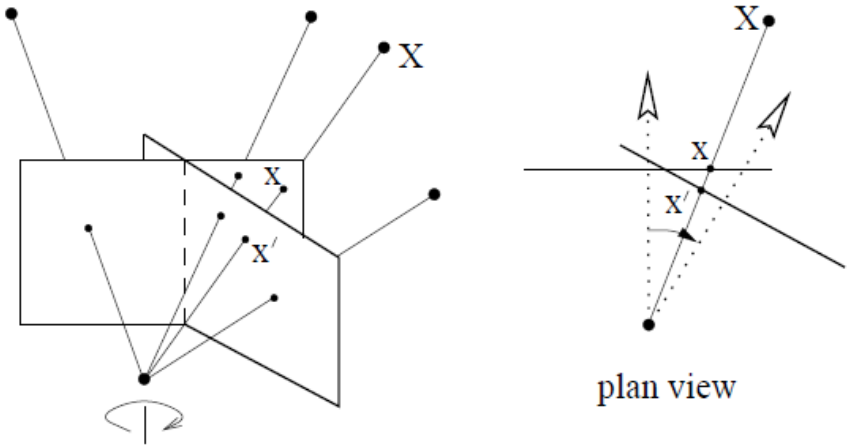


Figure 25. Rotation about the camera axis. As the camera is rotated, the points of intersection of the rays with the image plane are related by a planar homography. Image points x and x' correspond to the same scene point X. Points are related by $x' = \mathbf{H}x$.

4.2.2 SIFT and RANSAC to Estimate the Homography

There are many ways to find the features within an image, but according to [92] SIFT (Scale-Invariant Feature Transform) features are robust to photometric and geometric changes within two consecutive frames. Figure 27 shows different evaluations for feature descriptors: changes of viewpoint, scale changes combined with image rotation, image rotation only, image blur, JPEG compression, and illumination changes.

SIFT descriptors are computed for normalized image patches. A descriptor is a 3D histogram of gradient location and orientation. The location is quantized into a 4x4 location grid, and the gradient angle is quantized into eight orientations. The resulting descriptor is of dimension 128. Figure 26 illustrates this approach.

Each orientation plane represents the gradient magnitude corresponding to a given orientation. In order to obtain illumination invariance, the descriptor is normalized by the square root of the sum of squared components.

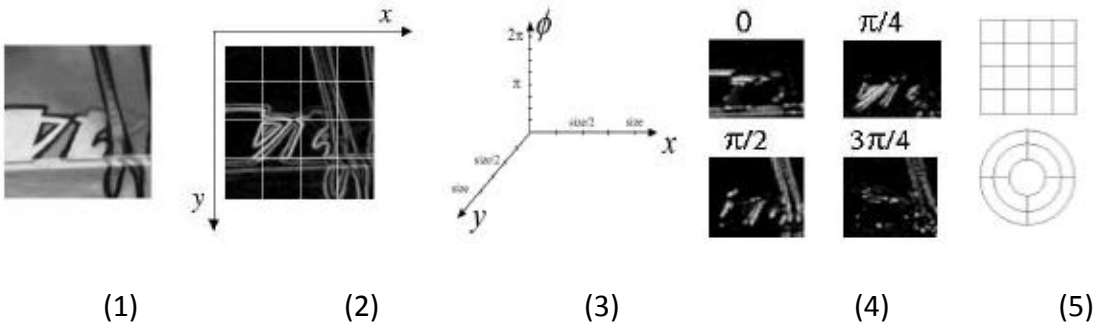


Figure 26. SIFT descriptor: (1) detected region, (2) gradient image and location grid, (3) dimensions of the histogram, (4) shows four of eight orientation planes, and (5) Cartesian and log-polar location grids.

The SIFT corner detector consists of the following four steps [62]:

1. **Scale-space extreme detection:** The scale-space extreme are the maximum and minimum of the difference-of-Gaussian images. They are potential distinctive features, detected by comparing a pixel with its 26 neighboring pixels in the current and adjacent scales.
2. **Key point localization:** The key points are selected through a second-order Taylor series expansion. Given the difference-of-Gaussian $D(x, y, \sigma)$, its Taylor expansion is given as

$$D(\mathbf{x}) = D(0) + \frac{\partial D^T}{\partial x}\mathbf{x} + \frac{1}{2}\mathbf{x}^T \frac{\partial^2 D^T}{\partial x^2}\mathbf{x} \tag{4.4}$$

   In the extreme position, we have $\frac{\partial D(x)}{\partial x} =,0$ and for $\mathbf{x}$ as

$$\hat{\mathbf{x}} = -\left(\frac{\partial^2 D}{\partial \mathbf{x}^2}\right)^{-1} \frac{\partial D}{\partial \mathbf{x}} \ . \tag{4.5}$$

3. **Orientation assignment:** One or more orientations are assigned to each key point location by selecting the peaks of the smoothed histogram of local gradients.
4. **Key point descriptor:** There are eight bins for a 4 x 4 gradient window around each key point. Thus, each key point can be described as a 128-dimensional vector.

Mikolajczyk et al. [73] compared different descriptors under different tests. The conclusion of their research was that SIFT is the best feature descriptor at the moment. Figure 27 shows the results of some tests taken from [73].

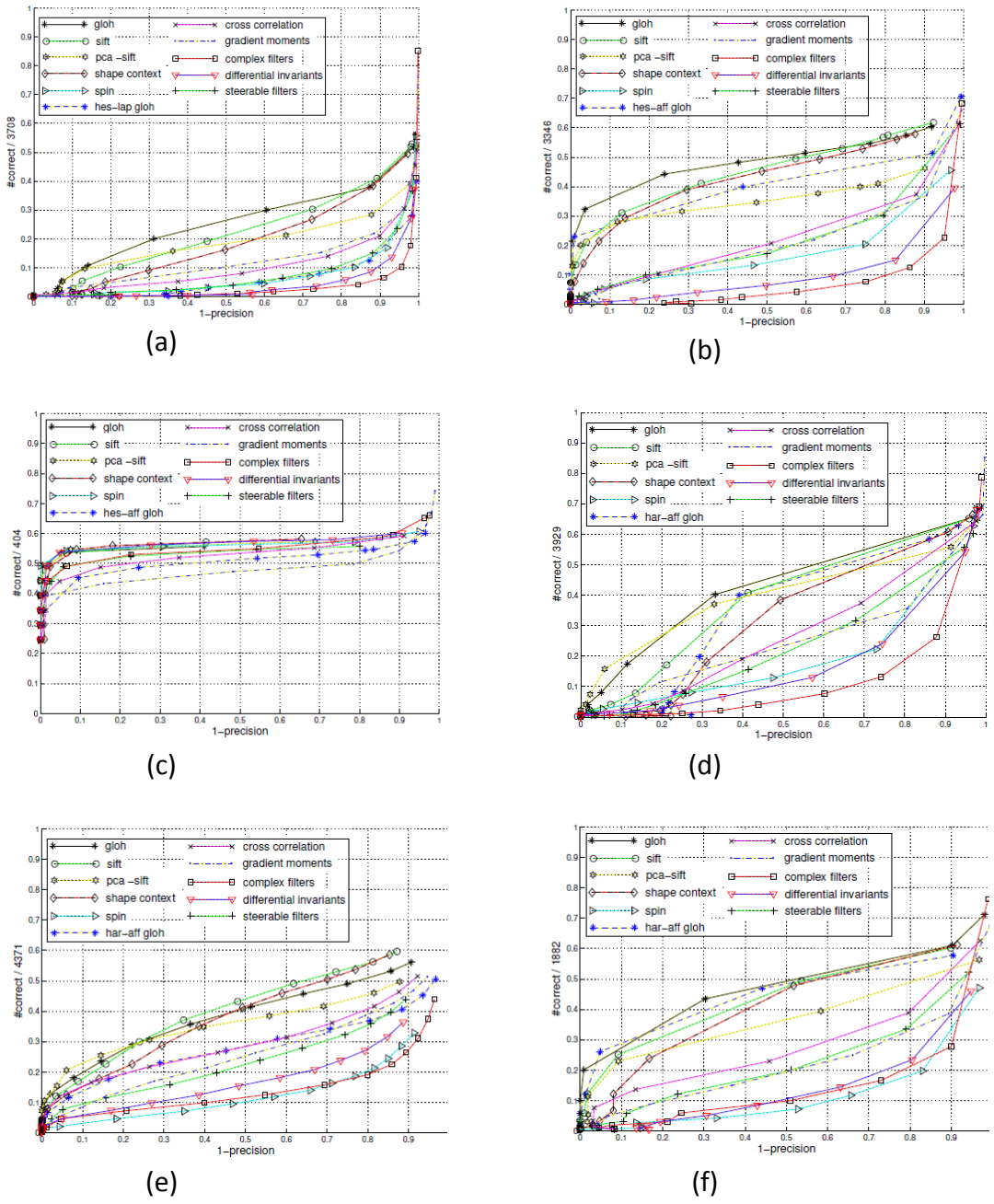


Figure 27. Evaluation of features descriptors for structured scene [73]: (a) viewpoint changes 40^o-60^o. (b) Scale changes of a factor 2-2.5 combined with image rotation of 30^o – 45^o, (c) image rotation of 30^o – 45^o, (d) image blur, (e) JPEG compression, and (f) illumination changes.

SIFT features are located at scale-space maxima/minima of the difference of Gaussian function, which gives a similarity-invariant frame. This allows edges to shift

slightly without altering the descriptor of the vector, giving robustness to affine changes. After extracting the features of the first two frames, it is necessary to extract the first set of matching features between them. The next section describes this process. Obviously for video mosaicking, the SIFT features are computed for every frame.

Figure 28 shows the SIFT features for five different frames taken from an infrared (IR) camera during flight tests 2007 by the UASE Laboratory at the University of North Dakota. The numbers of SIFT features for these frames are, in the order that they appear: frame 1: 228, frame 2: 293, frame 3: 361, frame 4: 387, and frame 5: 387.

4.2.3 Image Feature Matching and Homography Estimation

Once the SIFT features have been found within the input images, the next step is to find the matching features between consecutive frames. Given a set of features in one frame, we find the matching features in the other frame by finding the image's approximate k nearest neighbors in a kd tree using Best Bin First Search [85].

We use the full homography because is more accurate than the affine and Euclidean homography [64]. RANSAC (random sample consensus) [63] is used to find the homography, because of its robustness and minimal set of randomly sampled correspondences to estimate image transformation parameters. A solution is computed that has the best consensus with the data. Four correspondences is the minimum number necessary between two frames to instantiate the model defined by least-squares to find the planar homography $\mathbf{H}$. We repeat this *N* times until we obtain an error less than 0.01 pixels.

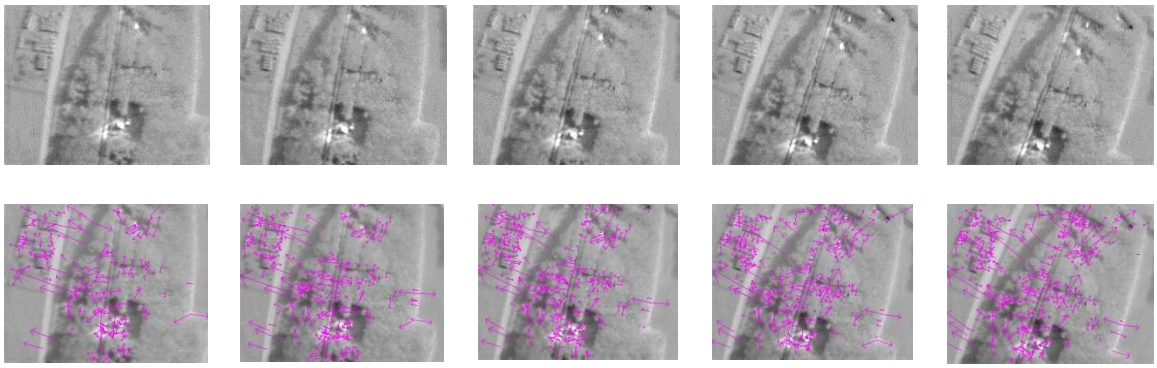


Figure 28. Sequence of five consecutive IR (infrared frames) taken in 2007 by the UASE Laboratory team at the University of North Dakota. Each frame contains 240x320 pixels. The first row of images shows the five frames, and the second row shows the corresponding SIFT features for every frame.

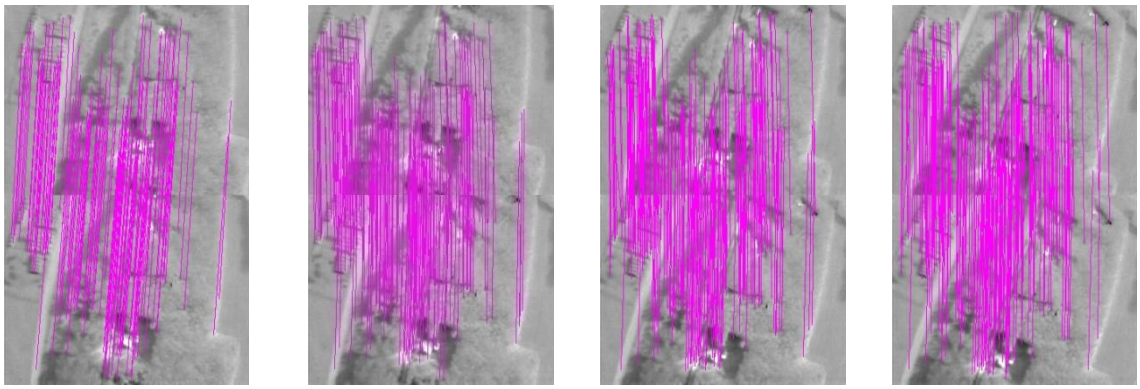


Figure 29. Results of the matching step. From left-to-right: (a) 153 matches between frame 1 and frame 2, (b) 187 matches between frame 2 and frame 3, (c) 206 matches between frame 3 and frame 4, and (d) 229 matches between frame 4 and frame 5.

Figure 29 shows the feature matches for the frames in Figure 28. There are 153 matches between frame 1 and frame 2, 187 matches between frame 2 and frame 3, 206 matches between frame 3 and frame 4, and 229 matches between frame 4 and frame 5. Table 1 shows the algorithm for the estimation of the homography between two images using RANSAC consensus and SIFT features.

4.2.4 Reprojection

Each time that a new frame is gathered by the surveillance camera of the UAS, the homography that relates it with the previous one is computed. The position of the frame in mosaic coordinates is computed by multiplying the current homography with all the previous homographies until the reference frame is reached. Equation (4.6) shows how to compute the homography between frame *j* and frame *i*:

$$\hat{\mathbf{H}}_{ji} = \prod_{k=j+1}^{i} \mathbf{H}_{(k-1)k} \tag{4.6}$$

We use this homography to start the construction of the mosaic. The mosaic is expressed in rectangular coordinates (x and y), because the use of cylindrical coordinates suffers from singularities at the poles [71].

Figure 30 shows the reprojection of the frames shown in Figure 28. The reprojection was shifted ten pixels in the x -direction and 63 pixels in the y -direction to have a complete view of the mosaic construction. The homographies for this reprojection were computed based on (4.6). Figure 31 shows the creation of the mosaic based on the reprojection of the frames in Figure 28. The last column of this figure shows the final mosaic.

Table 1. Estimation of the homography between two images using RANSAC and SIFT.

Objective: Compute the 2D homography between two video frames.

Algorithm:

1. **Features:** Compute interest point features in each image to subpixel accuracy using SIFT features.
2. **Putative correspondences:** Compute a set of interest point matches based on k nearest neighbors in a *kd*-tree using Best Bin First Search.
3. **RANSAC robust estimation:** Repeat for *N* samples:
    a) Select a random sample of four correspondences and compute the homography *H*.
    b) Compute the geometric image distance error for each putative correspondence.
    c) Compute the number of inliers consistent with *H* by the number of correspondences for which the distance error is less than a threshold. Choose the *H* with the largest number of inliers.
4. **Optimal estimation:** Re-estimate *H* from all correspondences classified as inliers, by minimizing the ML cost function using least squares.
5. **Guided matching:** Further interest point correspondences are now determined using the estimated *H* to define a search region about the transferred point position.

The last two steps can be iterated until the number of correspondences is stable.

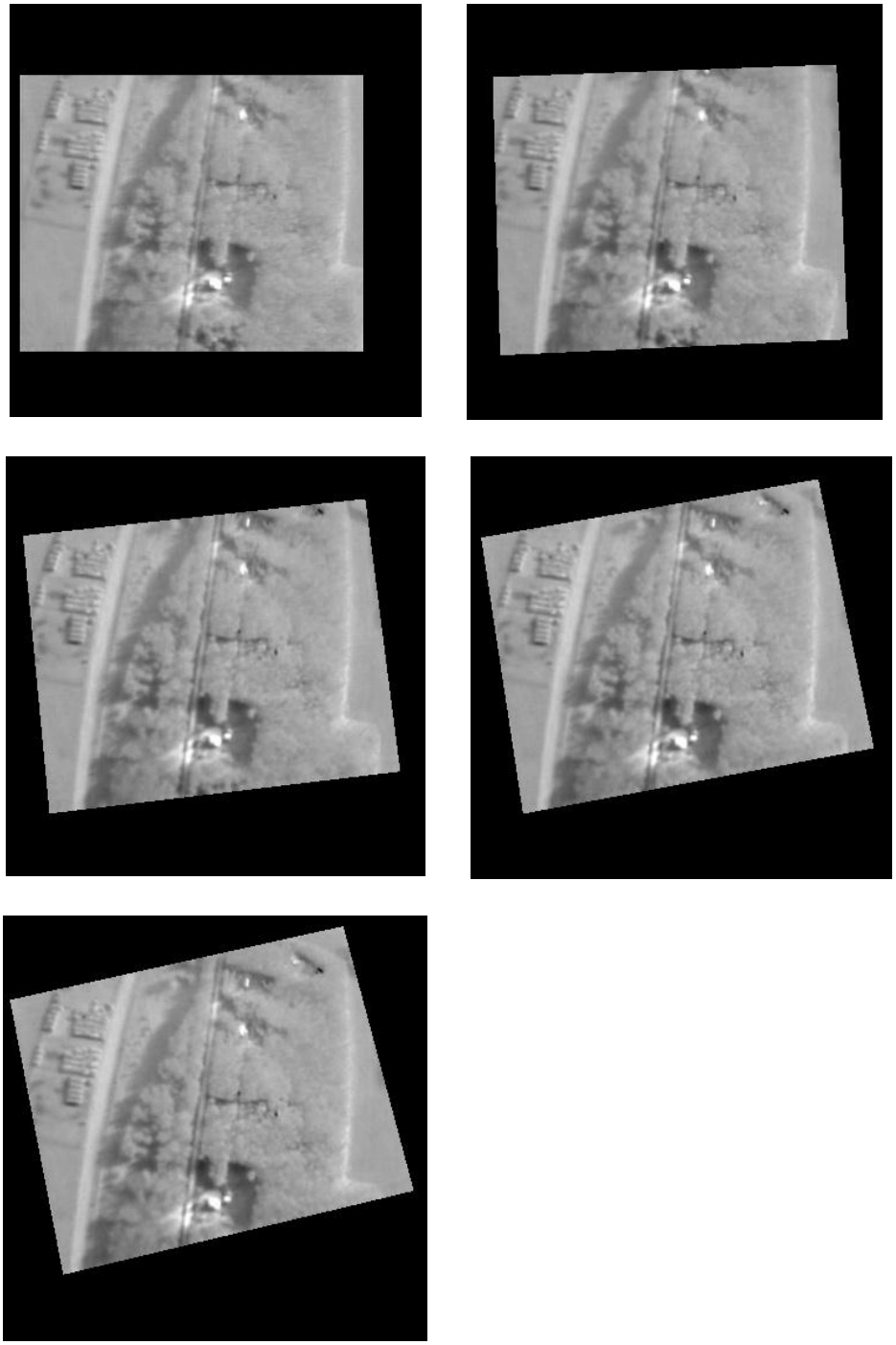


Figure 30. Results of the reprojection step, choosing frame 1 as the reference frame. The size of the mosaic is 360x360 pixels, and all the frames were offset 10 pixels in the x-direction and 63 pixels y-direction The first row shows the reprojection of frame 1 and frame 2, the second row shows the reprojection of the frames 3 and frame 4, and finally the third row shows the reprojection of frame 5.

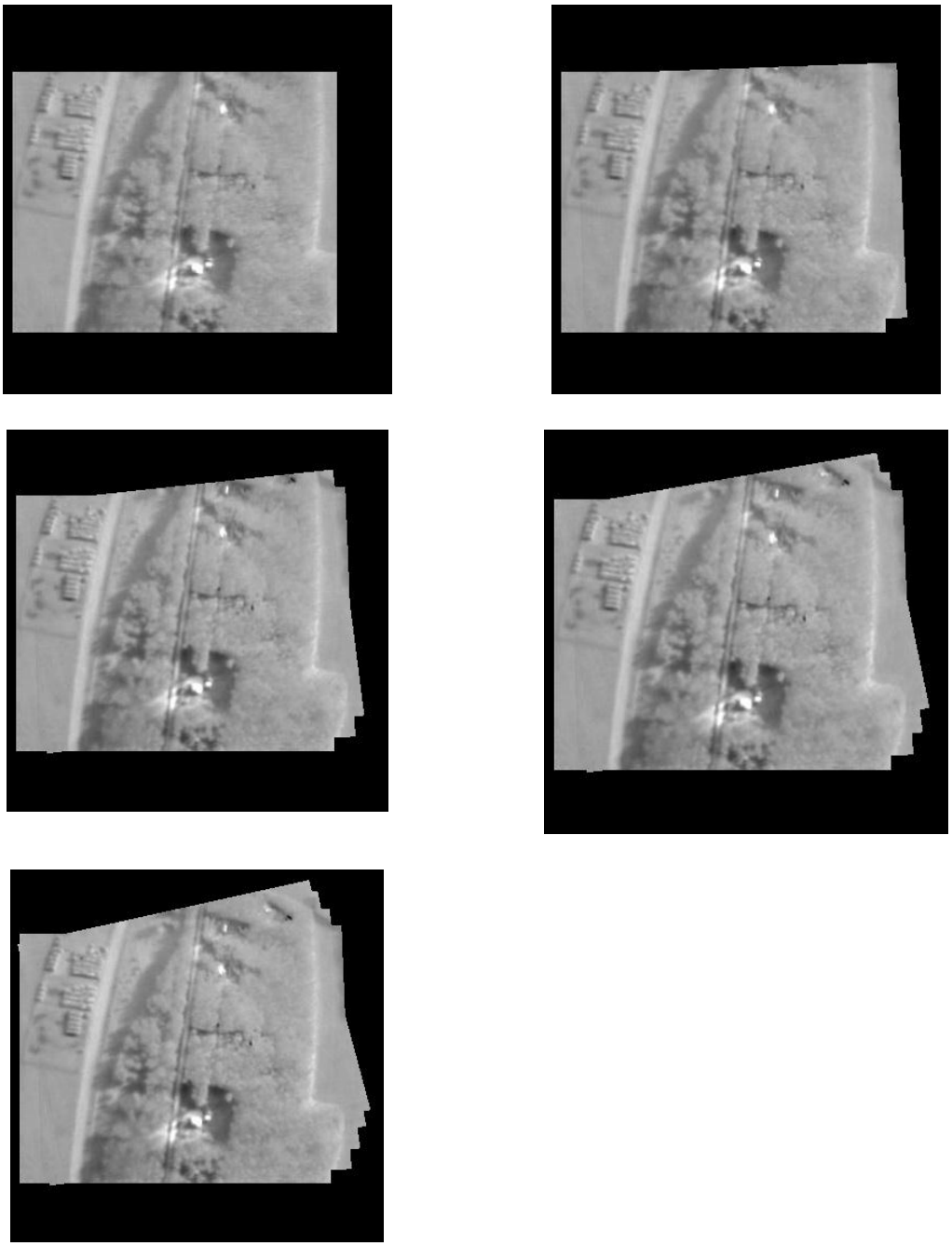


Figure 31. Reprojection in a common rectangular coordinate systems (CRCS). The first row shows the reprojection in the CRCS of frame 1 and frame 2, the second row shows the reprojection in the CRCS for frame 3 and frame 4, and the third row shows the reprojection in the CRCS for frame 5.

4.2.5 Blending

In the ideal case, every frame will have the same intensity level for the same image position in the mosaic. However, because of parallax effects, vignetting, mis-registration errors, radial distortion, and so on, there is always difference in intensity between the overlapping frames that form the mosaic. Therefore, there are image edges which are still visible.

In order to solve this problem, we use multi-band blending [75]. This algorithm needs the construction of masks where the frame is going to be blended. For the construction of the mask, it is necessary to first construct the weight function $W(x, y) = w(x)w(y)$, where $w(x)$ and $w(y)$ vary linearly from 1 at the center of the image to 0 at the edge of the image [76], $w(x)$ is for the width, and $w(y)$ is for the height of the image. The weight functions have to be warped into the mosaic coordinates $W_H^i(x, y)$.

The mask that is used in the blending is found using the weight functions for every frame. The values of the mask will be defined by which corresponding weight function has the largest value in the mosaic coordinates:

$$W_{\max}^i(x, y) = \begin{cases} 1, & \text{if } W_H^i(x, y) = \arg\max_j W_H^j(x, y), \\ 0, & \text{otherwise}. \end{cases} \quad (4.7)$$

Figure 32 shows the weight functions for two different frames (frame 1 and frame 2 from Figure 28) and their respective masks. Figures 32-b and 32-e are the weight functions for frame 1 and frame 2, respectively. Figures 32-c and 32-f are the respective masks for frame 1 and frame 2. Note that the weight function decreases linearly in both axes (x and y); also, the final mask has only two values (0 or 1).

The masks shown in Figures 32 (c) and 32 (f) are used in the multi-band blending algorithm [75]. Basically, it is necessary to construct two pyramids: Gaussian and Laplacian. The Gaussian pyramid is constructed with a sequence of lowpass filtered images. The lowest row, G_0, is the original image. The value of each node in the next level, G_l, is computed as a weighted average of a 5x5 array of G_0 nodes, the same for G_2, and so on. The sample distance is doubled with each iteration, so that successive arrays are half as large in dimension as their predecessors.

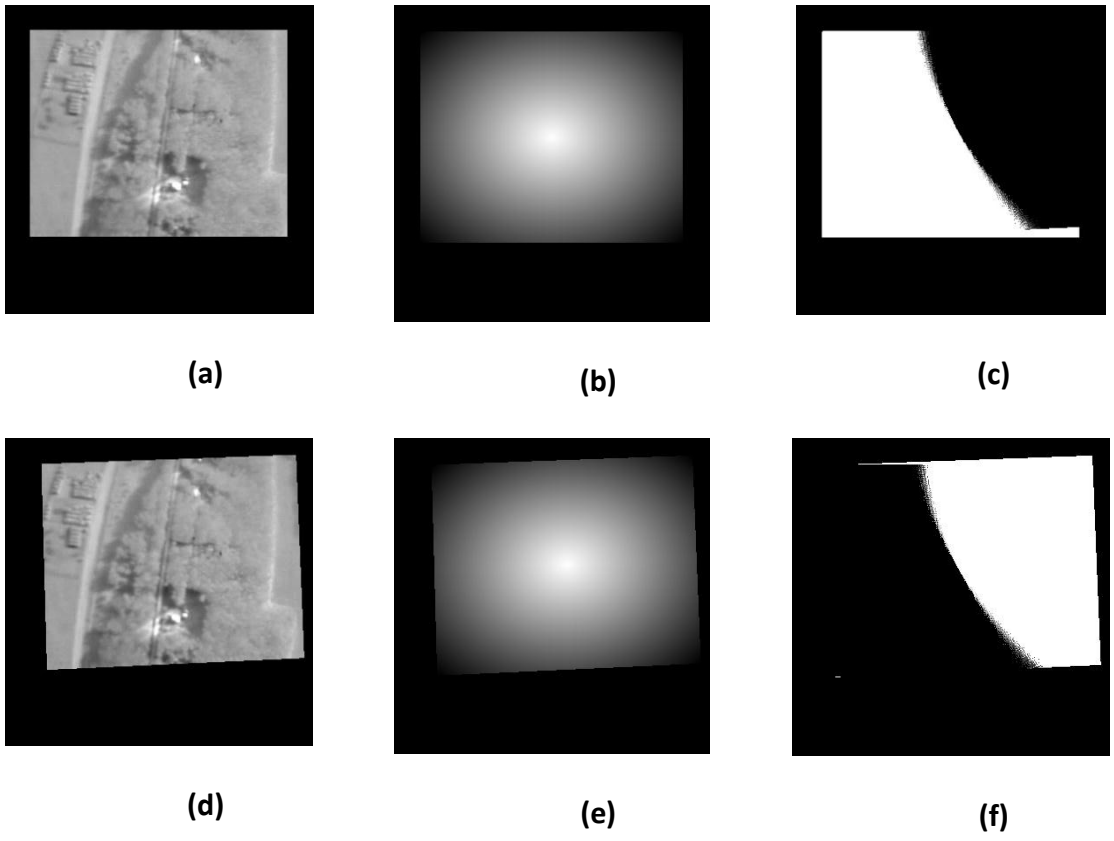


Figure 32. Creating the mask for multi-band blending. (a) and (b) show frame 1 and frame 2, respectively. (b) and (e) are the corresponding weight functions, and (c) and (f) are the masks that will be used in the multi-band blending.

The Gaussain pyramid is computed as

$$G_l(i,j) = \sum_{m,n=1}^{5} w(m,n) G_{l-1}(2i+m, 2j+n), \tag{4.8}$$

where $w(m,n)$ is a pattern of weights used to generate each pyramid level from its predecessors.

The Laplacian pyramid is constructed by subtracting each level of the Gaussian pyramid from the next lowest level. Since these arrays differ in sample density, it is necessary to interpolate new samples between those of a give array before it is subtracted from the next lowest arrays. The Laplacian pyramid is computed as

$$G_{l,k}(i,j) = 4\sum \sum_{m,n=1}^{2} G_{l,k-1}(\frac{2i+m}{2}, \frac{2j+n}{2}) \tag{4.9}$$

$$L_l = G_l - G_{l+1,1} \tag{4.10}$$

For the N^{th} level, $L_N = G_N$ since there is no higher level array to subtract from G_N Table 2 shows the algorithm for multiband-blending. Finally, Table 3 shows the complete algorithm to construct a mosaic for N input images.

Table 2. Algorithm to compute the multi-band blending of two images given a region *R*.

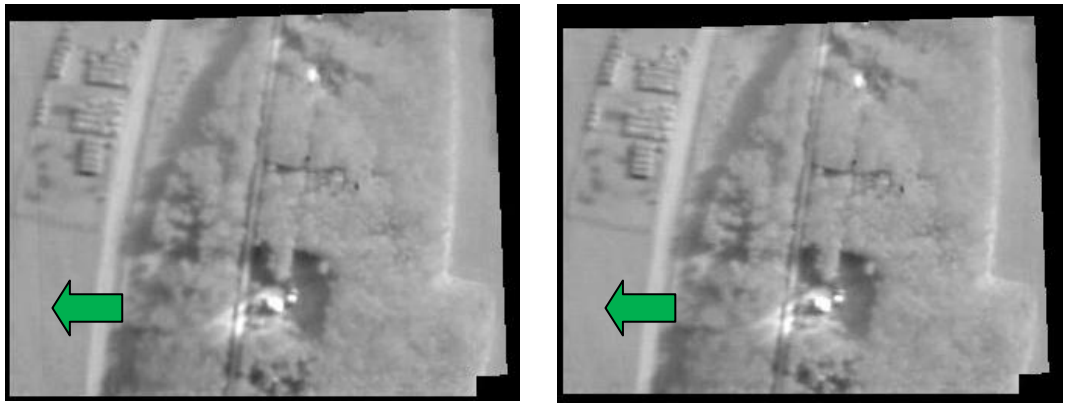
Objective: Blend two images using multi-band blending.

Algorithm:

1. Build the Gaussian pyramids *GA* and *GB* for images *A* and *B*.
2. Build the Laplacian pyramids *LA* and *LB* for images *A* and *B*.
3. Build the Gaussian pyramid *GR* for region *R*.
4. Form the blending pyramid *LS* from *LA* and *LB* using nodes of *GR* as weights:

$$LS_l(i, j) = GR_l(i, j)LA_l(i, j) + (1 - GR_l(i, j))LB_l(i, j)$$

5. Obtain the blended image *S* by expanding and summing the levels of the pyramid *LS*.



(a) (b)

Figure 33. Results of the multi-band blending algorithm. (a) Mosaic without blending and (b) mosaic blended using the multi-band blending algorithm.

Table 3. Algorithm to construct a mosaic given *N* input images.

Objective: Construct a mosaic image from *N* input images.

Algorithm:

1. Compute the SIFT features for all *N* input images.
2. Compute the homography H_{ij} between consecutive images; i.e., between images 1 and 2.
3. Choose an image as a reference image, normally the first image.
4. Reproject all the remaining images to the common coordinate system.
5. Blend the resulting mosaic to eliminate the parallax effect.

4.3 Video Mosaicking

This section expands the idea of image mosaicking to video mosaicking. The creation of a video mosaic follows the same steps, but there are some additional important considerations. One of these considerations involves MPEG video, commonly used in UAS surveillance systems. In the MPEG format, there are three different kinds of frames: I frames, B frames, and P frames [77]. The I-frame, also called the Intra picture is coded separately by itself. The B frame, also called the Bidirectionally Predictive picture, is coded with respect to the immediate next I- or P- frame. The P frame also called Predictive picture, is coded with respect to the immediately previous I- or P-frame.

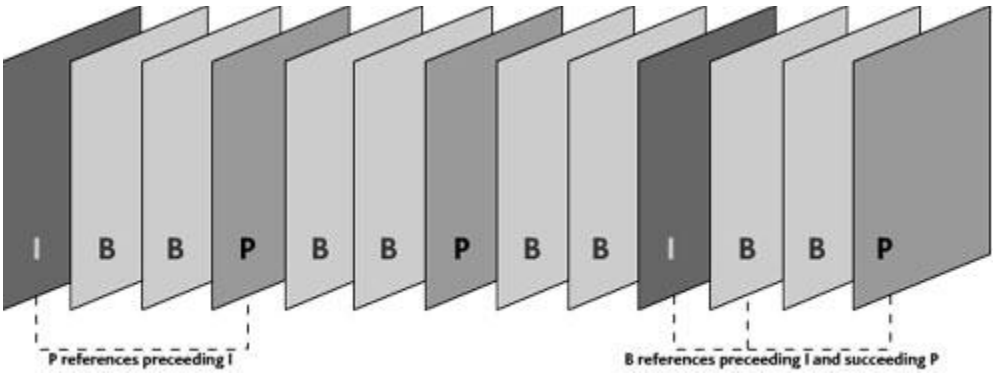


Figure 34. Example of MPEG Group of Pictures (GOP), I-frame, B-frame and P-frame.

MPEG encodes the video in a stream, The basic unit of the stream is a group of pictures (GOP), made up of I-frames, B-frames, and P-frames. The I-frames are coded using essentially JPEG compression, meaning that the information storage is complete enough to decode the frame without reference to any adjacent frames. B-frames and P-frames must be reconstructed by referring to the I-frame. Normally, there is only one I-frame per GOP (see Figure 34). Figures 35 and 36 show two different examples of video

mosaicking. Figure 35 shows the video mosaic constructed from an IR video captured in 2007 by the UASE Laboratory team at the University of North Dakota. Every frame has 240x320 pixels. Figure 36 shows a video mosaic constructed from a demonstration video of MATLAB/Simulink. The implementation of the algorithm shown in Table 4 uses OpenCV for most of the video and image processing, and FFMPEG to select the I-frame of the input video.

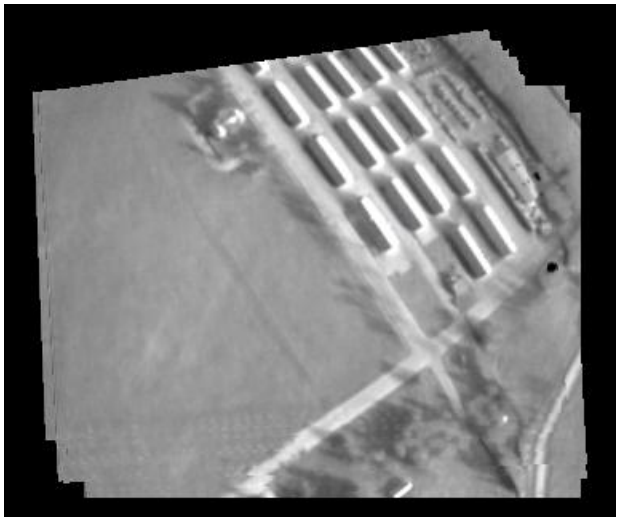


Figure 35. Infrared (IR) video mosaicking for 5.120 seconds at 25 frames per second captured in 2007 by the UASE Laboratory team at the University of North Dakota.

Figure 36. Video mosaicking for 4 seconds at 15 frames per second taken from the video mosaicking demonstration in MATLAB/Simulink.

Table 4. Algorithm to construct a video mosaic given an input MPEG video sequence.

Objective: Construct a video mosaic from an MPEG video sequence.

Algorithm:

1. Read the information of the video file (duration, frames per second, size of the frames, codec type, etc.)
2. Read the video frame-by-frame and do the following:
    a) Select the I-frames.
    b) Select the first frame as a reference frame.
    c) Compute the SIFT features for every frame and save them into memory.
    d) Compute the homography for two consecutive frames.
    e) Reproject the frame into a common coordinate system.

4.4 Geo-referenced Mosaic

In this section, the construction of a geo-referenced mosaic is explained. The first part covers the mathematical transformations to consider in the algorithms described in the next section. Section 4.4.2 explains the use of the Unscented Kalman Filter to geo-reference a mosaic in world coordinates.

4.4.1 The Geometry of Geo-location

The coordinate frames associated with the geo-referenced mosaic of UAS surveillance video include the inertial frame, the vehicle frame, the body frame, the gimbal frame, and the camera frame. The inertial frame, denoted by (X_I, Y_I, Z_I), which is a fixed frame with X_I directed to North, Y_I directed to East and, Z_I directed towards the center of the Earth. The vehicle frame is denoted by (X_v, Y_v, Z_v), is oriented identically to the initial frame, but its origin is at the vehicle center of mass. The body frame is denoted by (X_b, Y_b, Z_b), which is also centered at the center of mass of the UAS. X_b points through the nose of the UAS, Y_b points through the right wing, and Z_b points out the belly (see Figures 37 and 38). The gimbal frame is represented by (X_g, Y_g, Z_g), and its originates at the gimbal rotation center and is oriented so that X_g points along the optical axis, Y_g points in the image plane and Z_g points down in the image plane. The camera is denoted by (X_c, Y_c, Z_c), and it originates at the optical center of the camera, with X_c pointing up in the image, Y_c pointing right in the image plane, and Z_c directed along the optical axis.

In order to use these different frames, it is necessary to construct transformation matrices. The homogeneous transformation matrix from frame *i* to frame *j* is given by

$$T_i^{\ j} = \begin{pmatrix} R_i^j & -d_i^j \\ 0 & 1 \end{pmatrix}, \tag{4.11}$$

where $0 \in \Re^3$ is a row vector, and d_i^j is the resolved j^{th} coordinate frame. The inverse transformation is given by

$$T_j^i \equiv T_i^{j-1} = \begin{pmatrix} R_i^{jT} & R_i^{jT} d_i^j \\ 0 & 1 \end{pmatrix} \tag{4.12}$$

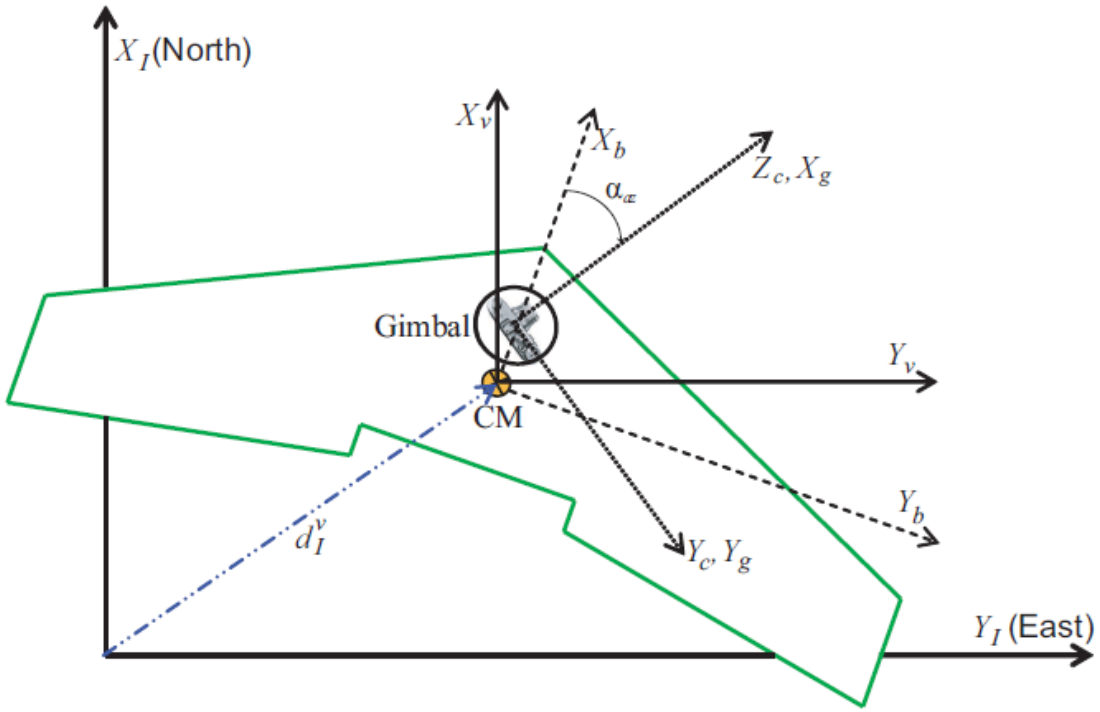


Figure 37. Top view of the coordinate frames. The inertial and vehicle (UAS) frames are aligned with the world, the body is aligned with the airframe, and the gimbal and camera frames are aligned with the camera.

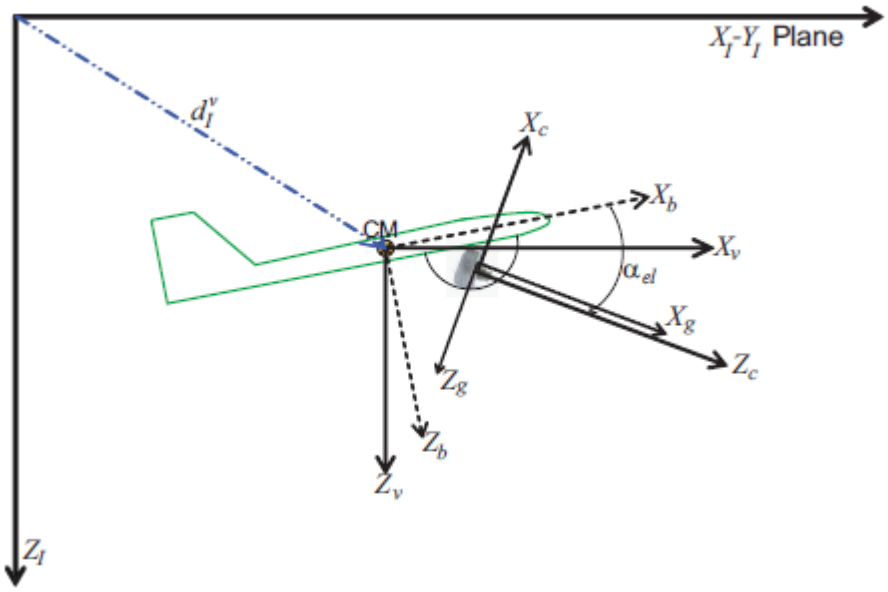


Figure 38. Lateral view of the coordinate frames.

Table 5. Homogeneous transformation matrices between frames.

Transformation	Description
T_I^v	Inertial to UAS frame vehicle
T_v^b	UAS vehicle to UAS body frame
T_b^g	UAS body frame to Gimbal frame
T_g^c	Gimbal to Camera frame

Transformation from the Inertial to the Vehicle (UAS) Frame

The transformation from the inertial to the vehicle frame is given by

$$T_I^v = \begin{pmatrix} I & -d_I^v \\ 0 & 1 \end{pmatrix}, \tag{4.13}$$

where

$$d_I^v = \begin{pmatrix} x_{UAS} \\ y_{UAS} \\ -h_{UAS} \end{pmatrix}. \tag{4.14}$$

In this expression, x_{UAS}. and y_{UAS}. represent the North and East locations of the UAS as measured by the GPS sensor, and h_{UAS} represents the altitude as measured by a relevant sensor, all data are provided by the TASE gimbal [93].

Transformation from the vehicle to the body frame

The transformation from the vehicle to the UAS body frame, T_v^b , consists of a rotation based on measurements of the Euler angles by the INS,

$$T_v^b = \begin{pmatrix} R_v^b & 0 \\ 0 & 1 \end{pmatrix} \tag{4.15}$$

where

$$R_v^b = \begin{bmatrix} c_\theta c_\psi & c_\theta s_\psi & -s_\theta \\ s_\phi s_\theta c_\psi - c_\phi s_\psi & s_\phi s_\theta s_\psi + c_\phi c_\psi & s_\phi c_\theta \\ c_\phi s_\theta c_\psi + s_\phi s_\psi & c_\phi s_\theta s_\psi - s_\phi c_\psi & c_\phi c_\theta \end{bmatrix}. \tag{4.16}$$

Here, ϕ , θ , and ψ represent the UAS roll, pitch, and heading angles in radians. Also, $c_\varphi = \cos\varphi$ and $s_\varphi = \sin\varphi$. All these angles are gathered by TASE gimbal.

Transformation from UAS body frame to the gimbal frame

The transformation from the body of the UAS to the gimbal frame, T_b^g depends on the location of the UAS center of mass with respect to the gimbal's rotation center, denoted by the vector d_b^g . T_b^g also depends on the rotation that aligns the gimbal's coordinate frame with the UAS's body frame. This rotation is denoted as R_b^g and requires measurements of the camera's azimuth and elevation angles,

$$T_b^g = \begin{pmatrix} R_b^g & -d_b^g \\ 0 & 1 \end{pmatrix} \tag{4.17}$$

where

$$R_b^g = R_{y,\alpha_{el}} R_{z,\alpha_{az}}$$

$$= \begin{bmatrix} c_{el}c_{az} & c_{el}s_{az} & s_{el} \\ -s_{el} & c_{az} & 0 \\ -s_{el}c_{az} & -s_{el}s_{az} & c_{el} \end{bmatrix} \tag{4.18}$$

Here, α_{az} denotes the azimuth angle of rotation about Z_g and α_{el} the elevation angle of rotation about Y_g , after α_{az} . This information is also provided by the TASE gimbal.

Transformation from the gimbal to the camera frame

The transformation from the gimbal to the camera reference, T_g^c , depends on the vector d_g^c , which describes the location of the gimbal's rotation center relative to the camera's optical center, and is resolved in the camera's coordinate frame. T_g^c depends also on a fixed rotation R_g^c, which aligns the camera's coordinates to the gimbal coordinates. This transformation is given by

$$T_g^c = \begin{pmatrix} R_g^c & -d_g^c \\ 0 & 1 \end{pmatrix}, \tag{4.19}$$

where

$$R_g^c = \begin{bmatrix} 0 & 0 & -1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix}. \tag{4.20}$$

4.4.2 Data fusion between GPS/IMU and Video

The data fusion of the GPS/IMU and video data from the UAS improves the accuracy of its pose estimation. There are many ways to do this, most commonly via the Kalman Filter [78]. This section presents an Unscented Kalman Filter (UKF) based on [79, 80, 81]. The advantage to using UKF is because of its ability to combine data with varying uncertainty in highly non-linear systems. The state of the UKF is a 12x1 vector representing the two most recent poses of the UAS. The state is initialized as

$$x_0 = [y_0, y_1]^T, \tag{4.21}$$

where y_t represents the pose of the UAS at time *t*. This pose includes three parameters for the location of the UAS, $(\hat{x}, \hat{y}, \hat{z})$, and three parameters for the attitude, (ϕ, θ, ψ). The reason behind this is because it is necessary to keep two pose estimates to extract information about camera motion.

Whenever a new pose estimate is received from the GPS/IMU (TASE gimbal telemetry information), the state of the UKF is updated to contain the most recent pose measurement, and the previous refined estimate,

$$x_t = [y_t^{'}, y_{t+1}]^T, \tag{4.22}$$

By keeping a refined pose estimate in the state of the UKF, the pose estimates becomes more accurate over time. Corresponding steps are taken with the covariance matrix associated with the state of the UKF.

In order to generate refined pose estimates, it is necessary to (1) define the state-measurement function, and (2) implement the measurement update step of the UKF.

State-measurement function

The displacement of the camera from one point to another is related to the homography, as was shown in Section 4.1.1.2. Furthermore, the homography can be expressed as

$$H = K_2(R + \frac{1}{d}TN^T)K_1^{-1}, \tag{4.23}$$

where R is the rotation of camera 1 to camera 2, and T is the translation (see Figure 4.16) from camera 1 to camera 2 [79, 80]. Because the gimbal has only one camera, $K_1 = K_2 = K$, and thus

$$H = K(R + \frac{1}{d}TN^T)K^{-1}. \tag{4.24}$$

For the state vector $x_t = \left[y_t^{'}, y_{t+1}\right]^T$ in the UKF, two rotation matrices $\left(R_{wc_t}, R_{wc_{t+1}}\right)$ can be obtained from the world frame (inertial frame as in Section 4.3.1) to the coordinate frame of the camera at time t and t+1. In the same way, two translation vectors for the cameras in world coordinates (inertial frame) can be derived.

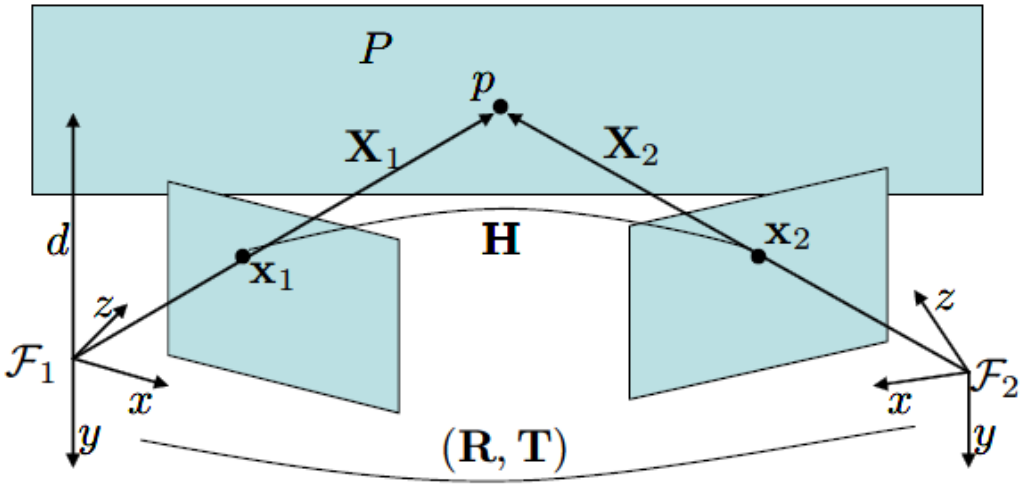


Figure 39. Rotation *R* and translation T from the camera. This rotation and translation are related by **H**. $\mathbf{x}_1$, and $\mathbf{x}_2$, and they are both representations of the real world point *p* in camera coordinates.

The rotation *R* and translation *T* can be computed using the rotations of point 1 and point 2 as:

$$R = R_{wc2} R_{wc1}^{\ T} \tag{4.25}$$

$$T = R_{wc2}(T_{wc1-} T_{wc2}) \tag{4.26}$$

To obtain *N*, the normal of the plane in world (inertial frame) coordinates, N_w must be rotated into camera 1 coordinates,

$$N = R_{wc1} N_w \ . \tag{4.27}$$

Since it is assumed that the world is perfectly flat, N_w = [0,0,1], and

$$H = K R_{wc2} \left(I - \frac{1}{d}(T_{wc1}^{w} - T_{wc2}^{w}) N_w^T \right) R_{wc1}^T K^{-1}. \tag{4.28}$$

Then, it is necessary to normalize (4.28) as

$$H = \frac{H}{h_{3,3}}. \tag{4.29}$$

Measurement Update

The measurement update of the UKF uses the current state, the current homography measurement, and their associated covariance matrices to estimate a more accurate current state. To do this, it is necessary to transform the state-covariance matrix into the measurement space. UKF achieves this transformation by sampling the distributions of the state variables to come up with a number of sample states. Table 6 shows the algorithm using UKF to compute a more accurate pose estimation using GPS, IMU, and video frames from a UAS. Equation (4.31) requires the computation of P_h, which is the covariance of the homography. In order to do this, it is necessary to first compute first the homography following the steps of the algorithm shown in Table 1.

Table 6. Algorithm for the pose estimation based on fusion of the GPS/IMU information and video frames using UKF.

Objective: Update the measurement by using UKF to estimate the pose of the UAS.

Algorithm:

1. Create the augmented state vector:

$$\mathbf{x}^a = \left[y'_{t-1}, y_t, v\right]^T \tag{4.30}$$

2. Create an augmented covariance matrix from the covariance

$$P^a = \begin{bmatrix} P_{t-1} & 0 & 0 \\ 0 & P_t & 0 \\ 0 & 0 & P_h \end{bmatrix} \tag{4.31}$$

Continued…

3. Construct the set of sample points χ_i:

$$\chi_0 = x^a$$

$$\chi_i = x^a + \left(\sqrt{(L+k)P^a}\right)_i \tag{4.32}$$

$$\chi_i = x^a - \left(\sqrt{(L+k)P^a}\right)_i$$

Here, $\sqrt{(L+k)P^a}$ is the i^{th} column of the matrix square root of $(L+k)P^a$. The value of k is set such that $L+k = 3$ [78].

4. Transform the sample points with the state-measurement transformation:

$$\gamma_i = \varphi(\chi_i) \tag{4.33}$$

5. Find the mean of the sample homographies:

$$\hat{\mathbf{h}} = \sum_{i=0}^{2L} W_i \gamma_i. \tag{4.34}$$

6. Compute the covariance from the weighted outer product of the transformed points

$$P_{hh} = \sum_{i=0}^{2L} W_i \left(\gamma_i - \hat{\mathbf{h}}\right)\left(\gamma_i - \hat{\mathbf{h}}\right)^T \tag{4.35}$$

7. Compute the state-measurement cross-correlation from the weighted outer product of both sets of points:

$$P_{xh} = \sum_{i=0}^{2L} W_i \left(\chi_i - \mathbf{x}\right)\left(\gamma_i - \hat{\mathbf{h}}\right)^T \tag{4.36}$$

8. Compute the Kalman gain

$$K = P_{xh} P_{hh}^{-1} \tag{4.37}$$

Continued …

9. Update the pose estimates

$$\hat{\mathbf{x}} = \mathbf{x} + K(\mathbf{h} - \hat{\mathbf{h}}) \tag{4.38}$$

10. Update the state covariance

$$P_{xx} = P_{xx} - KP_{hh}K^T \tag{4.39}$$

From the estimation of the homography, we obtain a set of inlier SIFT features. Using these inliers. we can compute the standard deviation of this residual error, σ, by

$$\sigma = \sqrt{\frac{\sum_{i=1}^{n} \left\| \Gamma_i^{'} - \mathbf{H}\Gamma_i \right\|^2}{n}} \ , \tag{4.40}$$

where *n* is the number of inlier SIFT features. Using σ, we can compute the covariance matrix, P_{Γ}, representing the uncertainty in all features in both x and y locations in the image:

$$P_{\Gamma} = \frac{\sigma^2}{2}\mathbf{I} \tag{4.41}$$

Due the non-linearity of the transformation from point correspondences to a homography matrix, it is necessary to use UKF. In order to do that, a set of sample point correspondences is created, transforming them into homography matrices and computing the covariance of the resulting matrices.

A set of $4n+1$ sample points χ_i are created as follows:

$$\begin{aligned} \chi_0 &= \Gamma' \\ \chi_i &= \Gamma' + \left(\sqrt{(L+k)P}\right) \\ \chi_i &= \Gamma' - \left(\sqrt{(L+k)P}\right) \end{aligned} \tag{4.42}$$

Here, $L = 2n$, $k = 3 - L$, and $\Gamma' = \left[\Gamma_1' ... \Gamma_n'\right]$.

A homography is computed for each sample χ_i by computing the sample point with the feature points $\Gamma' = \left[\Gamma_1' ... \Gamma_n'\right]$ using the algorithm shown in Table 1, creating $4n+1$ vectorized homography matrices, γ_i. Each homography is then normalized using equation (4.29). The covariance of the homographies is computed as

$$P_h = \frac{1}{2n}\sum_{i=1}^{2n}(\gamma_i - \gamma_0)(\gamma_i - \gamma_0)^T, \tag{4.43}$$

Figure 40 shows an example of geo-referenced mosaic taken from [80], and Figure 41 shows geo-referenced images taken from a color (red, green) and near infrared (false color composite) camera, along with GPS and IMU data.

Figure 40. Example of geo-referenced mosaic taken from [80] .

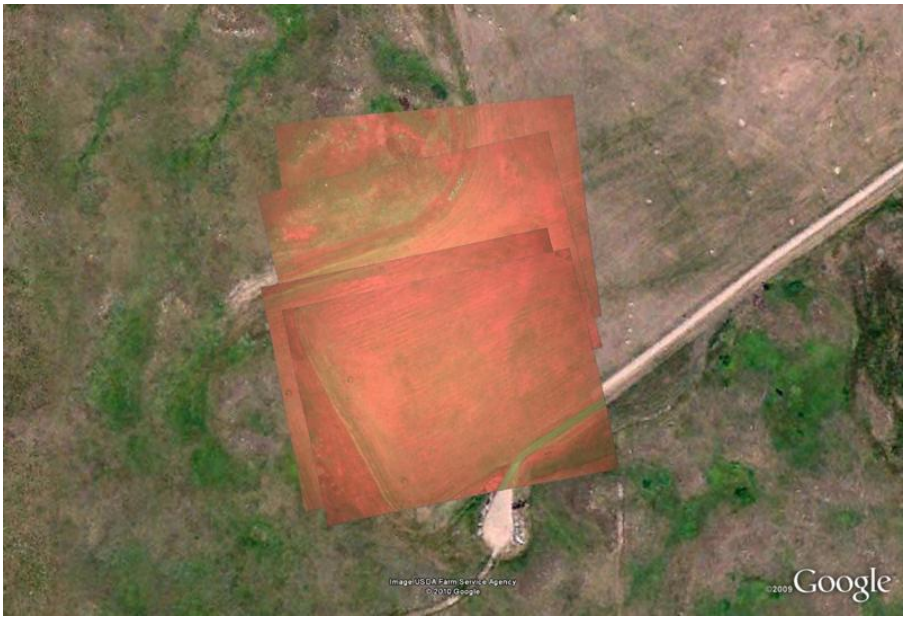


Figure 41. Example of geo-referenced images using images from a color (red, green) and near infrared (false color composite) camera along with GPS and IMU information. (Photo: Courtesy of David Dvorak) .

CHAPTER 5

SUPER-RESOLUTION MOSAICKING USING STEEPEST DESCENT, CONJUGATE GRADIENT, AND LEVENBERG MARQUARDT OPTIMIZATION

5.1 Introduction

In this chapter, different optimization algorithms are used to compute the super-resolution mosaic. The optimization algorithms used are: 1) steepest descent (SD) 2) conjugate gradient (CG), and 3) Levenberg Marquardt (LM). The use of the Levenberg Marquardt algorithm is a novel optimization method for super-resolution. Furthermore, super-resolution mosaicking can be represented as a large sparse linear system, but we present a different framework for solving this system efficiently using spatial-domain operations.

Super-resolution mosaicking combines both methods, and it has a number of applications when UAS or satellite surveillance video is enhanced. One clear application is the surveillance of certain areas even during night with the use of an uncooled infrared (IR) imaging system. The UAS can fly over areas of interest and generate super-resolved mosaics that can be analyzed at the ground control station. Other important applications involve the supervision of high voltage transmission lines, oil pipes, and the highway system. NASA also uses super-resolution mosaics to study the surface of Mars, the moon, and other planets.

Super-resolution mosaicking has been studied by many researchers. Zomet and Peleg [85] used the overlapping area within a sequence of video frames to create a super-resolved mosaic. In this method, the SR reconstruction technique proposed in [24] is

applied to a strip rather than a entire image. This means that the resolution of each strip is enhanced by the use of all the frames that contain that particular strip. The disadvantage is that this method is computationally expensive. Ready and Taylor [91] use a Kalman filter to compute the super-resolved mosaic. They add unobserved data to the mosaic using Dellaert's method [86, 91]. Basically, they construct a matrix that relates the observed pixels to estimated mixel values. This matrix is constructed using the homography matrix and the point spread function (PSF). The problem is that this matrix is extremely large, so they use a Kalman filter and diagonalization of the covariance matrix to reduce the amount of storage and computation required. The drawback of this algorithm is the use of the large matrix, and the best results with synthetic data obtain a peak signal-to-noise ratio (PSNR) of 31.6 dB. Simolic and Wiegand [87] use a method based on image warping. In this method, each pixel of each frame is mapped onto the SR mosaic, and its gray level value is assigned to the corresponding pixel in the SR mosaic within a range of ± 0.2 pixel units. The drawback of this method is that it requires highly accurate motion vectors and homography estimates, which is difficult when dealing with real surveillance video from UAS. Wang, Fevig, and Schultz [94] use the overlapped area within five consecutive frames from a video sequence. Then they use sparse matrices to model the relationship between the LR and SR frames, which is solved using maximum *a posteriori* estimation. To deal with the ill-posed problem of the super-resolution model, they use hybrid regularization. The drawback of this method is that it has to be used every five frames, which means that for every five frames, several sparse matrices have to be constructed. Therefore, this method does not seem to be appropriate to deal with a real video sequence which has thousand of frames. Pickering and Ye [83] proposed an

interesting model for mosaicking and super-resolution of video sequences using the Laplacian operator to find the regularization factor. The problem with the use of the Laplacian factor is that it forces spatial smoothness. Therefore, edge pixels are removed in the regularization process, eliminating sharp edges [83]. Arican and Frossard [95] use the Levenberg Marquardt algorithm to compute the SR of onmidirectional images. Chung [96] proposed different Gauss Newton methods to compute the SR of images, the disadvantage is that this optimization technique works only for small images.

In this chapter, different algorithms to compute super-resolution mosaics are presented, and compared to one another. The advantages of these proposed algorithms are that they do not need the creation of huge sparse matrices, and they are fast, adaptive and robust. Therefore, it is feasible to apply the algorithms to a relatively large image sequences to obtain a video mosaics. Also, we use Huber regularization, which preserves high frequency pixels so sharp edges are preserved.

5.2 Observation Model

Assuming that there are K frames of LR images available, the observation model can be represented as

$$\mathbf{y}_k = \mathbf{D}\mathbf{B}_k\mathbf{W}_k\mathbf{R}[\mathbf{x}]_k + \boldsymbol{\eta}_k = \mathbf{H}_k\mathbf{x} + \boldsymbol{\eta}_k. \tag{5.1}$$

Here , $\mathbf{y}_k$ (k =1,2, ..., K), $\mathbf{x}$, and $\boldsymbol{\eta}_k$ represent the k^{th} LR image, the part of the real world depicted by the super-resolution mosaic, and the additive noise, respectively. The observation model in (5.1) introduces $\mathbf{R}[\mathbf{x}]_k$, which represents the reconstruction of the k^{th} warped SR image from the original high-resolution data $\mathbf{x}$ [83]. The geometric warp operator and the blur matrix between $\mathbf{x}$ and the k^{th} LR image, $\mathbf{y}_k$ are represented by $\mathbf{W}_k$ and $\mathbf{B}_k$, respectively. The decimation operator is denoted by $\mathbf{D}$.

5.3 Robust Super-resolution Mosaicking

The estimation of the unknown SR mosaic image is not only based on the observed LR images, but also on many assumptions such as the blurring process and additive noise. The motion model is computed as a projective model using the homography between frames; the blur is considered only optical. The additive noise, $\boldsymbol{\eta}_k$, is considered to be independent and identically distributed white Gaussian noise. Therefore, the problem of finding the maximum likelihood estimate of the SR mosaic image $\hat{\mathbf{x}}$ can be formulated as

$$\hat{\mathbf{x}} = \arg\min_{\mathbf{x}} \left\{ \left\| \sum_{k=1}^{K} \mathbf{y}_k - \mathbf{D}\mathbf{B}_k\mathbf{W}_k\mathbf{R}[\mathbf{x}]_k \right\|_2^2 \right\}. \tag{5.2}$$

In this case, $\| \ \|_2$ denotes the Euclidean norm. As the SR reconstruction is an ill-posed inverse problem, we need to add another term for regularization, which must contain prior information for the SR mosaicking. This regularization term helps to convert the ill-posed problem into a well-posed problem. We use Huber regularization, because it preserves edges and high frequency information [69,70]:

$$\hat{\mathbf{x}} = \arg\min_{\mathbf{x}} \left\{ \left\| \sum_{k=1}^{K} \mathbf{y}_k - \mathbf{D}\mathbf{B}_k\mathbf{W}_k\mathbf{R}[\mathbf{x}]_k \right\|_2^2 + \lambda \sum_{g \in \mathbf{G}_x} \rho(g, \alpha) \right\} \tag{5.3}$$

The Huber function is defined as

$$\rho(x,\alpha) = \begin{cases} x^2, & \text{if} \quad |x| \le \alpha, \\ 2\alpha|x| - \alpha^2, & \text{otherwise.} \end{cases} \tag{5.4}$$

5.4 Super-resolution Mosaicking Using Steepest Descent

Based on the gradient descent algorithm for minimizing (5.3), the robust iterative update for $\hat{\mathbf{x}}$ can be expressed as

$$\hat{\mathbf{x}}^{(n+1)} = \hat{\mathbf{x}}^{(n)} + \alpha^{(n)} \left\{ \mathbf{R}^T \left[\mathbf{W}_k^T \mathbf{B}_k^T \mathbf{D}^T (\mathbf{y}_k - \mathbf{D}\mathbf{B}_k \mathbf{W}_k \mathbf{R}[\hat{\mathbf{x}}^{(n)}]_k \right]_{k=1}^{K} - \lambda^{(n)} \mathbf{G}^T \rho'(G\hat{\mathbf{x}}^{(n)}, \alpha) \right\} \tag{5.5}$$

where $\mathbf{G}$ is the gradient operator over the cliques [81,7], and $\lambda^{(n)}$, the regularization operator can be computed as

$$\lambda^{(n)} = \left(\frac{\sum_{k=1}^{K} \left\| \mathbf{y}_k - \mathbf{D}\mathbf{B}_k \mathbf{W}_k \mathbf{R}[\hat{\mathbf{x}}^{(n)}]_k \right\|_2}{K \sum_{g \in Gx} \rho(g, \alpha)} \right)^2 \cdot \ldots\ldots \qquad \ldots\ldots\ldots (5.6)$$

Furthermore, the derivative of the Huber function is given as:

$$\rho'(x, \alpha) = \begin{cases} 2x, & \text{if} \quad |x| \le \alpha, \\ 2\alpha sign(x), & \text{otherwise.} \end{cases} \tag{5.7}$$

In this case,

- $\mathbf{R}$ and $\mathbf{R}^T$: $\mathbf{R}$ represents the reconstruction from the mosaic image, and $\mathbf{R}^T$ represents the construction of a mosaic.
- $\mathbf{W}$ and $\mathbf{W}^T$: $\mathbf{W}$ represents the backward warping inverse of the homography, and $\mathbf{W}^T$ is the forward warping using the homography. The homography is computed using the SIFT (Scale Invariant Feature Transform) features and RANSAC (Random Sample Consensus) algorithm detailed in Chapter 4, Table 1.
- $\mathbf{B}$ and $\mathbf{B}^T$: $\mathbf{B}$ represents the blurring effect, and is implemented by a convolution with the PSF kernel. $\mathbf{B}^T$ is implemented by convolution with the flipped PSF kernel.

- $\mathbf{D}$ and $\mathbf{D}^T$: $\mathbf{D}$ represents image interpolation $\mathbf{D}^T$ represents image decimation. Image interpolation refers to the process of upsampling followed by appropriate lowpass filtering, and image decimation refers to downsampling after appropriate anti-alias filtering (see Figure 42).

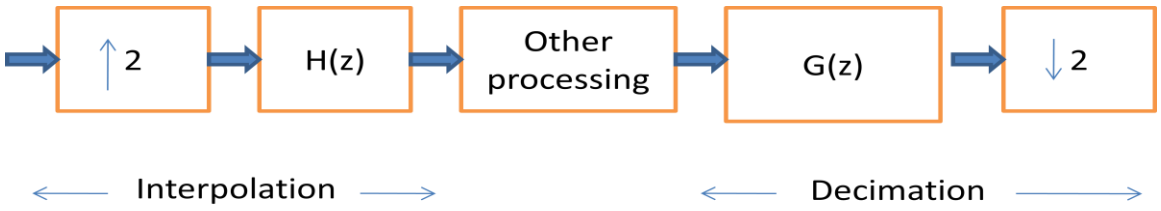


Figure 42. Image Interpolation and decimation.

The gradient operator $\mathbf{G}$ has the advantage over the Total Variation (TV) prior; the function and its gradient with respect to $\hat{\mathbf{x}}^{(n)}$ are continuous as well as convex [90]. Therefore, the optimization problem can be solved easily using gradient-descent methods such as steepest descent and conjugate gradient methods.

The clique structure determines the spatial interactions. The spatial interactions are used with our proposed method, and its activity is computed using finite difference approximations to second-order directional derivatives (vertical, horizontal, and two diagonal directions) in each super-resolution mosaic $\hat{\mathbf{x}}^{(n)}$.

$$\partial_1 \hat{x}^{(n)} = \hat{x}^{(n)}[n_1 - 1, n_2] - 2\hat{x}^{(n)}[n_1, n_2] + \hat{x}^{(n)}[n_1 + 1, n_2] \quad (5.8)$$

$$\partial_2 \hat{x}^{(n)} = \hat{x}^{(n)}[n_1, n_2 - 1] - 2\hat{x}^{(n)}[n_1, n_2] + \hat{x}^{(n)}[n_1, n_2 + 1] \quad (5.9)$$

$$\partial_3 \hat{x}^{(n)} = \frac{1}{2}\hat{x}^{(n)}[n_1 + 1, n_2 - 1] - \hat{x}^{(n)}[n_1, n_2] + \frac{1}{2}\hat{x}^{(n)}[n_1 - 1, n_2 + 1] \quad (5.10)$$

$$\partial_4 \hat{x}^{(n)} = \frac{1}{2}\hat{x}^{(n)}[n_1 - 1, n_2 - 1] - \hat{x}^{(n)}[n_1, n_2] + \frac{1}{2}\hat{x}^{(n)}[n_1 + 1, n_2 + 1] \quad (5.11)$$

Figure 43 graphically shows these finite four difference approximations for the pixel *n* in the super-resolution mosaic $\hat{x}^{(n)}$.

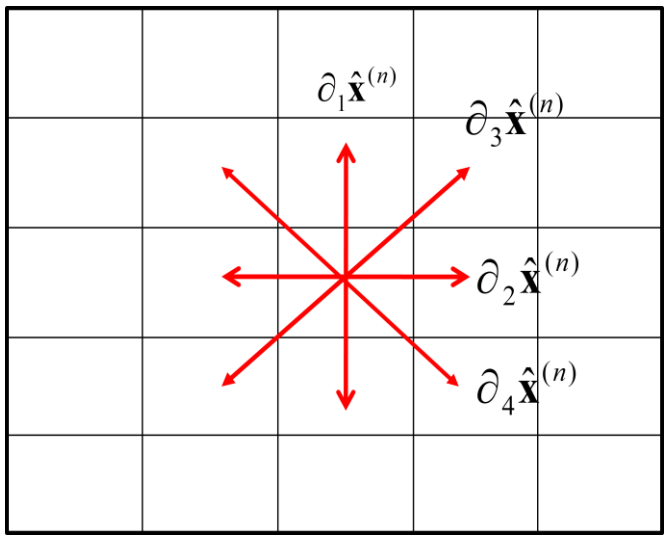


Figure 43. Local spatial interactions representing by four finite difference approximations of the pixel $\hat{x}^{(n)}$.

Table 7 describes the construction of a super-resolved mosaic given a set of *N* input frames or images using steepest descent algorithm.

Table 7. Algorithm to construct a super-resolved mosaic using steepest descent algorithm given a set of input frames.

Objective: Construct a super-resolved mosaic using steepest descent algorithm from a set of *N* input frames.

Algorithm:

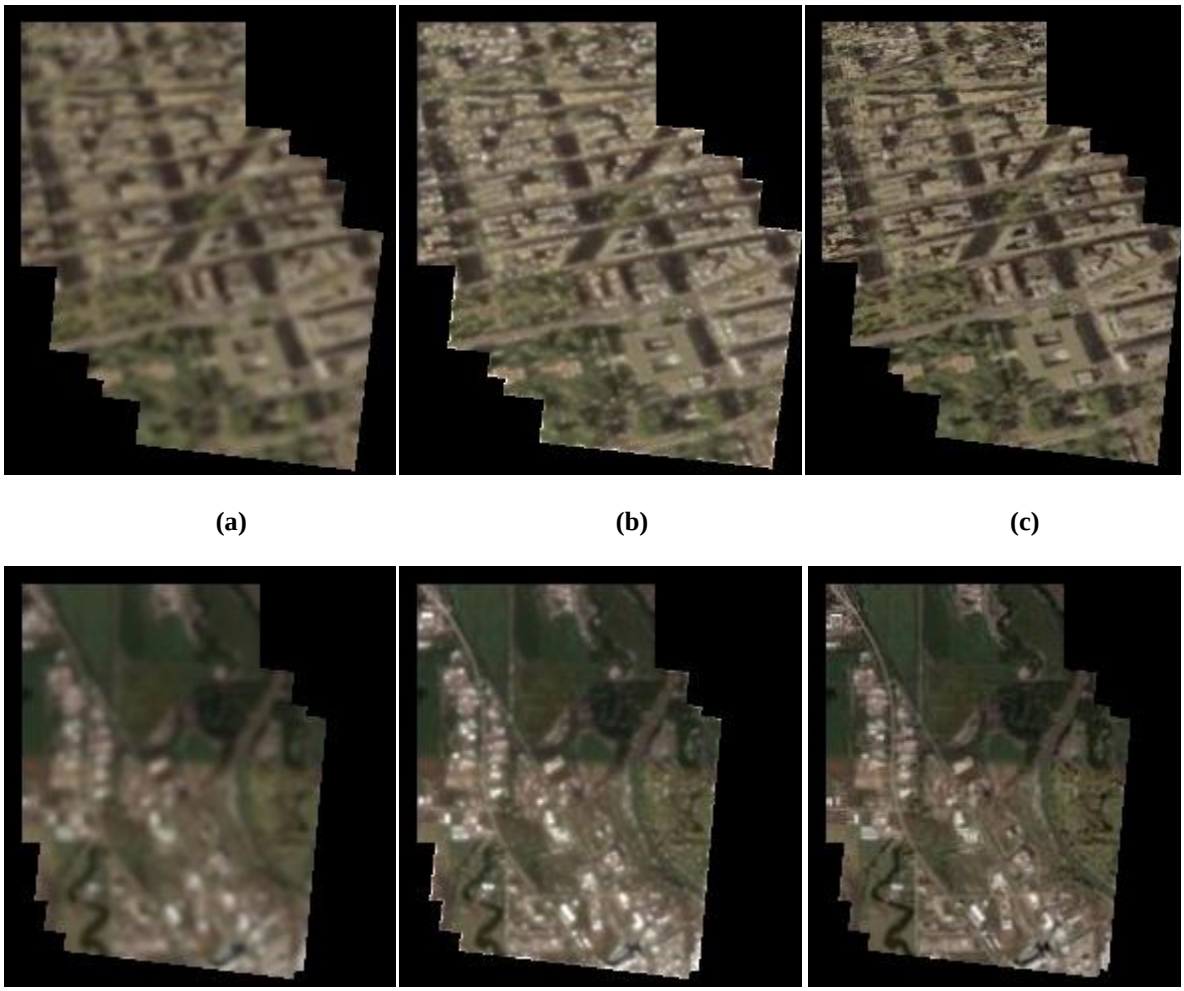
1. Compute the homography using Table 1 between consecutive frames; i.e., $h_{12}, h_{23}, ..., h_{N-1,N}$.
2. Reproject all the input frames to a common coordinate system using equation (4.6). The result of this step becomes the initial condition $\hat{\mathbf{x}}^{(0)}$ SR mosaic (iteration 0).
3. While iteration *n* is less than the maximum number of iterations:
    a) Construct a set of *N* reconstructed frames based on $\hat{\mathbf{x}}^{(n)}$.
    b) Construct a set of *N* difference frames, subtracting the input frames from the set of frames in a).
    c) Construct a mosaic using Table 1 with the set of difference frames.
    d) Construct the regularization factor given by $\lambda^{(n)}\mathbf{G}^T\rho'(G\hat{\mathbf{x}}^{(n)}, \alpha)$.
    e) Subtract the result of c) from d) and then add to the previous SR mosaic $\hat{\mathbf{x}}^{(n-1)}$.

5.4.1 Experimental Results for Super-resolution Mosaicking Using Steepest Descent

We conducted four different tests, the first two are with synthetic data and the last two with real frames from UAS surveillance video. The parameter $\alpha^{(n))}$ was chosen to be 1.75 for the entire test, and the algorithm was set to run 10 iterations.

Results Using Synthetic Frames

We created synthetic LR frames from a single high resolution image. These LR frames where created using different translations (18 to 95 pixels), rotations (5^o to 10^o), and scales (1 to 1.5); we blurred the frames with a Gaussian kernel of size 3x3. Figure 5.3 shows the results of the proposed algorithm for two different sets of data. Figures 44 (a) and 44 (d) are the input LR mosaics, and Figures 44 (b) and 44 (e) are SR mosaics obtained by the proposed algorithm. Finally, Figures 44 (c) and 44 (f) are the mosaics generated using the high-resolution frames, these mosaic are going to be our ground truth for computing the PSNR. The results obtained by our algorithm are close to high-resolution mosaicking. Both set of images consist of five frames of 128x128x3 pixels.



(a) **(b)** **(c)**

(d) **(e)** **(f)**

Figure 44. Results of SR mosaicking for synthetic frames using steepest descent algorithm. The mosaic was constructed using five frames. Figures (a) and (d) show the mosaic for the first and second set of synthetic frames, respectively. These mosaics are the input to the algorithm. (b) and (e) are the super-resolved mosaics of (a) and (d) respectively. These mosaics are the output of the proposed algorithm. Figures (c) and (f) show the ground truth mosaics, which are the mosaics constructed using high resolution frames.

The PSNR was computed according to (5.12), resulting in a values of 43.86 dB

$$PSNR = 10\log_{10}\frac{255^2 N}{\left\|\mathbf{x}-\hat{\mathbf{x}}\right\|^2} \tag{5.12}$$

and 47.52 dB for the first and second sets of synthetic images, respectively Table 8 shows more results including the total time of processing all computations. This time also includes the computation of the mosaic. Figure 45 shows the evolution in every iteration of the Lagrange multiplier.

Table 8. Results of the computation of super-resolution mosaicking using steepest descent algorithm for two different sets of color synthetic frames.

Test	PSNR (dB)	Final error $\left\|\hat{\mathbf{x}}_{k+1}-\hat{\mathbf{x}}_k\right\|_2 \Big/ \left\|\mathbf{x}_k\right\|_2$	Total Processing Time on CPU (sec)
First set of five synthetic color frames.	43.86	0.006391	4.625
Second set of five synthetic color frames	47.52	0.0029404	3.875

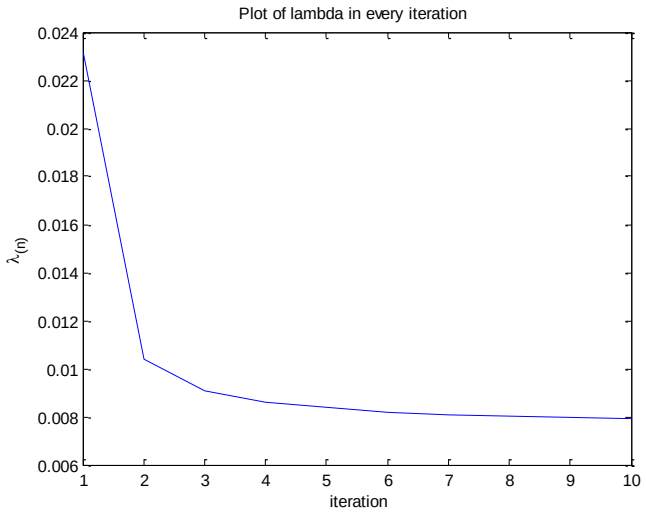


Figure 45. Regularization parameter (Lagrange multiplier) versus the number of iterations for the second set of synthetic color frames. The regularization parameter decreased as expected.

Results Using Real Frames from UAS

This section presents the results of the computation of super-resolution mosaicking for real UAS frames. There were three different tests performed: 1) using color frames, 2) using IR frames taken of buildings, and 3) using IR frames taken of vegetation. Figure 46 shows the results for the first test, where is clear to see significant visual improvement. The image on the left is the LR mosaic, and on the right is the SR mosaic. The SR mosaic contains more details, is less cloudy, and also the colors and textures are much better than the LR mosaic.

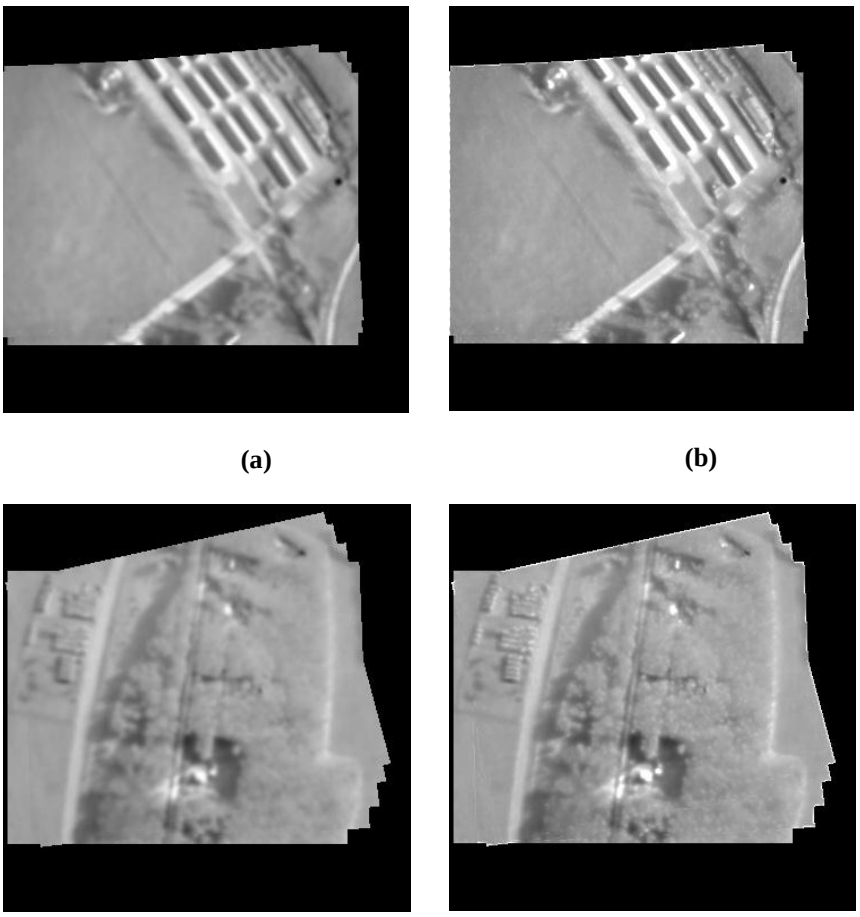
The LR mosaic was constructed using five different frames, courtesy of Cloud Cap Technology. The original size of the images were 640x480x3 pixels, but because of memory issues with the GNU Scientific Library (GSL), that is used to compute the homography, we had to downsample the images to 320x240x3 pixels.

Figure 46. Results of super-resolution mosaicking using the steepest descent algorithm. Left: LR mosaic. Right: the SR mosaic. The mosaic was constructed using five frames of size 320x240x3 pixels.

Figure 47 shows the results for the second and third tests. Both of these tests were applied to IR (infrared) frames captured in 2007 by the UASE Laboratory team at the University of North Dakota. The size of these frames is 320x240 pixels. Figure 48 (b, d) shows the LR mosaic and the SR mosaic for the second test. There is more detail in the SR mosaic, the buildings are sharper, and the trees have more texture. Figure 49(a-b,c-d) shows the LR mosaic and the SR mosaic for the third test. In this case, the difference is clearer, the SR mosaic is less cloudy and sharper and the trees have more texture. Figure 50 shows the evolution for every iteration of the Lagrange multiplier $\lambda^{(n)}$.

Table 9. Results of computing of super-resolution mosaics for two different sets of color synthetic frames.

Test	Final error $\left\|\hat{\mathbf{x}}_{k+1} - \hat{\mathbf{x}}_k\right\|_2 \Big/ \left\|\mathbf{x}_k\right\|_2$	Total Processing Time on CPU (sec)
Test #1: color frames of a road and forest.	0.002469	16.218
Test #2: IR frames of buildings	0.002004	10.844
Test #3: IR frames of a forest.	0.001018	16.688



(a) **(b)**

(c) **(d)**

Figure 47. Results of the SR mosaic for real frames from UAS using the steepest descent algorithm. The mosaic was constructed using five frames. Figures (a) and (c) show the mosaic for the first and second set of frames, respectively. These mosaics are the input to the algorithm. (b) and (d) are the super-resolved mosaics of (a) and (c), respectively. These mosaics are the output of the proposed algorithm.

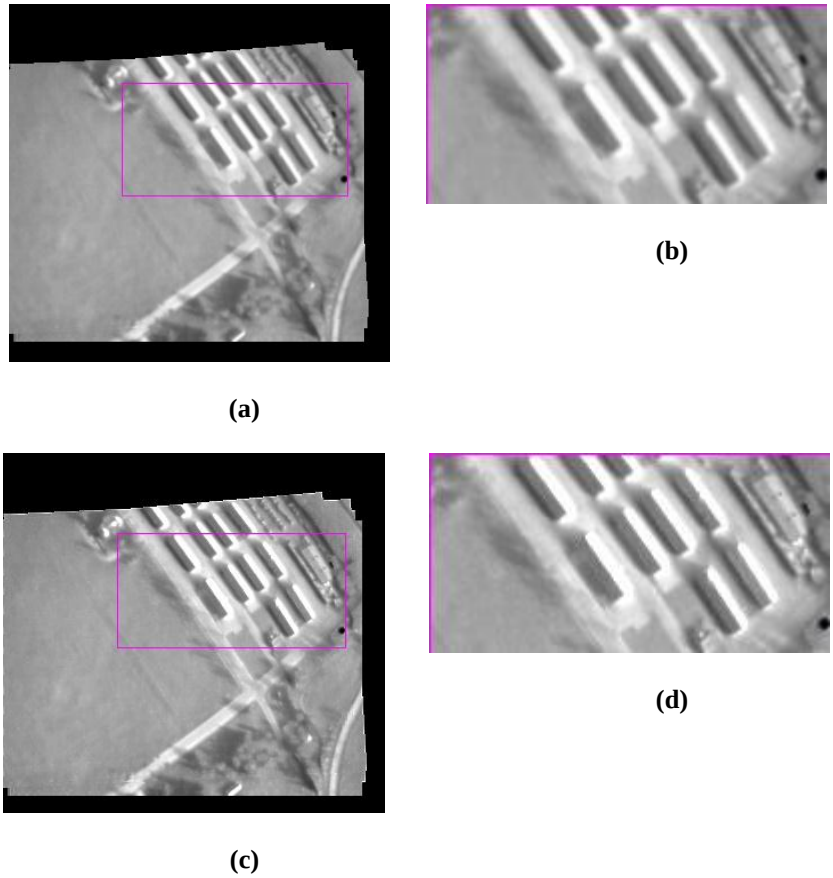
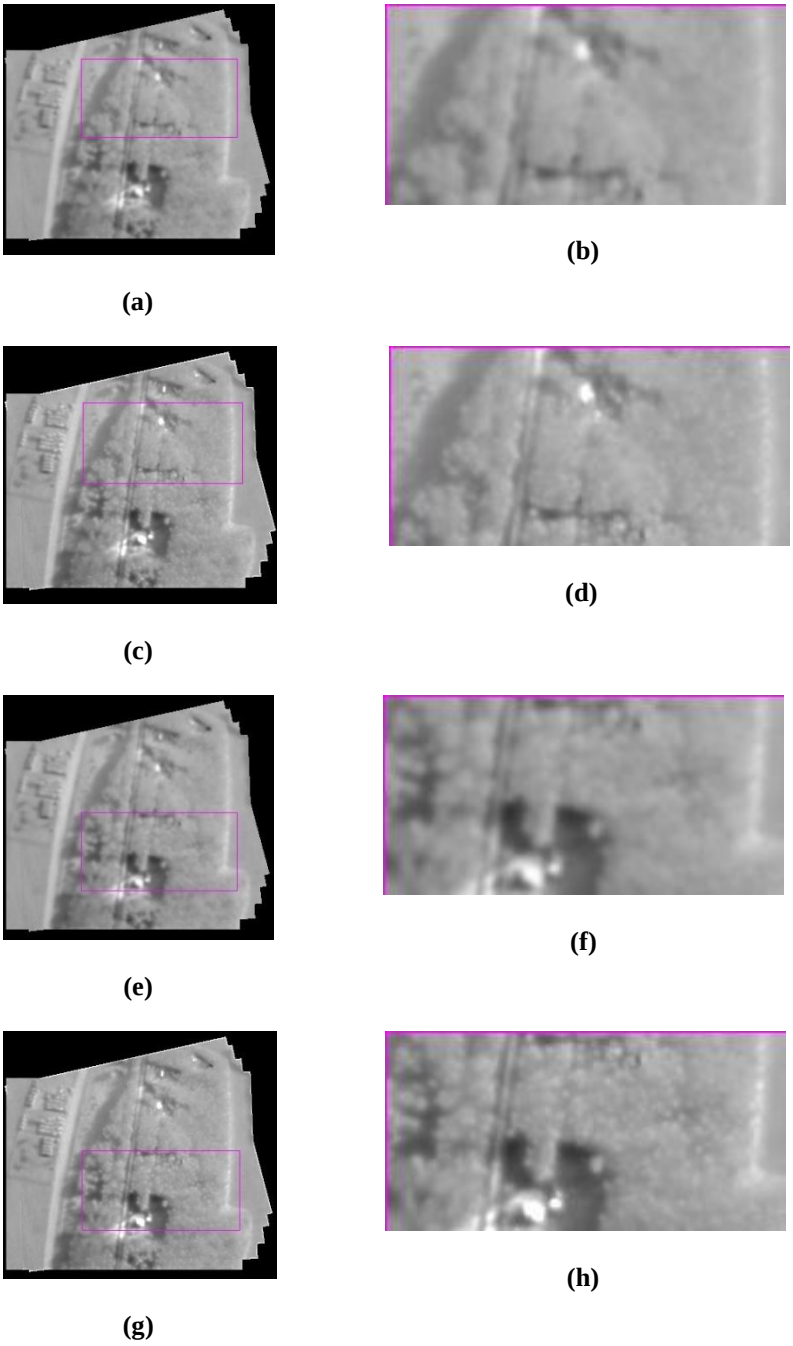


Figure 48. Region of Interest cropped to see a better comparison of the results of the algorithm for the first set of real UAS video frames. Figure (a) shows the region of interest selected from the whole LR mosaic, and (b) shows the selected LR area. Figure (c) shows the region of interest selected from the whole SR mosaic, and (d) shows the selected SR area.



(a) **(b)** **(c)** **(d)** **(e)** **(f)** **(g)** **(h)**

Figure 49. Region of Interest cropped to see a better comparison of the results of the algorithm for the second set of real UAS video frames. Figures (a) and (e) show the region of interest selected from the whole LR mosaic. Figures (b) and (f) show the region of interest selected from the whole LR mosaic. Figures (d) and (h) show the selected SR area.

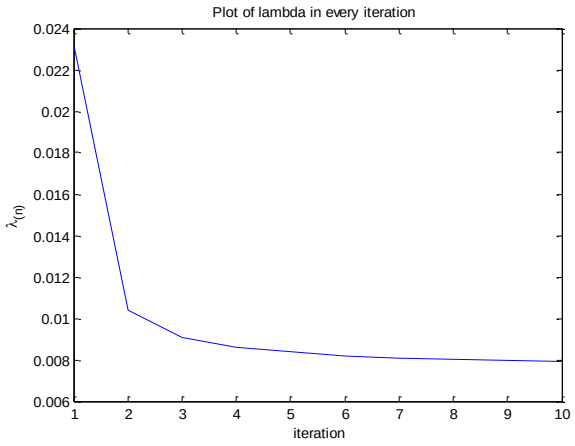


Figure 50. Regularization parameter $\lambda^{(n)}$ (Lagrange multiplier) versus the number of iterations for the second set of real IR video frames from UAS. The regularization parameter decreased as expected.

5.5 Super-resolution Mosaicking Using Conjugate Gradient

Based on the conjugate gradient method for minimizing (5.3), the robust iterative update for $\hat{\mathbf{x}}$ can be expressed as

$$\hat{\mathbf{x}}^{(n+1)} = \hat{\mathbf{x}}^{(n)} + \beta^{(n)}\mathbf{p}^{(n)}, \tag{5.13}$$

where $\mathbf{p}^{(n)}$ is chosen to be conjugate to all previous search directions with respect to the Hessian matrix $\mathbf{H}$:

$$\mathbf{p}^{(n)T}\mathbf{H}\mathbf{p}^{(j)}, \quad 0 \le j < n \tag{5.14}$$

Therefore, the resulting search directions are mutually linearly independent. $\mathbf{p}^{(n)}$ can be chosen using only knowledge of $\mathbf{p}^{(n-1)}$, $\nabla f(\hat{\mathbf{x}}^{(n-1)})$ and $\nabla f(\hat{\mathbf{x}}^{(n)})$ [79], given as

$$\mathbf{p}^{(n)} = \nabla f(\hat{\mathbf{x}}^{(n)}) + \left(\frac{\nabla f(\hat{\mathbf{x}}^{(n)})^T \nabla f(\hat{\mathbf{x}}^{(n)})}{\nabla f(\hat{\mathbf{x}}^{(n-1)})^T \nabla f(\hat{\mathbf{x}}^{(n-1)})} \right) \mathbf{p}^{(n-1)}. \tag{5.15}$$

The gradient vector, $\nabla f(\hat{\mathbf{x}}^{(n)})$,is given by the following expression:

$$\nabla f(\hat{\mathbf{x}}^{(n)}) = \mathbf{R}^T \left[\mathbf{W}_k^T \mathbf{B}_k^T \mathbf{D}^T (\mathbf{y}_k - \mathbf{D}\mathbf{B}_k \mathbf{W}_k \mathbf{R}[\hat{\mathbf{x}}^{(n)}]_k \right]_{k=1}^{K} - \lambda^{(n)} \mathbf{G}^T \rho'(G\hat{\mathbf{x}}^{(n)}, \alpha) \quad (5.16)$$

where $\mathbf{G}$ and $\lambda^{(n)}$ are computed in the same manner that for the case of the steepest descent algorithm. Table 10 details the implementation of the conjugate gradient for the computation of super-resolution mosaic.

5.5.1 Experimental Results for Super-resolution Mosaicking Using Conjugate Gradient

We conducted four different tests, the first two are with synthetic data and the last two with real frames from UAS surveillance video. The parameter $\beta^{(n)}$ was chosen to be 1.75 for the entire test, and the algorithm was set to run for 10 iterations. The results of both tests are show in the next sections.

Results Using Synthetic Frames

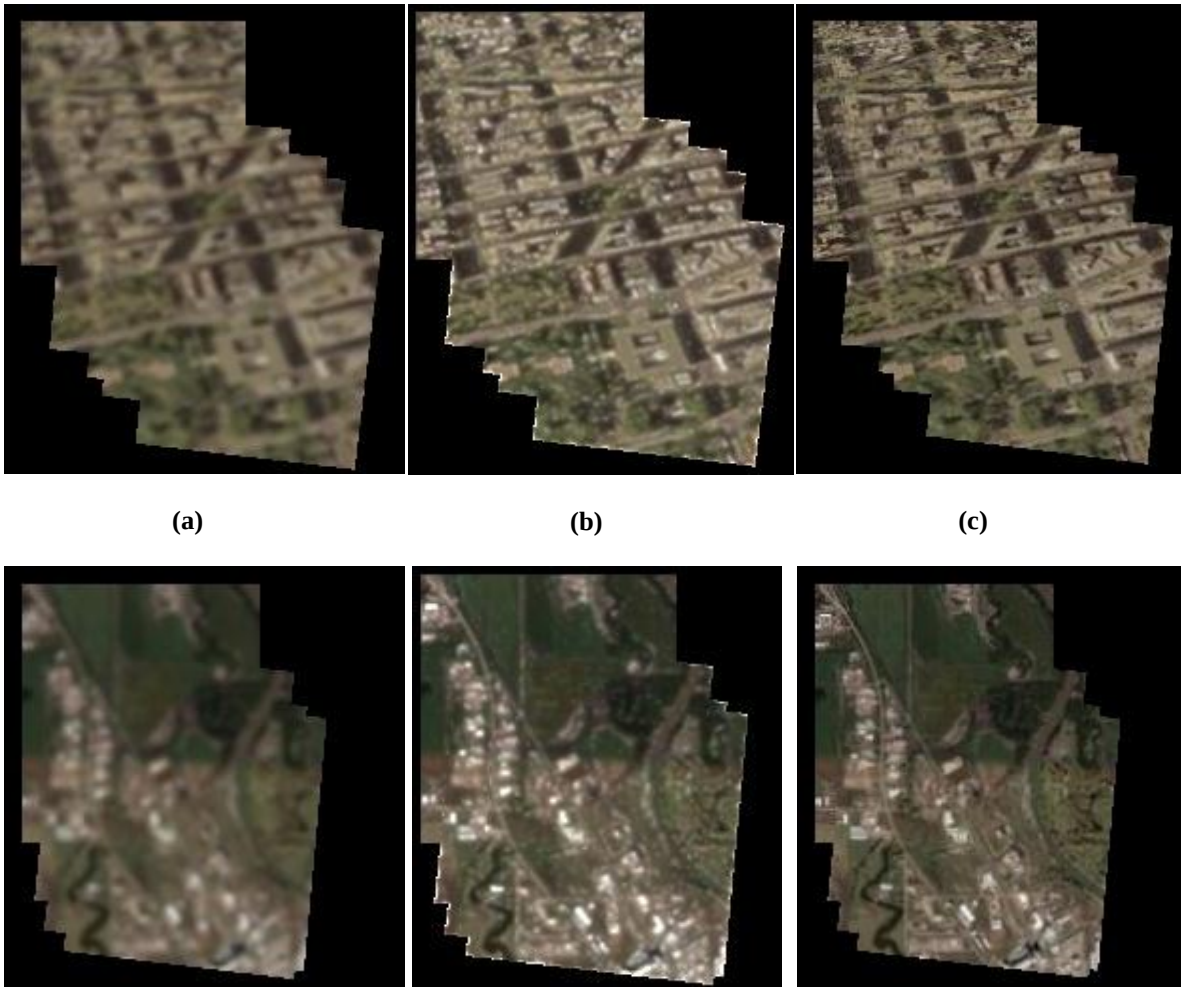
This section shows the results of the conjugate gradient algorithm for the same data set of synthetic images as Section 5.2.1.1. Figure 51 shows the results for two sets of synthetic color frames. Figure 51 (b) and 51 (e) are the SR mosaics obtained by the proposed CG algorithm. These images are very close to the ground truth; the color, texture, and sharpness are recovered in most of the images. Table 11 shows the PSNR, execution time, and final error for both sets of images. According to this table, the results of using CG are slightly better in comparison with SD in quality, but it takes more computation time.

Table 10. Algorithm to construct a super-resolved mosaic using conjugate gradient algorithm given a set of *N* input frames.

Objective: Construct a super-resolved mosaic using conjugate gradient algorithm from a set of *N* input frames.

Algorithm:

1. Compute the homography using Table 1 between consecutive frames; i.e., $h_{12}, h_{23}, ..., h_{N-1,N}$.
2. Reproject all the input frames to a common coordinate system using equation (4.6). The result of this step becomes the initial condition of the $\hat{\mathbf{x}}^{(0)}$ SR mosaic (iteration 0), and it is used to compute the gradient for iteration 0 using (5.16).
3. While iteration *n* is less than the maximum number of iterations:
    a) Construct the gradient using equation (5.16).
    b) Construct $\mathbf{p}^{(n)}$ using equation (5.15).
    c) Update $\hat{\mathbf{x}}^{(n)}$, using equation (5.13).



(a) **(b)** **(c)**

(d) **(e)** **(f)**

Figure 51. Results of SR mosaicking for synthetic frames using the conjugate gradient algorithm. The mosaics were constructed using five frames. Figures (a) and (d) show the mosaics for the first and second set of synthetic frames respectively. These mosaics are the input to the algorithm. (b) and (e) are the super-resolved mosaics using the CG algorithm on (a) and (d), respectively. These mosaics are the output of the proposed algorithm. Figures (c) and (f) show the ground truth mosaics, which are the mosaics constructed using high-resolution frames.

Table 11. Results of computating super-resolution mosaics using the conjugate gradient algorithm for two different set of color synthetic frames.

Test	PSNR (dB)	Final error $\left\|\hat{\mathbf{x}}_{k+1}-\hat{\mathbf{x}}_k\right\|_2 \Big/ \left\|\mathbf{x}_k\right\|_2$	Total Processing Time on CPU (sec)
First set of five synthetic color frames.	43.98	0.004381	5.047
Second set of five synthetic color frames	47.54	0.006212	4.250

Results Using Real Frames from UAS

This section shows the results of the conjugate gradient method for the same data set of real frames captured by UAS as described in Section 5.2.1. Figure 52 shows the results applying the conjugate gradient algorithm to super-resolution mosaicking for a set of five color frames from a UAS. The results are similar as that for the case of steepest descent method, but the color and sharpness show a slight improvement. Table 12 shows more details of the results with the three data sets. This table shows the final error and the total processing time in seconds for the three tests.

Table 12. Results of the capturing the super-resolution mosaics using the proposed conjugate gradient algorithm for three different sets real frames from UAS.

Test	Final error $\left\|\hat{\mathbf{x}}_{k+1}-\hat{\mathbf{x}}_k\right\|_2 \Big/ \left\|\mathbf{x}_k\right\|_2$	Total Processing Time on CPU (sec)
Test #1: color frames of a road and forest.	0.005055	16.891
Test #2: IR frames of buildings	0.003379	11.907
Test #3: IR frames of a forest.	0.003655	11.219

Figure 52. Results of super-resolution mosaicking using the conjugate gradient algorithm. Left: LR mosaicking. Right: SR mosaic. The mosaics were constructed using five frames of a size of 320x240x3 pixels.

Figure 53 shows the results of using conjugate gradient for computing SR mosaic for real IR video frames. The results are now clearer than for steepest descent method. The SR mosaic has more details, even the shape of the buildings are more clear (see Figure 53 (b)); also, the trees are sharper.

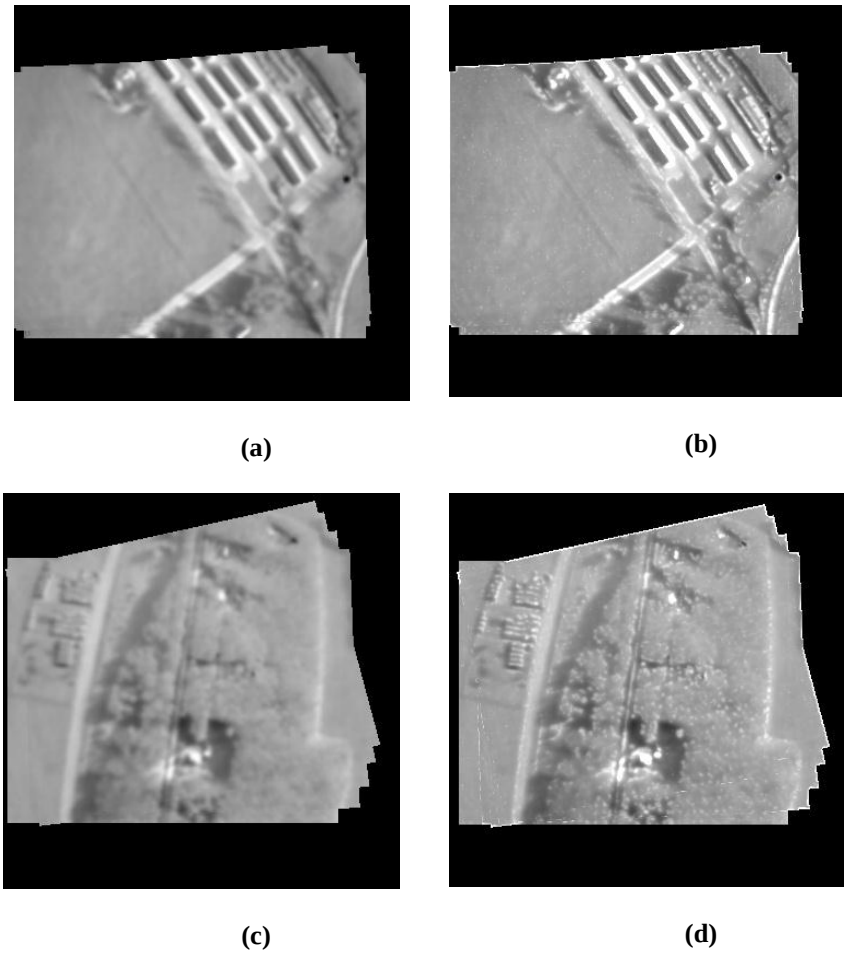


Figure 53 . Results of the SR mosaic for real frames from UAS using the conjugate gradient method. The mosaic was constructed using five frames. Figures (a) and (c) show the mosaics for the first and second set of frames, respectively. These mosaics are the input to the algorithm; (b) and (d) are the super-resolved mosaics of (a) and (c), respectively. These mosaics are the output of the proposed conjugate gradient method.

5.6 Super-resolution Mosaicking Using Levenberg Marquardt

The Levenberg Marquardt (LM) method was proposed by [1,97] as a new method to solve nonlinear problems. This algorithm shares with gradient methods their ability to converge from an initial guess which may be outside of the region of convergence of other methods. Based on the Levenberg Marquardt method for minimizing (5.3), and defining $f(\mathbf{x})$ as

$$f(x) = \left\| \sum_{k=1}^{K} \left(y_k - \mathbf{D}\mathbf{B}_k \mathbf{W}_k \mathbf{R}[\mathbf{x}]_k \right) \right\|_2^2 + \lambda \sum_{g \in G_x} \rho(g, \alpha),$$

$$f(\mathbf{x} + \delta\mathbf{x}) \approx f(\mathbf{x}) + J\delta\mathbf{x}, \tag{5.17}$$

where $J(\mathbf{x})$ is given as the Jacobian matrix:

$$J = \partial \frac{f(\mathbf{x})}{\partial \mathbf{x}} = \partial \frac{\left\{ \left\| \sum_{k=1}^{K} \mathbf{y}_k - \mathbf{D}\mathbf{B}_k \mathbf{W}_k \mathbf{R}[\mathbf{x}]_k \right\|_2^2 + \lambda \sum_{g \in G_x} \rho(g, \alpha) \right\}}{\partial \mathbf{x}} \tag{5.18}$$

$$J = \mathbf{R}^T \left[\mathbf{W}_k^T \mathbf{B}_k^T \mathbf{D}^T (\mathbf{y}_k - \mathbf{D}\mathbf{B}_k \mathbf{W}_k \mathbf{R}[\hat{x}^{(n)}]_k \right]_{k=1}^{K} - \lambda^{(n)} G^T \rho'(G\hat{\mathbf{x}}^{(n)}, \alpha) \tag{5.19}$$

The Levenberg Marquardt method is iterative. Initiated at the starting point $\hat{\mathbf{x}}^{(0)}$, the method requires finding $\delta\mathbf{x}$ that minimizes

$$\left\| \hat{\mathbf{x}} - f(\hat{\mathbf{x}} + \delta\mathbf{x}) \right\| \approx \left\| \hat{\mathbf{x}} - f(\mathbf{x}) - J\delta\mathbf{x} \right\| = \left\| \varepsilon - J\delta\mathbf{x} \right\| \tag{5.20}$$

$\delta\mathbf{x}$ is found by solving a linear least squares problem. The minimum is attained when $J\delta\mathbf{x} - \boldsymbol{\varepsilon}$ is orthogonal to the column space of J. This leads to:

$$J^T J\delta\mathbf{x} = J^T\boldsymbol{\varepsilon}$$
$$H^*\delta\mathbf{x} = J^T\boldsymbol{\varepsilon} \tag{5.21}$$

where H^* is called the pseudo Hessian, defined as $H^* = J^T J$ Levenberg Marquardt solves equation (5.21), adding a damping term to the diagonal elements of H^*. Therefore, the Levenberg Marquardt equation is

$$(H^* + cI)\delta\mathbf{x} = J^T\boldsymbol{\varepsilon}, \tag{5.22}$$

where $\delta\mathbf{x}$ is found by solving

$$\delta\mathbf{x} = \arg\min_{\delta\mathbf{x}} \left\| \left(H^* + cI\right)\delta\mathbf{x} - J^T\boldsymbol{\varepsilon} \right\|_2 . \tag{5.23}$$

Then,

$$\hat{\mathbf{x}}^{(n+1)} = \hat{\mathbf{x}}^{(n)} + \delta\mathbf{x}, \tag{5.24}$$

where c is the Levenberg Marquardt damping term that determines the behavior of the gradient in each iteration. If c is close to zero, then the algorithm behaves like a Gauss Newton (GN) method, but if $c \to \infty$, then the algorithm behaves like the steepest descent (SD) algorithm. The values of c during the iterative process are chosen in the following way: at the beginning of the iterations, c is set to a large value, so that the LM method uses the robustness of SD, and the initial guess of the solution to (5.3) can be chosen with less caution. It is necessary to save the value of the errors for each iteration, and do the comparison between two consecutive errors. In the case that $error_{(k)} < error_{(k-1)}$, c is decreased by a certain amount so that LM behaves like Gauss-Newton to take advantage of the speed up to convergence. Otherwise, c increases to a larger value, thus increasing the searching area, which means that LM behaves like SD. The $error_{(k)}$ is defined as

$$error_{(k)} = \frac{\left\| \hat{\mathbf{x}}_{k+1} - \hat{\mathbf{x}}_k \right\|_2}{\left\| \hat{\mathbf{x}}_k \right\|_2} \tag{5.25}$$

The following section shows the experimental results with synthetic and real data from UAS.

5.6.1 Results Using Synthetic Frames

This section shows the results of the Levenberg Marquardt method for the same data set of synthetic images as shown in Section 5.2. Figure 54 shows the results for the two sets of synthetic color frames. Figures 54 (b) and 54 (e) are the SR mosaics obtained by the proposed algorithm based on Levenberg Marquardt. These images are very close to the ground truth as well, even the color is recovered.

Table 13 shows the algorithm used to compute the super-resolution mosaic using Levenberg Marquardt algorithm, this table shows the PSNR, final error obtained in ten iterations and the total processing time in seconds.

Table 14 shows the algorithm to compute super-resolution mosaic using Levenberg Marquardt

Table 13. Results computing super-resolution mosaics using Levenberg Marquardt algorithm for two different sets of color synthetic frames.

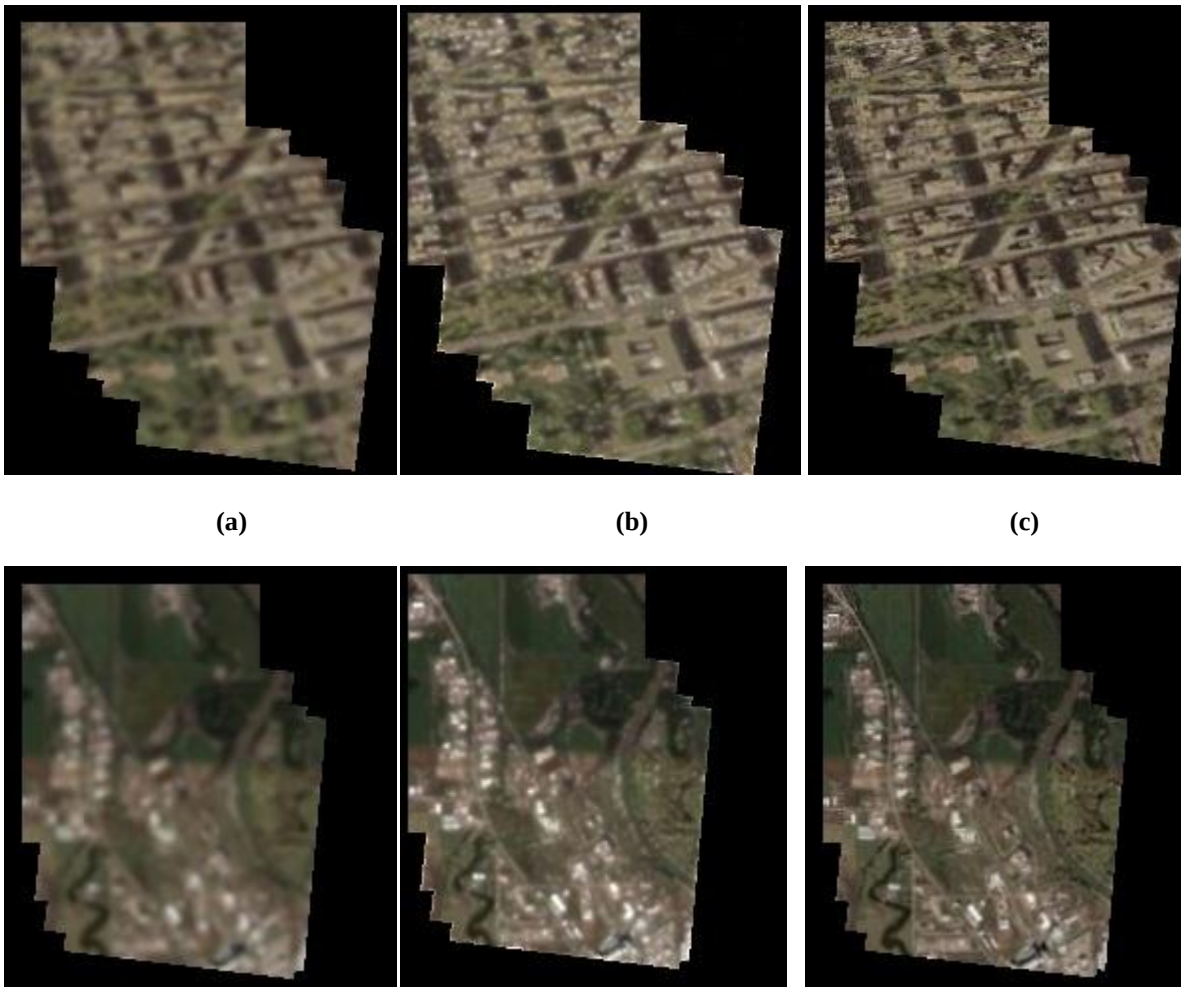
Test	PSNR (dB)	Final error $\left\| \hat{\mathbf{x}}_{k+1} - \hat{\mathbf{x}}_k \right\|_2 \Big/ \left\| \mathbf{x}_k \right\|_2$	Total Processing Time on CPU (sec)
First set of five synthetic color frames.	43.77	0.002833	5.109
Second set of five synthetic color frames	47.45	0.002505	4.281

Table 14. Algorithm to construct a super-resolved mosaic using the Levenberg Marquardt optimization method given a set of *N* input frames.

Objective: Construct a super-resolved mosaic using the Levenberg Marquardt optimization method from a set of *N* input frames.

Algorithm:

1. Compute the homography using Table 1 between consecutive frames; i.e., $h_{12}, h_{23}, ..., h_{N-1,N}$.
2. Reproject all the input frames to a common coordinate systems using equation (4.6). The result of this step becomes the initial condition of the $\hat{\mathbf{x}}^{(0)}$ SR mosaic (iteration 0).
3. While iteration *n* is less than the maximum number of iterations:
    a) Construct the Jacobian using equation (5.19).
    b) Construct the pseudo Hessian matrix given by $\mathbf{H}^{*} = J^{T} J$.
    c) Solve the linear least squares equation using the singular value decomposition (SVD) for $\delta\mathbf{x}$, in equation (5.23).
    d) Update $\hat{\mathbf{x}}^{(n)}$ in equation (5.24).



(a) **(b)** **(c)**

(d) **(e)** **(f)**

Figure 54. Results of SR mosaicking for synthetic frames using the Levenberg Marquardt method. The mosaic was constructed using five frames. Figures (a) and (d) show the mosaic for the first and second sets of synthetic frames, respectively. These mosaics are the input to the algorithm. (b) and (e) are the super-resolved mosaics applying the LM method to (a) and (d), respectively. These mosaics are the output of the proposed algorithm. Figures (c) and (f) show the ground truth mosaics, which are the mosaics constructed using high-resolution frames.

5.6.2 Results Using Real Frames from UAS

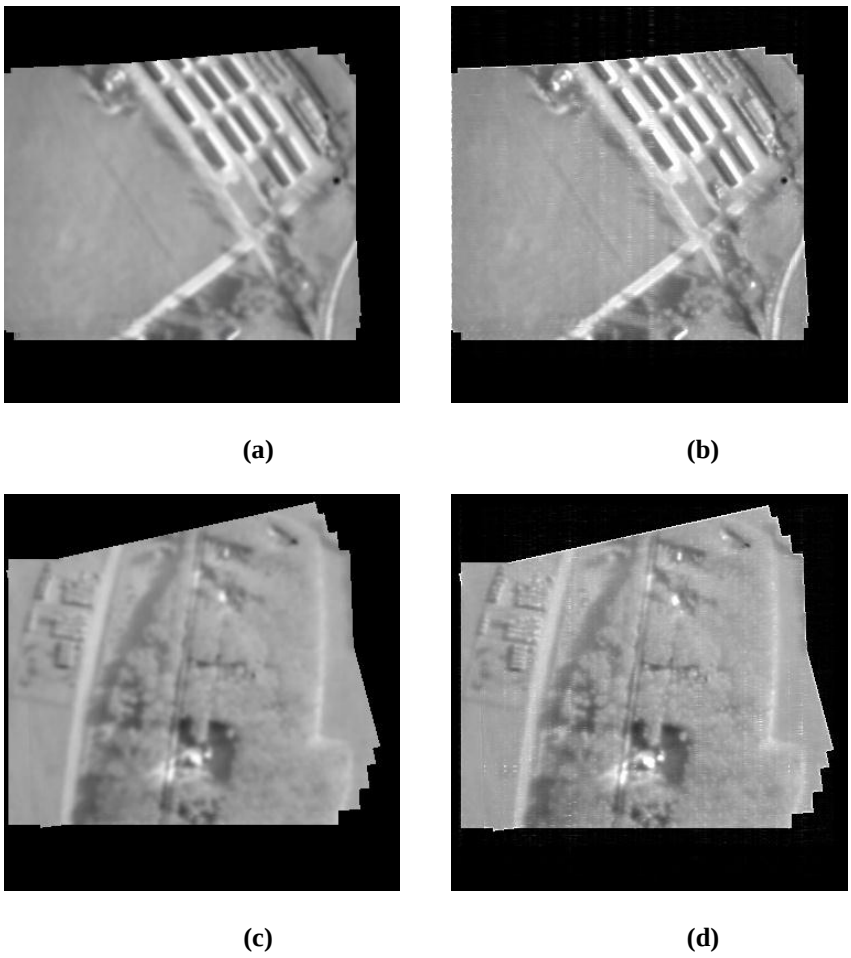
This section shows the results of the Levenberg Marquardt algorithm for the same data set of real frames captured by UAS in Section 5.2.1.2. Figure 55 shows the results applying the Levenberg Marquardt method for super-resolution mosaicking to a set of five color frames from a UAS. The results are similar to those for the case of steepest descent, but the color and sharpness have better improvement. But, there are also some artifacts introduced by the solution of the equation (5.23); the reason of this is because the pseudo Hessian is close to be singular. Table 15 shows more details of the results with the three data sets (three different tests).

Table 15. Results of computing the super-resolution mosaics using the proposed Levenberg Marquardt algorithm for three different sets real frames from UAS.

Test	Final error $\left\|\hat{\mathbf{x}}_{k+1} - \hat{\mathbf{x}}_k\right\|_2 \Big/ \left\|\mathbf{x}_k\right\|_2$	Total Processing Time on CPU (sec)
Test #1: color frames of a road and forest.	0.005424	17.485
Test #2: IR frames of buildings.	0.003514	11.766
Test #3: IR frames of a forest.	0.004298	11.391

Figure 55. Result of super-resolution mosaicking using the proposed Levenberg Marquardt method. Left: LR mosaicking. Right: SR mosaicking. The mosaics were constructed using five frames of size of 320x240x3 pixels.

Figure 56 shows the results of using the Levenberg Marquardt algorithm for real IR video frames. The SR mosaic has more details, even the shape of the buildings are more clear (see Figure 56 (b)); also, the trees are more sharp. However, there are some also artifacts since the Levenberg Marquardt requires the pseudo Hessian which has a sparse structure and is close to singular. Note that these artifacts are not produced in the real color frames from the UAS video (Figure 55).



(a) **(b)**

(c) **(d)**

Figure 56. Results of the SR mosaic for real frames from UAS using the proposed Levenberg Marquardt method. The mosaic was constructed using five frames. Figures (a) and (c) show the mosaic for the first and second set of frames, respectively. These mosaics are the input to the algorithm. (b) and (d) are the super-resolved mosaics of (a) and (c) respectively. These mosaics are the output of the proposed LM algorithm.

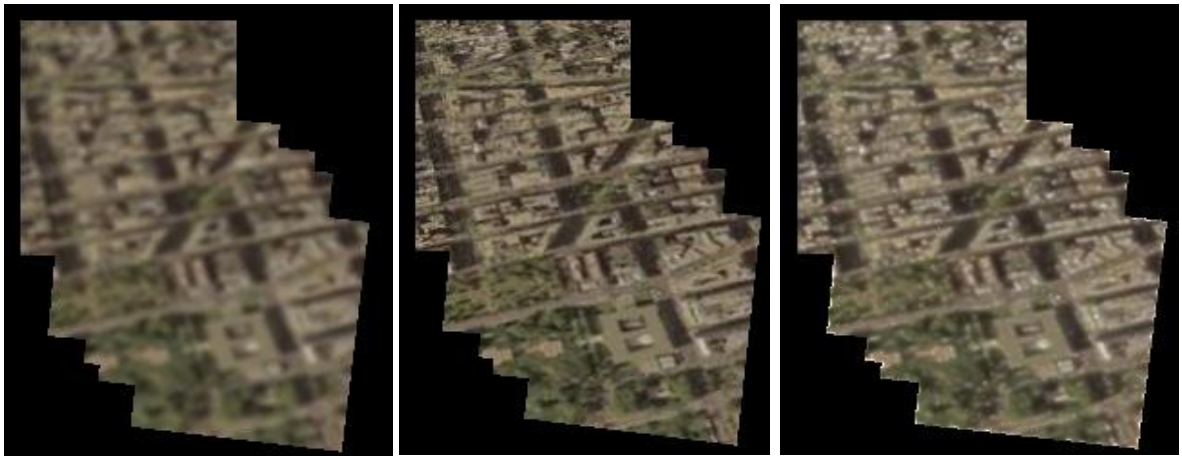
5.7 Comparison of metrics for Super-Resolution Mosaicking by the three algorithms

This section compares the results of the three proposed algorithms for super-resolution mosaicking. The comparison will be based on PSNR, time, and error for the synthetic data sets, and time and error for the real frames from UAS because there is no ground truth data available to compute the PSNR with real frames.

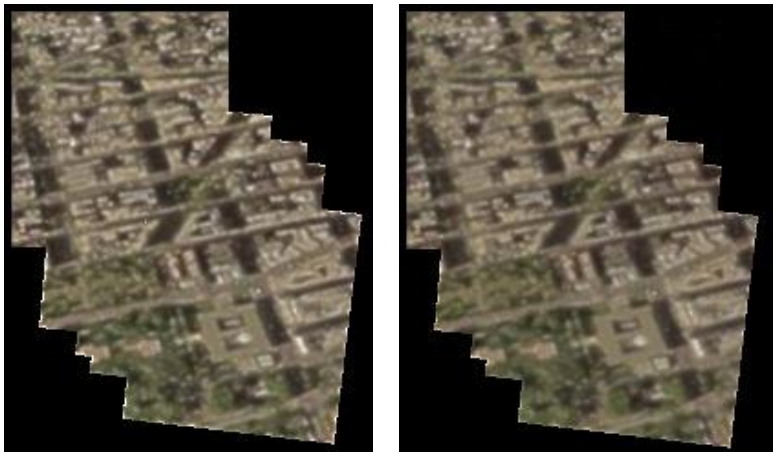
From the comparisons shown in Figures 57 to 61 and from Tables 16 to 20, all the methods improve the resolution of the LR mosaic, and all of them improve the color, details, and sharpness. But, when the image is black and white (IR images), the Levenberg Marquardt produces some artifacts since it solves linear squares equation that is close to be singular (5.23). The final error for the steepest descent and conjugate gradient algorithms decreases with every iteration, which means that they find the optimal solution in every iteration; but for the Levenberg Marquardt algorithm, this error can decrease or increase due to the use of the damping factor, c, which accelerates the search for the optimal solution.

Table 16. Comparison of the three proposed algorithms to compute super-resolution mosaics for the first set of synthetic color frames.

Algorithm	PSNR (dB)	Final error $\left\|\hat{\mathbf{x}}_{k+1} - \hat{\mathbf{x}}_k\right\|_2 \Big/ \left\|\mathbf{x}_k\right\|_2$	Total Processing Time on CPU (sec)
Super-resolution using steepest descent algorithm.	43.86	0.006391	4.625
Super-resolution using conjugate gradient algorithm.	43.98	0.004381	5.047
Super-resolution using Levenberg Marquardt algorithm.	43.77	0.002833	5.422

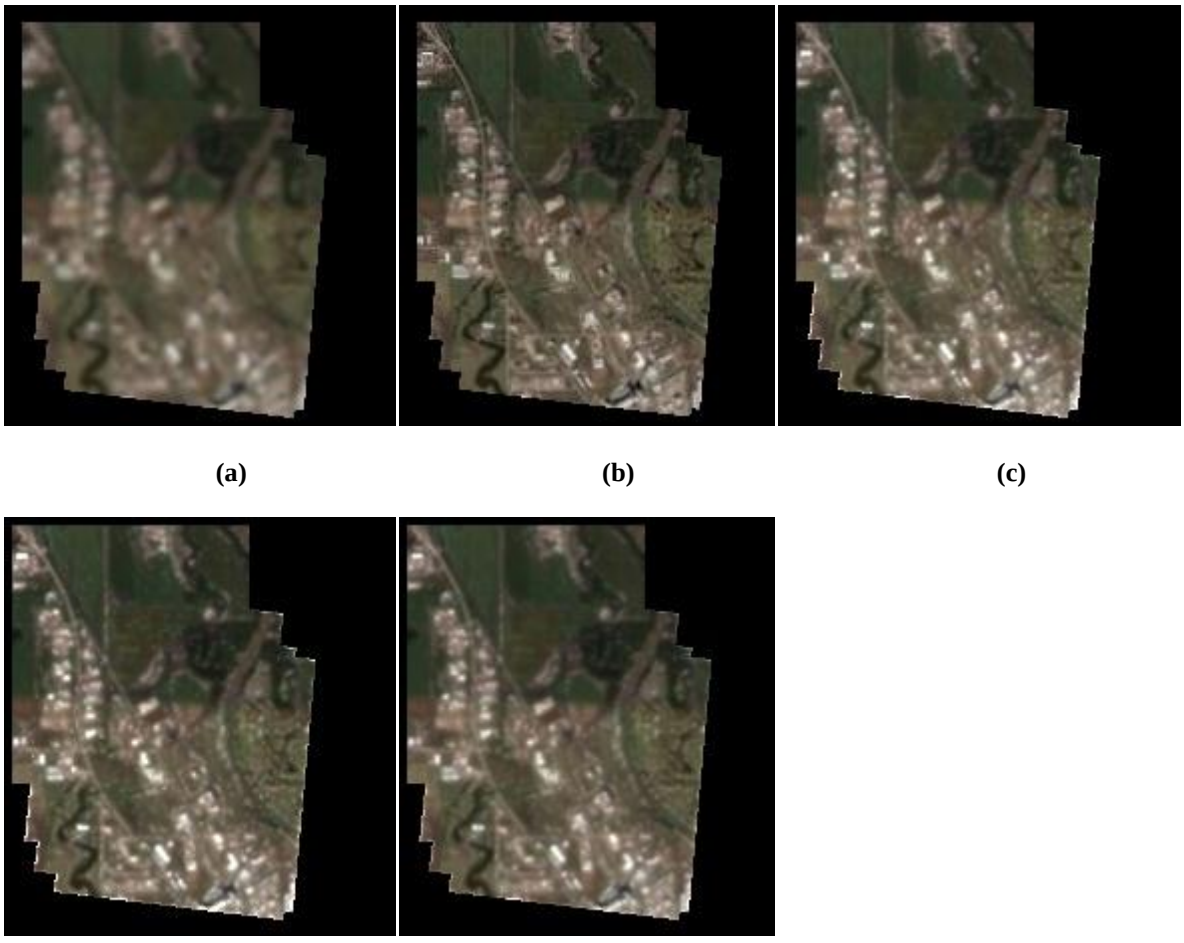


(a) **(b)** **(c)**



(d) **(e)**

Figure 57. Comparison of the three proposed algorithms: steepest descent, conjugate gradient, and Levenberg Marquardt. These images belong to the first set of synthetic frames created. (a) LR mosaic. (b) Ground truth HR mosaic. (c) SR mosaic using steepest descent. (d) SR mosaic using conjugate gradient. (e) SR mosaic using Levenberg Marquardt.



(a) **(b)** **(c)**

(d) **(e)**

Figure 58. Comparison of the three proposed algorithms: steepest descent, conjugate gradient and Levenberg Marquardt. These images belong to the second set of synthetic frames created. (a) LR mosaic. (b) Ground truth HR mosaic. (c) SR mosaic using steepest descent. (d) SR mosaic using conjugate gradient. (e) SR mosaic using Levenberg Marquardt.

Table 17. Comparison of the three proposed algorithms to compute super-resolution mosaics for the second set of synthetic color frames.

Algorithm	PSNR (dB)	Final error $\left\|\hat{\mathbf{x}}_{k+1}-\hat{\mathbf{x}}_k\right\|_2 \Big/ \left\|\mathbf{x}_k\right\|_2$	Total Processing Time on CPU (sec)
Super-resolution using steepest descent algorithm.	47.52	0.0029404	3.875
Super-resolution using conjugate gradient algorithm.	47.54	0.006212	4.250
Super-resolution using Levenberg Marquardt algorithm.	47.45	0.002505	4.281

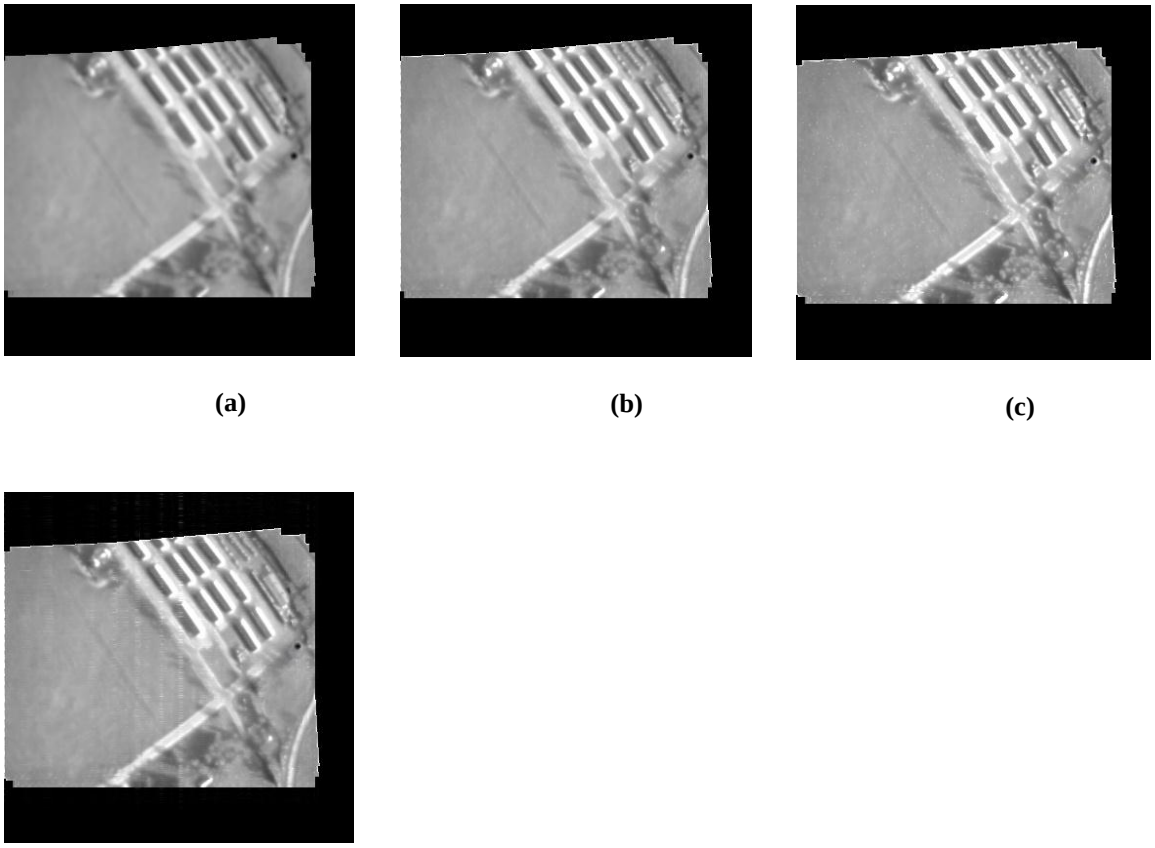
Table 18. Comparison of the three proposed algorithms to compute super-resolution mosaics for the first set of real video color frames captured by UAS.

Algorithm	Final error $\left\|\hat{\mathbf{x}}_{k+1}-\hat{\mathbf{x}}_k\right\|_2 \Big/ \left\|\mathbf{x}_k\right\|_2$	Total Processing Time on CPU (sec)
Super-resolution using steepest descent method.	0. 002469	16.218
Super-resolution using conjugate gradient method.	0.005055	16.891
Super-resolution using Levenberg Marquardt method.	0.005424	17.485

(a) **(b)** **(c)**

(d)

Figure 59. Comparison of the three proposed algorithms: steepest descent, conjugate gradient, and Levenberg Marquardt. The images belong to the first set of color video frames captured from UAS. (a) LR mosaic. (b) SR mosaic using steepest descent. (c) SR mosaic using conjugate gradient. (d) SR mosaic using Levenberg Marquardt.



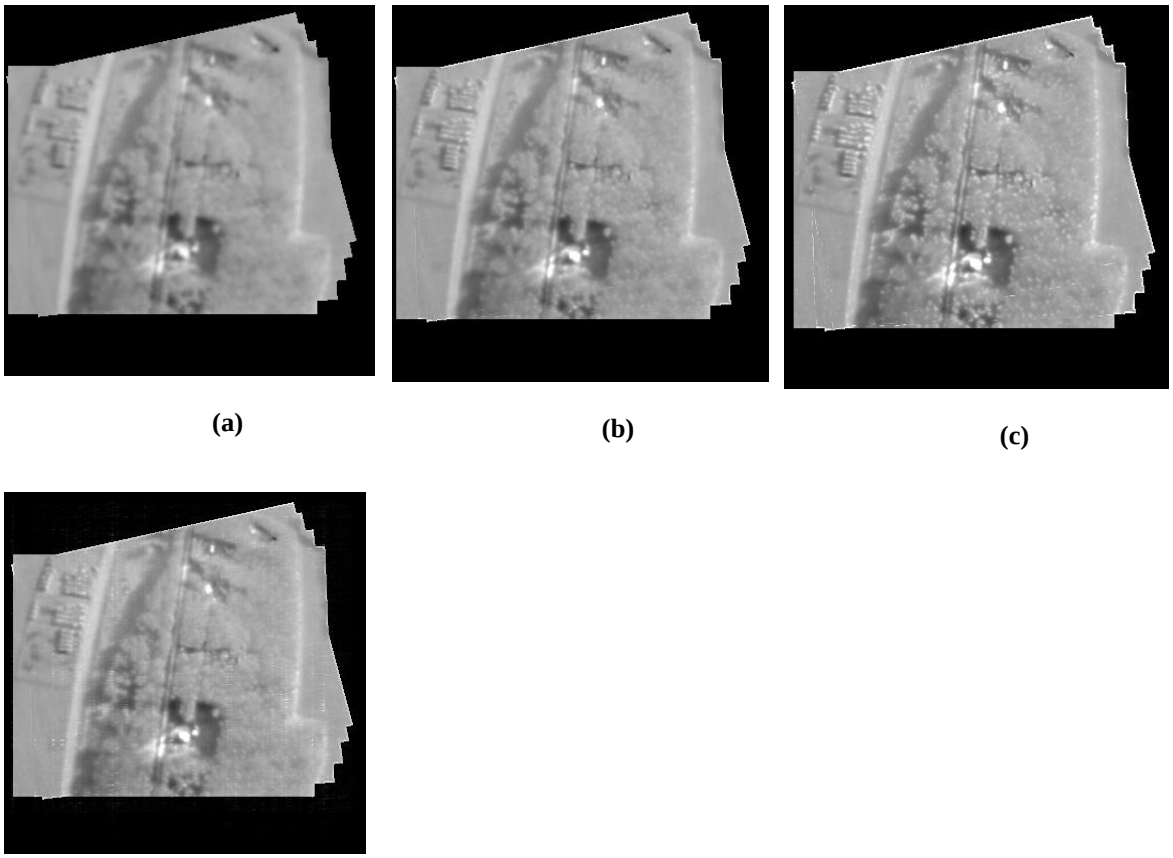
(a) **(b)** **(c)**

(d)

Figure 60. Comparison of the three proposed algorithms: steepest descent, conjugate gradient, and Levenberg Marquardt. These images belong to the first set of real IR video frames captured from UAS. (a) LR mosaic. (b) SR mosaic using steepest descent. (c) SR mosaic using conjugate gradient. (d) SR mosaic using Levenberg Marquardt.

Table 19. Comparison of the three proposed algorithms to compute super-resolution mosaics for the first set of real video IR frames captured by UAS.

Algorithm	Final error $\left\|\hat{\mathbf{x}}_{k+1} - \hat{\mathbf{x}}_k\right\|_2 \Big/ \left\|\mathbf{x}_k\right\|_2$	Total Processing Time on CPU (sec)
Super-resolution using steepest descent algorithm.	0.065014	10.844
Super-resolution using conjugate gradient algorithm.	0.097590	11.907
Super-resolution using Levenberg Marquardt algorithm.	0.068155	11.750



(a) **(b)** **(c)**

(d)

Figure 61. Comparison of the three proposed algorithms: steepest descent, conjugate gradient and Levenberg Marquardt methods. These images belong to the second set of real IR video frames captured from UAS. (a) LR mosaic. (b) SR mosaic using steepest descent method. (c) SR mosaic using conjugate gradient. (d) SR mosaic using Levenberg Marquardt.

Table 20. Comparison of the three proposed algorithms to compute super-resolution mosaics for the second set of real video IR frames captured by UAS.

Algorithm	Final error $\left\|\hat{\mathbf{x}}_{k+1}-\hat{\mathbf{x}}_k\right\|_2 \Big/ \left\|\mathbf{x}_k\right\|_2$	Total Processing Time on CPU (sec)
Super-resolution using steepest descent algorithm.	0.001018	16.688
Super-resolution using conjugate gradient algorithm	0.003655	11.907
Super-resolution using Levenberg Marquardt algorithm	0.004298	11.391

5.8 Conclusions

Three algorithms, based on steepest descent, conjugate gradient and Levenberg Marquardt are introduced for the computation of the super-resolution mosaicking. Levenberg Marquardt is a novel algorithm in this regard. All these algorithms use the same prior and Lagrange multiplier based on the Huber function. The computation of the super-resolution mosaic for all the proposed methods takes five low-resolution frames as input, then constructs the mosaic, applies the regularization factor, and finally applies the corresponding optimization method to solve it. Therefore, the proposed methods are complete, robust, adaptive, and independent.

The results show that the three proposed algorithms work not only with synthetic images, but also with real images in both color and gray pixel levels. Furthermore, the size of the test frames are standard sizes for video frames. The processing time is also reduced, which makes these algorithms capable of dealing with real data in real applications.

CHAPTER 6

GPU-CPU IMPLEMENTATION FOR VIDEO MOSAICKING AND SUPER-RESOLUTION MOSAICKING

6.1 Introduction

The construction of video mosaics and the super-resolution reconstruction of mosaics for a set of color (EO) and IR frames has been analyzed and explained in previous chapters. The computations of those results all used CPU (Central Processing Unit) processing. This chapter implements the video mosaicking and super-resolution mosaicking over GPU (Graphical Process Unit) and CPU processors, using the same algorithms of Chapters 4, and 5.

In order to create a "good" video mosaic and super-resolution mosaic, the registration between the frames should be accurate at the sub-pixel level. SIFT (Scale Invariant Feature Transform) has been chosen because of its robustness, invariance to change of illumination, and highly distinctive features, all of which help in the matching process [70]. Once the SIFT features are found, it is necessary to match them and find the best transformation matrix between them called a homography. All this registration process takes a great deal of computational resources, serving as the computation bottleneck of the whole process for video (image) mosaicking and super-resolution mosaicking.

This chapter implements the registration process on GPU and the NVIDIA CUDA^{TM} technology. The GPU is a highly parallel, multithread, many core processor

with tremendous computational horsepower and very high memory bandwidth. Algorithms developed using the GPU as programming platforms are commonly referred as GPGPU (General Purpose Computation on GPU) [2]. The speed up of the registration process is up to 54 times faster than using CPU technology. All the results that we show in this section were obtained on with a desktop Dell computer with the GeForce 9800 GT GPU card installed.

Section 6.2 explains the use of GPU and CUDA™ technology, Section 6.3 shows the implementation of GPU-CPU for the computation of video mosaics based on the algorithms in Chapter 4, and the use of I-frames generated for MPEG video. Section 6.4 shows the results for the computation of super-resolution mosaics using GPU-CPU and a performance comparison between the results obtained using GPU-CPU with only CPU (extracted from Chapter 5).

6.2 GPU Programming Paradigm

The use of the graphical processing unit (GPU) to accelerate general-purpose computations has become an important technique in scientific research. Graphics hardware has evolved tremendously over the last several years. It started with basic polygon rendering via 3dfx's Voodoo Graphics in 1996, and continued with custom vertex manipulations four years later. Within ten years, the GPU increased its speed by approximately 750 times (1996 to 2006), and is still growing exponentially each year. Conversely, CPU performance doubles only every 22 months.

GPU is designed for math-intensive, parallel problems (see Figure 62). More specifically, the GPU is especially well-suited to address problems that can be expressed as data-parallel computations. Because the same program is executed for each data

element, there is a lower requirement for sophisticated flow control, and the memory latency can be hidden with calculations instead of large data caches. CUDA ™ was introduced by NVIDIA in November 2006. CUDA™ is a C-based general purpose parallel computing architecture, with a new parallel programming model and instruction set architecture, which leverages the parallel compute engine in NVIDIA GPUs.

Successful use of GPU for general purpose computation requires taking into account the significant overhead incurred in executing and managing GPU kernels, which includes queuing, overhead, scheduling overhead, and the GPU progress check period.

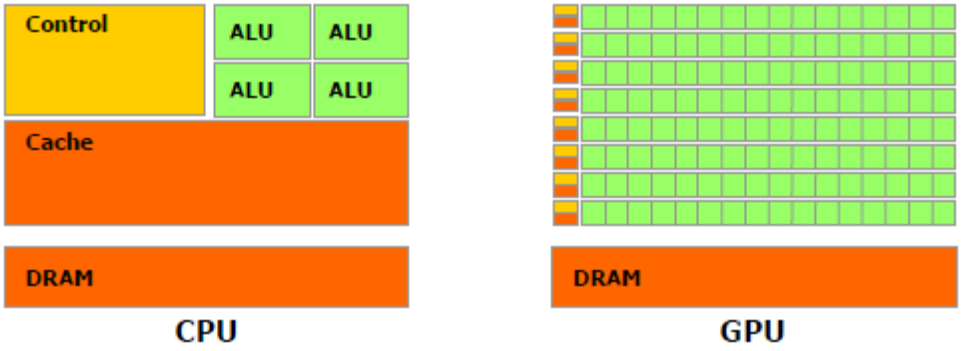


Figure 62. GPU uses more transistors for data processing.

The kernels, C functions, are executed as a grid of thread blocks. A thread block is a batch of threads that can cooperate with one another by sharing data through shared memory or synchronizing their execution. The threads from different blocks cannot cooperate. Figure 63 shows a kernel in the host (CPU side), which is implemented in the device (GPU side). Note that a grid is a set of blocks, and a block is a set of threads.

Recently, GPUs can handle more than 512 independent thread processors, whereas CPUs at the desktop level have only reached eight cores.

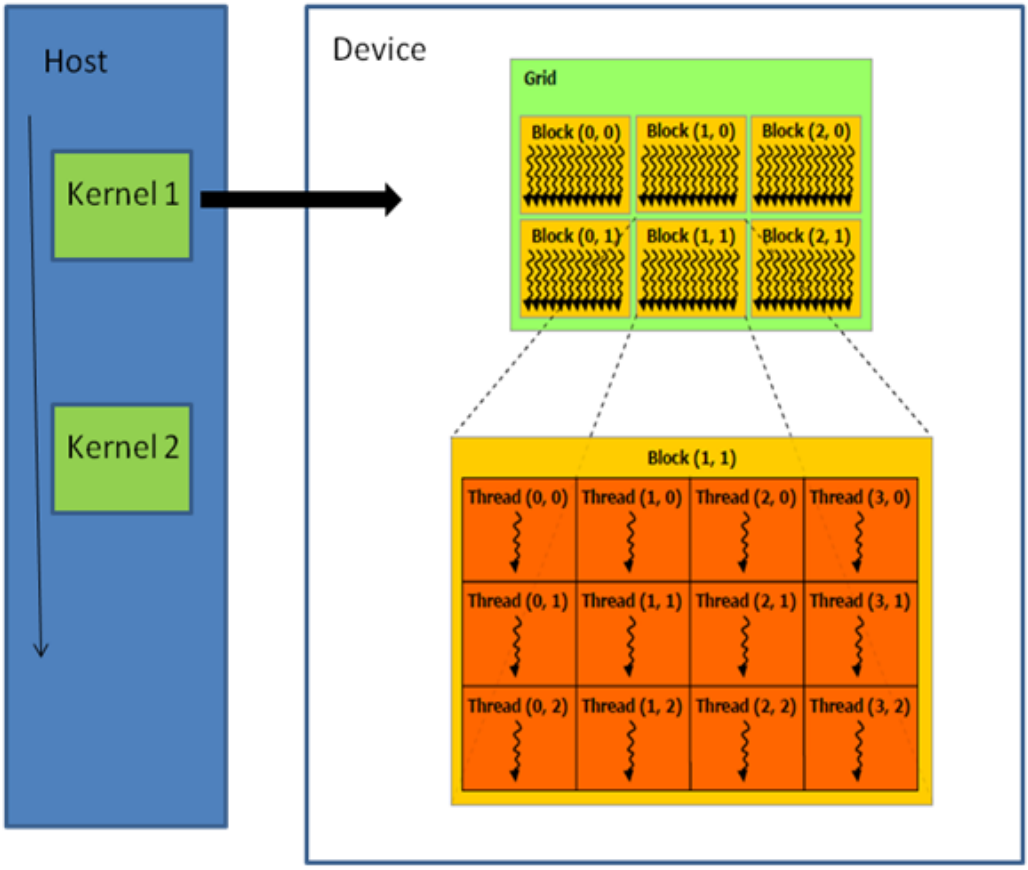


Figure 63. Programming model over GPU.

6.3 GPU-CPU Implementation for Video Mosaicking of UAS Surveillance Video

In this section, the construction of dynamic video mosaicking using GPU-CPU is presented. The GPU is used to compute the homography for registration, and the CPU is allocated the remaining jobs: read the video data and reprojection.

Figure 64. Real time video mosaicking constructed using GPU-CPU. The video belongs to MATLAB/Simulink demonstrations.

Table 21 shows the time comparison between GPU and CPU for the computation of the homography. The GPU is almost 55 times faster than the CPU for this process. This represents a good speed up that can allow us to achieve our goal of real-time video mosaicking.

Table 22 details the algorithm to construct the video mosaic using GPU-CPU; also Figure 64 shows an example of video mosaicking constructed in real time. This video mosaic was taken from a video demonstration in MATLAB/ Simulink.

Table 21. Comparison of the computational time of the homography.

Type of Test	**Time GPU (ms)**	**Time CPU (ms)**	**Time CPU/ Time GPU**
Find the Homography for UAS IR video	20.8580	1141.0	54.7

Table 22. Algorithm to construct a video mosaic given an input video over GPU – CPU.

Objective: Construct a video mosaic from a video input using GPU-CPU.

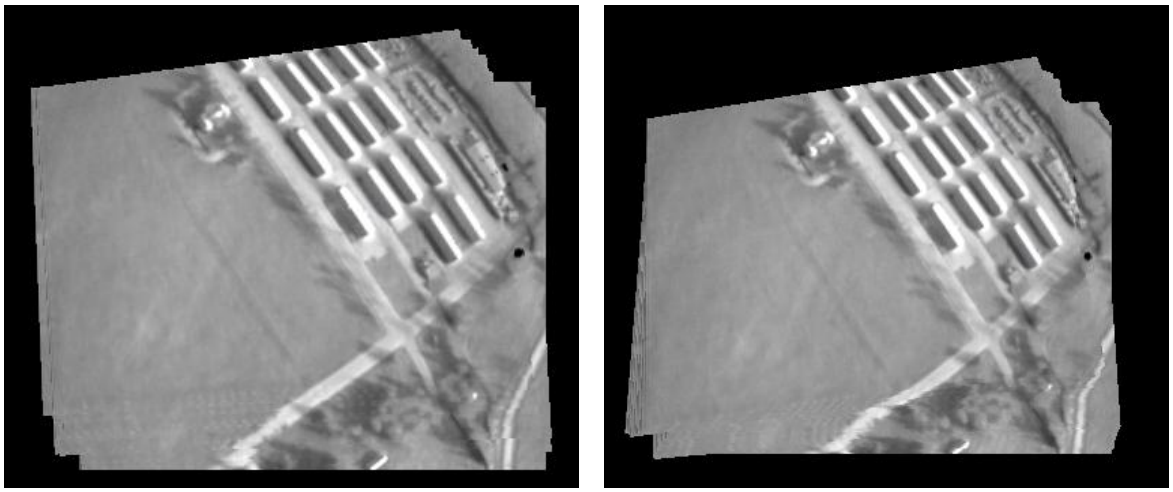
Algorithm:

1. Read the information from the video file (duration, frames per second, size of the frames, codec type, etc.).
2. Read the video frame-by-frame and do the following:
    a) Select the I-frames (for MPEG video).
    b) Select the first frame as a reference frame.
    c) Copy the frame from the host (CPU) memory to device (GPU) memory.
    d) Compute the SIFT features for every frame and save them into a GPU memory.
    e) Compute the homography for two consecutive frames.
    f) Copy the homography matrix from device memory to host memory.
    g) Reproject the frame into a common coordinate system.

Table 23 shows a computational time comparison for different tests. Here, the time considers the whole process to construct the video mosaic according to Table 21. It is important to note that the performance of constructing the video mosaic using MPEG I-frames generates not only better results (see Figure 65), but is also faster than using all of the frames.

Table 23. Comparison of the computational time for complete video mosaicking.

Type of Test	Condition	Time GPU – CPU (sec)	Time CPU (sec)	Time CPU/ Time GPU-CPU
Video Mosaicking of the demo video of MATLAB	Reading at 15 fps	9.297	107.125	11.52
Video Mosaicking of the UAS IR video	Reading at 25 fps	17.938	135.953	7.58
Video Mosaicking of the UAS IR video	Using MPEG-I frames	4.234	14.468	3.41



(a) (b)

Figure 65. Video mosaicking results for 5.120 seconds at 25 frames per second for the IR video captured in 2007 by the UASE Laboratory team at the University of North Dakota. Left: Mosaic result using only MPEG I-frames. Right: Result using all frames from 5.120 seconds of video.

6.4 GPU-CPU Implementation for Super-Resolution Mosaicking of UAS Surveillance Video

This section shows the results of using GPU-CPU implementation for the computation of super-resolution mosaics using the three different optimization

algorithms: steepest descent, conjugate gradient, and Levenberg Marquardt. The tests performed use the same data as in Chapter 5.

6.4.1 GPU-CPU Implementation for Super-Resolution Mosaicking Using Steepest Descent

This section presents the results for the computation of super-resolution mosaicking using the steepest descent algorithm over GPU-CPU. Table 24 details the algorithm used; basically, this is almost the same as the algorithm in Chapter 5, with the difference being that the registration step is performed over the GPU, increasing the speed up by more than 50 times (see Table 21).

Figure 66 shows the results for the first and second set of synthetic images. As expected, the results are similar to those obtained using only CPU, but the PSNR is a little bit lower than using only CPU (Table 25); the time difference is about 1.4 to 1.7 seconds faster, since the registration is done using GPU, with remaining task taken over by the CPU (Figure 44).

Figure 67 shows the LR and SR mosaics for a set of five color frames captured by a UAS. This SR mosaic has less quality than that obtained using only CPU (Figure 46). The reason for this is that GPU registration had some issues with this set of images, especially with the second and third frames. The result was that the homography between them was not accurate enough.

Figure 68 shows the LR and SR mosaics for two different sets of IR frames captured in 2007 by the UASE Laboratory team at the University of North Dakota. The results are similar to those obtained using only CPU (Figure 47).

Table 24. Algorithm to construct a super-resolved mosaic using steepest descent algorithm over GPU-CPU given a set of *N* input frames.

Objective: Construct a super-resolved mosaic using the steepest descent algorithm over GPU-CPU from a set of *N* input frames.

Algorithm:

1. Compute the homography on the GPU side, using Table 1 between consecutive frames, i.e., $h_{12}, h_{23}, ..., h_{N-1,N}$
2. Copy the homography from the device memory (GPU) to the host memory (CPU).
3. Reproject all the input frames to a common coordinate system using equation (4.6). The result of this step becomes the initial condition $\hat{\mathbf{x}}^{(0)}$ for the SR mosaic (iteration 0).
4. While iteration *n* is less than the maximum number of iterations:
    a) Construct a set of *N* reconstructed frames based on $\hat{\mathbf{x}}^{(n)}$.
    b) Construct a set of *N* difference frames but subtracting the input frames from the set of frames in a).
    c) Construct a mosaic using Table 1 with the set of difference frames.
    d) Construct the regularization factor given by $\lambda^{(n)}\mathbf{G}^T\rho'(G\hat{\mathbf{x}}^{(n)}, \alpha)$.
    e) Subtract the result of c) from d) and then add to the previous SR mosaic $\hat{\mathbf{x}}^{(n-1)}$.

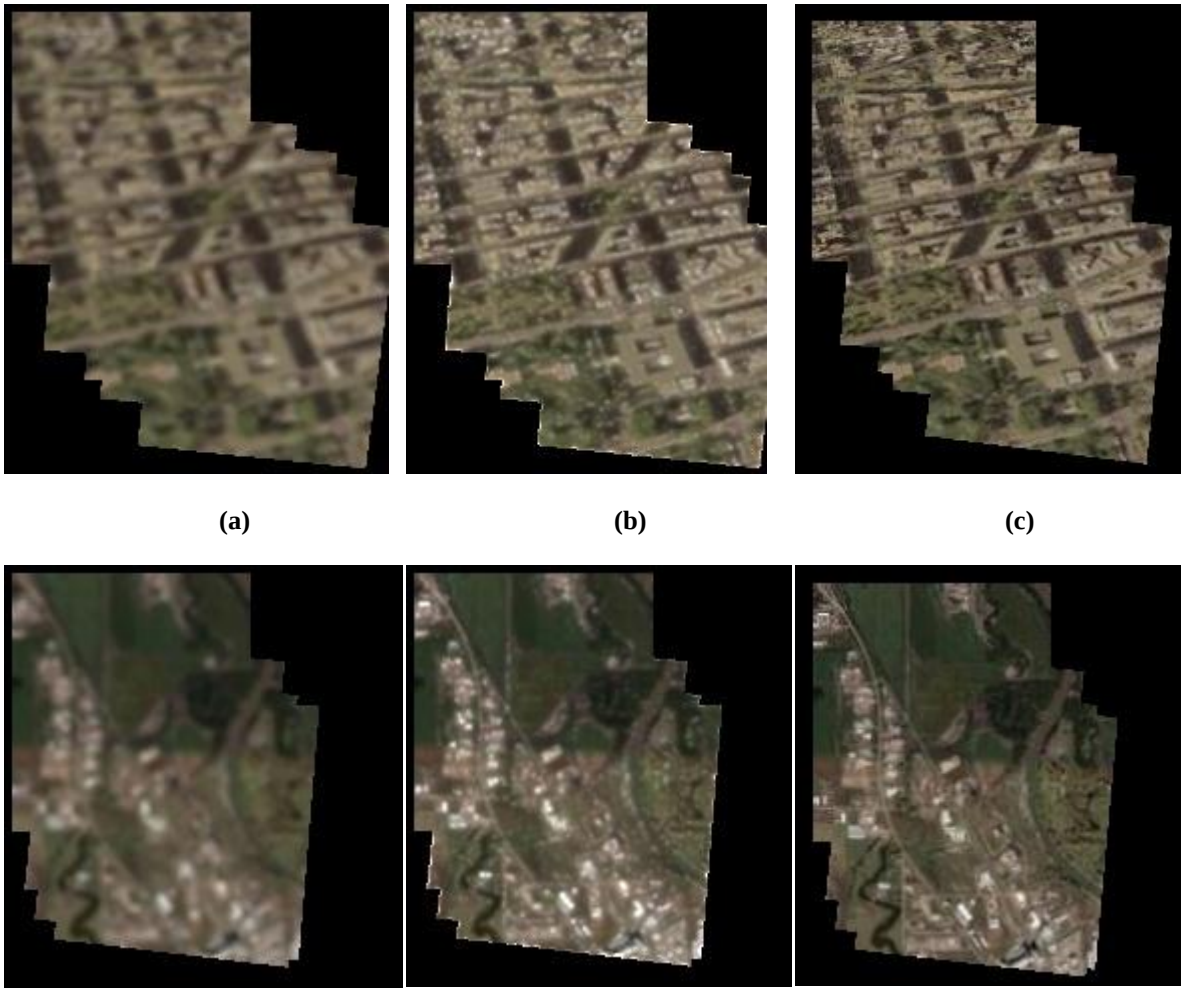


Figure 66. Results of the SR mosaicking for synthetic frames the using steepest descent algorithm implemented over GPU-CPU. The mosaics were constructed using five frames. Figures (a) and (d) show the mosaics for the first and second set of synthetic frames, respectively. These mosaics are the input to the algorithm. (b) and (e) are the super-resolved mosaics of (a) and (d), respectively. These mosaics are the output of the proposed algorithm. Figures (c) and (f) show the ground truth mosaics, constructed using high-resolution frames.

Table 26 shows the results obtained with the three different sets of real UAS frames. Additionally, information for the results using only CPU is included. The advantage of GPU-CPU over CPU is clear in this table, and the speed up is significant.

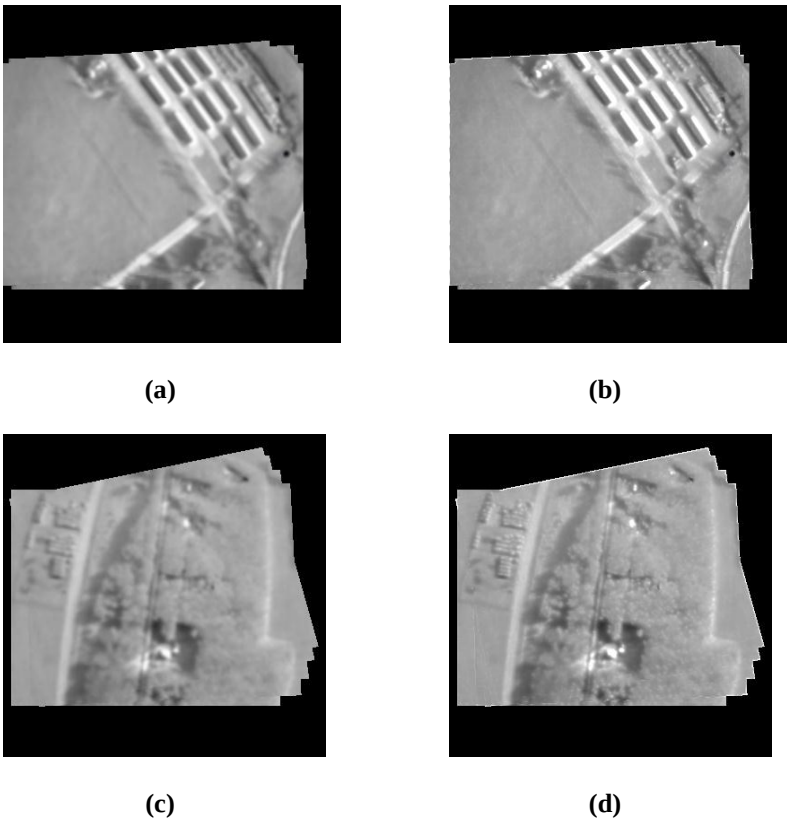
Table 25. Results of computating of super-resolution mosaics using steepest descent over GPU-CPU for two different sets of color synthetic frames.

Test	PSNR (dB) (GPU-CPU)	PSNR (dB) (CPU)	Final error (GPU-CPU) $\left\|\hat{\mathbf{x}}_{k+1}-\hat{\mathbf{x}}_k\right\|_2 \Big/ \left\|\mathbf{x}_k\right\|_2$	Final error (CPU) $\left\|\hat{\mathbf{x}}_{k+1}-\hat{\mathbf{x}}_k\right\|_2 \Big/ \left\|\mathbf{x}_k\right\|_2$	Total Processing Time on GPU-CPU (sec)	Total Processing Time on CPU (sec)
First set of five synthetic color frames.	35.16	43.86	0.002608	0.006391	3.234	4.625
Second set of five synthetic color frames	46.41	47.52	0.002727	0.0029404	3.156	3.875

Figure 67. Result of super-resolution mosaicking using steepest descent implemented over GPU-CPU. Left: LR mosaic. Right: SR mosaic. The mosaics were constructed using five frames of size 320x240x3 pixels.

Table 26. Results of computing of super-resolution mosaics using steepest descent over GPU-CPU for three different sets of real frames captured by UAS.

Test	Final error (GPU-CPU) $\left\|\hat{\mathbf{x}}_{k+1}-\hat{\mathbf{x}}_k\right\|_2 \Big/ \left\|\mathbf{x}_k\right\|_2$	Final error (CPU) $\left\|\hat{\mathbf{x}}_{k+1}-\hat{\mathbf{x}}_k\right\|_2 \Big/ \left\|\mathbf{x}_k\right\|_2$	Total Processing Time on GPU-CPU (sec)	Total Processing Time on CPU (sec)
Test #1: color frames of a road and forest.	0.003089	0.005055	8.594	16.891
Test #2: IR frames of buildings.	0.001849	0.003379	8.547	11.907
Test #3: IR frames of a forest.	0.001036	0.003655	8.469	11.219



(a) **(b)**

(c) **(d)**

Figure 68. Results of SR mosaicking for real frames from UAS using steepest descent implemented over GPU-CPU. The mosaics were constructed using five frames. Figures (a) and (c) show the mosaics for the first and second sets of frames, respectively. These mosaics are the input to the algorithm. (b) and (d) are the super-resolved mosaics of (a) and (c) respectively. These mosaics are the output of the steepest descent super-resolution mosaicking algorithm.

6.4.2 GPU-CPU Implementation for Super-Resolution Mosaicking Using Conjugate Gradient

This section presents the results for the computation of super-resolution mosaics using the conjugate gradient algorithm over GPU-CPU. Table 27 details the algorithm used, essentially the same as the algorithm in Chapter 5 with the difference being that the registration step is performed over GPU.

Figure 69 shows the results for the first and second sets of synthetic images. The results are similar to those obtained using only CPU (Figure 51), but the PSNR is a little bit lower than using only CPU (Table 28); the time difference is about 0.9 to 1.2 seconds faster.

Figure 70 shows the LR and SR mosaics for a set of five color frames captured by a UAS. This SR mosaic has slightly less quality than obtained using only CPU (Figure 52).

Figure 71 shows the LR and SR mosaics for two different sets of IR frames captured in 2007 by the UASE Laboratory team at the University of North Dakota. The results shown are similar to those obtained using only CPU (Figure 53).

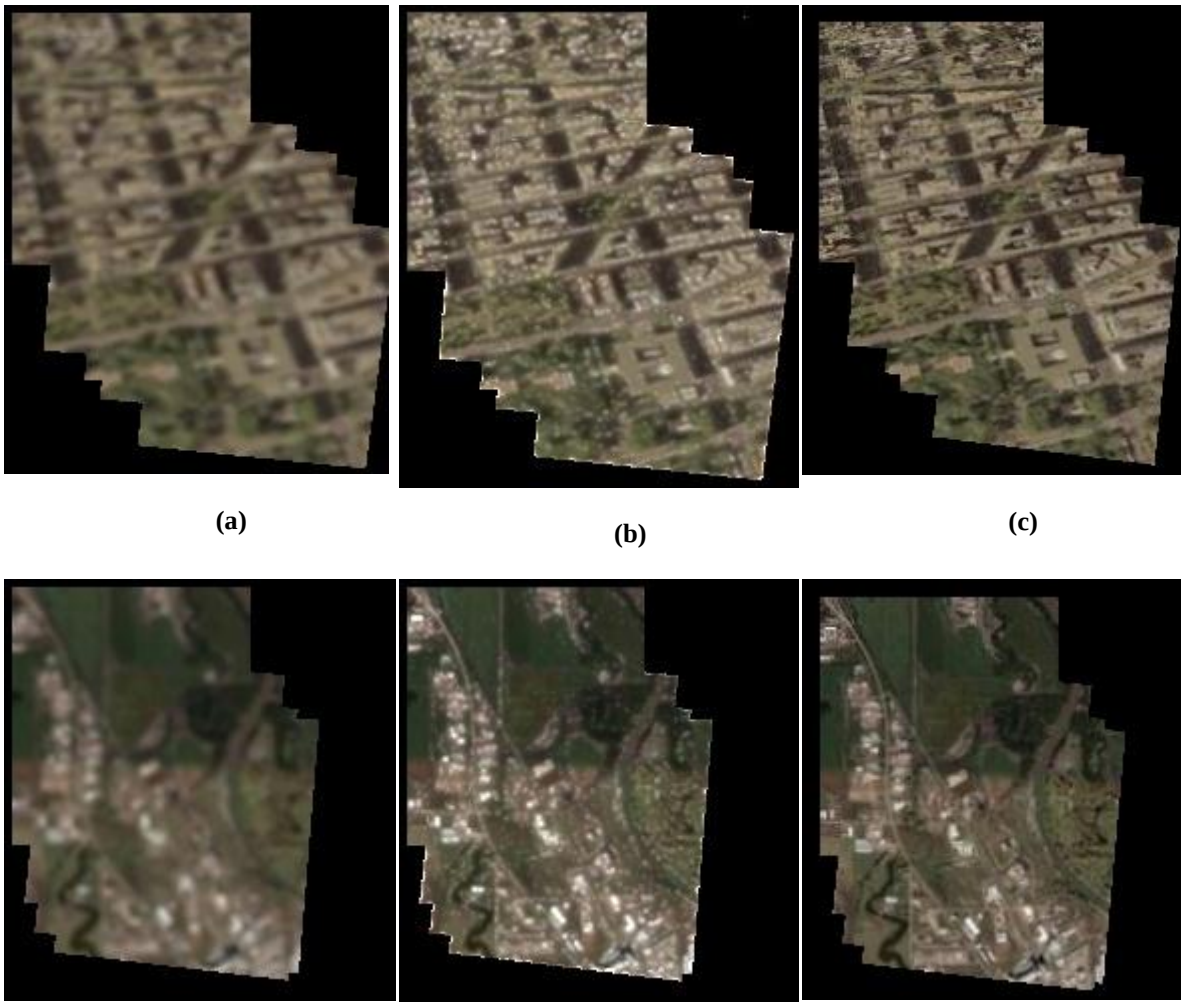
Table 29 shows the results obtained with the three different sets of real UAS frames, along with information for the results using only CPU. The advantage of GPU-CPU over CPU is clear in this table, and the speed up is significant.

Table 27. Algorithm to construct a super-resolved mosaic using conjugate gradient over GPU-CPU given a set of *N* input frames.

Objective: Construct a super-resolved mosaic using the conjugate gradient optimization algorithm over GPU-CPU from a set of *N* input frames.

Algorithm:

1. Compute the homography on the GPU side using Table 1 between consecutive frames; i.e., $h_{12}, h_{23}, ..., h_{N-1,N}$.
2. Copy the homography from the GPU (device memory) to the CPU (host memory).
3. Reproject all the input frames to a common coordinate system using equation (4.6). The result of this step becomes the initial condition for the $\hat{\mathbf{x}}^{(0)}$ SR mosaic (iteration 0), and it is used to compute the gradient for the iteration 0 using (5.16).
4. While iteration *n* is less than the maximum number of iterations:
    a) Construct the gradient using equation (5.16).
    b) Construct $\mathbf{p}^{(n)}$ using equation (5.15).
    c) Update $\hat{\mathbf{x}}^{(n)}$ using equation (5.13).



(a) **(b)** **(c)**

(d) **(e)** **(f)**

Figure 69. Results of SR mosaicking for synthetic frames using the conjugate gradient algorithm implemented over GPU-CPU. The mosaics were constructed using five frames. Figures (a) and (d) show the mosaics for the first and second sets of synthetic frames, respectively. These mosaics are the input to the algorithm. (b) and (e) are the super-resolved mosaics using the CG algorithm over GPU-CPU of (a) and (d), respectively. These mosaics are the output of the CG algorithm. Figures (c) and (f) show the ground truth mosaics, constructed using high-resolution frames.

Table 28. Results of computating super-resolution mosaics using the conjugate gradient algorithm over GPU-CPU for two different sets of color synthetic frames.

Test	PSNR (dB) (GPU-CPU)	PSNR (dB) (CPU)	Final error (GPU-CPU) $\left\|\hat{\mathbf{x}}_{k+1}-\hat{\mathbf{x}}_k\right\|_2 \Big/ \left\|\mathbf{x}_k\right\|_2$	Final error (CPU) $\left\|\hat{\mathbf{x}}_{k+1}-\hat{\mathbf{x}}_k\right\|_2 \Big/ \left\|\mathbf{x}_k\right\|_2$	Total Processing Time on GPU-CPU (sec)	Total Processing Time on CPU (sec)
First set of five synthetic color frames.	35.17	43.98	0.006790	0.004381	3.218	5.047
Second set of five synthetic color frames	46.59	47.54	0.008713	0.006212	3.187	4.250

Figure 70. Results of super-resolution mosaicking using the conjugate gradient algorithm implemented over GPU-CPU. Left: LR mosaic. Right: SR mosaic. The mosaics was constructed using five frames of size 320x240x3 pixels.

Table 29. Results of computating of super-resolution mosaics using the conjugate gradient algorithm over GPU-CPU for three different sets of real frames captured by UAS.

Test	Final error (GPU-CPU) $\left\|\hat{\mathbf{x}}_{k+1}-\hat{\mathbf{x}}_k\right\|_2 \Big/ \left\|\mathbf{x}_k\right\|_2$	Final error (CPU) $\left\|\hat{\mathbf{x}}_{k+1}-\hat{\mathbf{x}}_k\right\|_2 \Big/ \left\|\mathbf{x}_k\right\|_2$	Total Processing Time on GPU-CPU (sec)	Total Processing Time on CPU (sec)
Test #1: Color frames of a road and forest.	0.009512	0.005055	9.093	16.891
Test #2: IR frames of buildings.	0.005004	0.003379	8.797	11.907
Test #3: IR frames of a forest.	0.004392	0.003655	8.719	11.219

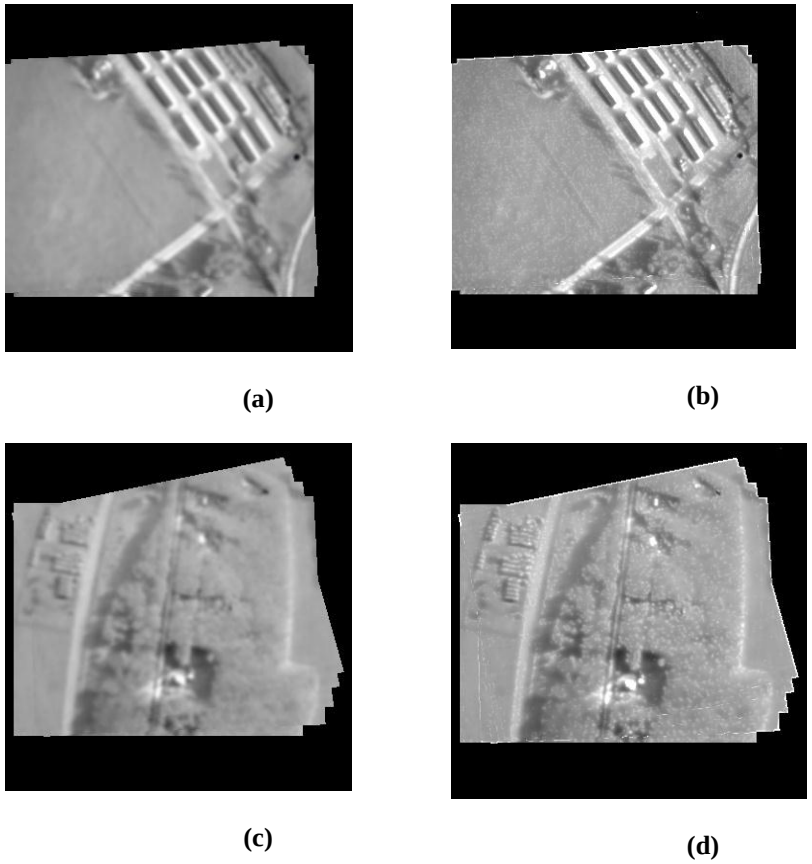


Figure 71. Results of SR mosaic for real frames from UAS using the conjugate gradient algorithm implemented over GPU-CPU. The mosaics were constructed using five frames. Figures (a) and (c) show the mosaics for the first and second sets of frames, respectively. These mosaics are the input to the algorithm. (b) and (d) are the super-resolved mosaics of (a) and (c), respectively. These mosaics are the output of the proposed conjugate gradient super-resolution mosaicking algorithm.

6.4.3 GPU-CPU Implementation for Super-Resolution Mosaicking Using the Levenberg-Marquardt

This section presents the results for the computation of super-resolution mosaics using the Levenberg Marquardt algorithm over GPU-CPU. Table 30 details the algorithm used. Essentially, it is the same as the algorithm in Chapter 5, with the difference being that the registration step is performed over the GPU.

Figure 72 shows the results for the first and second sets of synthetic images. The results are similar to those obtained using only CPU (Figure 54), but the PSNR is a little bit lower than using only CPU (Table 31); the difference in computation time is about 0.8 to 0.9 seconds.

Figure 73 shows the LR and SR mosaics for sets of five color frames captured by a UAS. These SR mosaic has less quality than those obtained using only CPU (Figure 55).

Figure 74 shows the LR and SR mosaics for two different sets of IR frames captured in 2007 by the UASE Laboratory team at the University of North Dakota. The results shown are similar to those obtained using only CPU (Figure 56).

Table 32 shows the results obtained with the three different sets of real UAS frames, along with information of the results using only CPU. The advantage of GPU-CPU over CPU is clear in this table, and the speed up is significant.

Table 30. Algorithm to construct super-resolved mosaics using the Levenberg Marquardt algorithm over GPU-CPU given a set of *N* input frames.

Objective: Construct a super-resolved mosaic using the Levenberg Marquardt algorithm over GPU-CPU from a set of *N* input frames.

Algorithm:

1. Compute the homography on the GPU side using Table 1 between consecutive frames; i.e., $h_{12}, h_{23}, ..., h_{N-1,N}$.
2. Copy the homography from the GPU (device memory) to the CPU (host memory).
3. Reproject all the input frames to a common coordinate system using equation (4.6). The result of this step becomes the initial condition for the $\hat{\mathbf{x}}^{(0)}$ SR mosaic (iteration 0).
4. While iteration *n* is less than the maximum number of iterations:
    a) Construct the Jacobian using equation (5.19).
    b) Construct the pseudo Hessian matrix given by $\mathbf{H}^* = J^T J$.
    c) Solve the linear least squares equation using the singular value decomposition (SVD) for $\delta\mathbf{x}$., in equation (5.21)
    d) Update $\hat{\mathbf{x}}^{(n)}$ in equation (5.24).

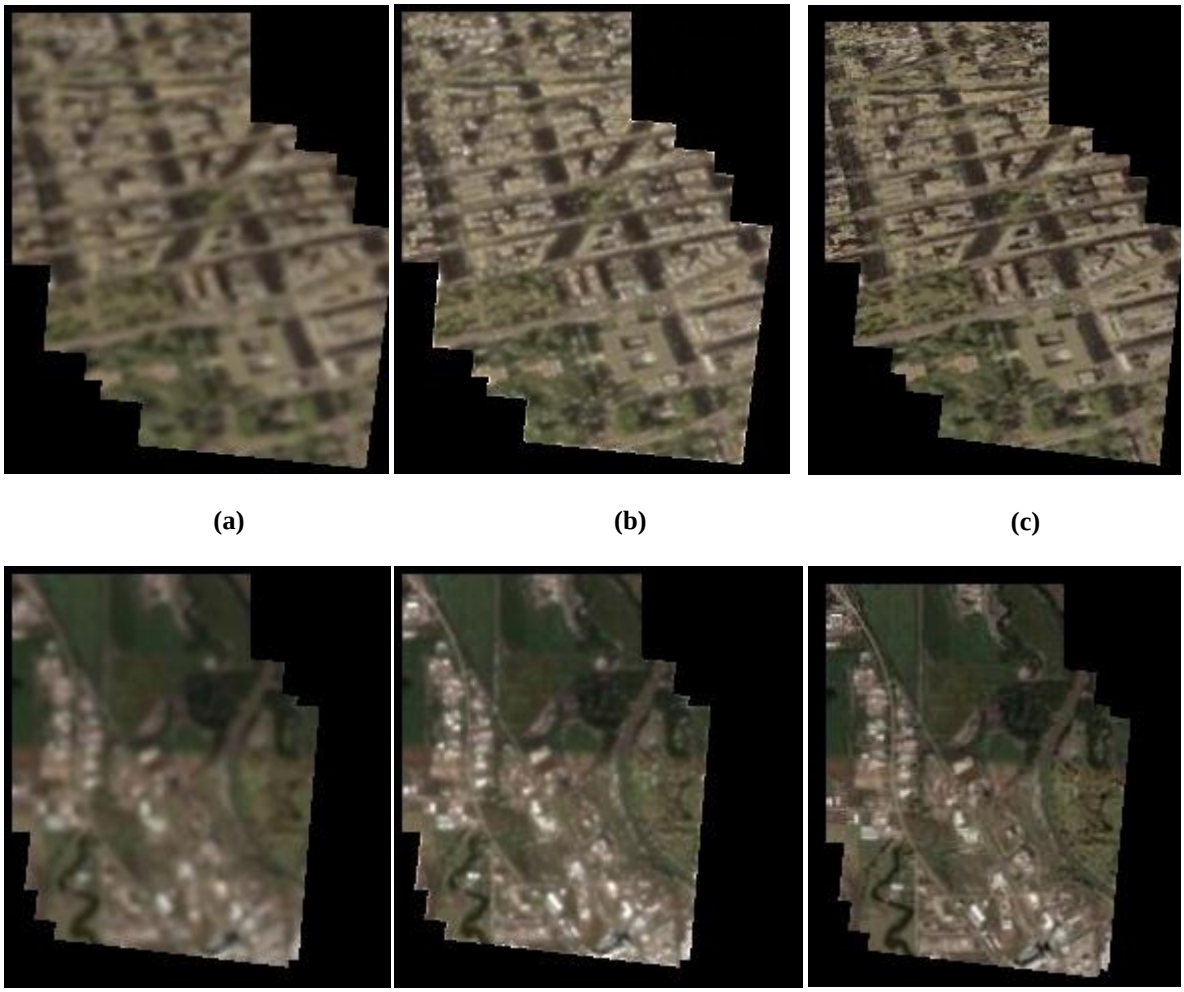


Figure 72. Results of SR mosaicking for synthetic frames using the Levenberg Marquardt algorithm implemented over GPU-CPU. The mosaics were constructed using five frames. Figures (a) and (d) show the mosaics for the first and second set of synthetic frames, respectively. These mosaics are the input to the algorithm. (b) and (e) are the super-resolved mosaics using the LM algorithm of (a) and (d), respectively. These mosaics are the output of the proposed algorithm. Figures (c) and (f) show the ground truth mosaics, constructed using high-resolution frames.

Table 31. Results of the computating of super-resolution mosaicks using the Levenberg Marquardt algorithm over GPU-CPU for two different set of color synthetic frames.

Test	PSNR (dB) (GPU-CPU)	PSNR (dB) (CPU)	Final error (GPU-CPU) $\left\|\hat{\mathbf{x}}_{k+1}-\hat{\mathbf{x}}_k\right\|_2 \Big/ \left\|\mathbf{x}_k\right\|_2$	Final error (CPU) $\left\|\hat{\mathbf{x}}_{k+1}-\hat{\mathbf{x}}_k\right\|_2 \Big/ \left\|\mathbf{x}_k\right\|_2$	Total Processing Time on GPU-CPU (sec)	Total Processing Time on CPU (sec)
First set of five synthetic color frames.	35.14	43.77	0.006790	0.002833	3.359	5.109
Second set of five synthetic color frames	46.49	47.45	0.008713	0.002505	3.391	4.281

Figure 73. Results of super-resolution mosaicking using the Levenberg Marquardt algorithm implemented over GPU-CPU. Left: LR mosaic. Right: SR mosaic. The mosaics were constructed using five frames of size 320x240x3 pixels.

Table 32. Results of computing of super-resolution mosaics using the Levenberg Marquardt algorithm over GPU-CPU for three different sets of real frames captured by UAS.

Test	Final error (GPU-CPU) $\left\|\hat{\mathbf{x}}_{k+1}-\hat{\mathbf{x}}_k\right\|_2 \Big/ \left\|\mathbf{x}_k\right\|_2$	Final error (CPU) $\left\|\hat{\mathbf{x}}_{k+1}-\hat{\mathbf{x}}_k\right\|_2 \Big/ \left\|\mathbf{x}_k\right\|_2$	Total Processing Time on GPU-CPU (sec)	Total Processing Time on CPU (sec)
Test #1: color frames of a road and forest.	0.009512	0.016835	10.547	17.485
Test #2: IR frames of buildings	0.005004	0.004181	9.328	11.766
Test #3: IR frames of a forest.	0.004392	0.005295	9.266	11.391

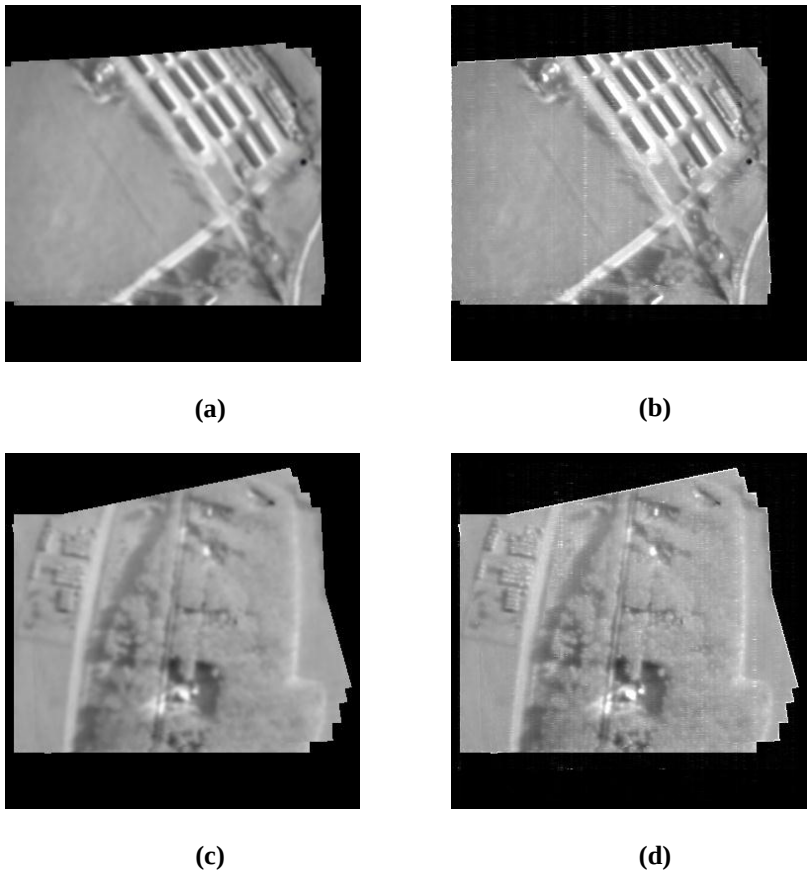


Figure 74. Results of SR mosaicking for real frames from UAS using the Levenberg Marquardt algorithm implemented over GPU-CPU. The mosaics were constructed using five frames. Figures (a) and (c) show the mosaics for the first and second sets of frames, respectively. These mosaics are the input to the algorithm. (b) and (d) are the super-resolved mosaics of (a) and (c), respectively. These mosaics are the output of the proposed Levenberg Marquardt super-resolution mosaicking algorithm.

6.5 Conclusions

The use of GPU for a highly computation step in the super-resolution mosaicking process decreases the computational time significantly. The results for most of the tests performed are quite similar to those obtained using CPU. Conversely, for the first set of synthetic images, the PSNR of the super-resolved mosaics using only CPU are much better than using GPU-CPU for all the three algorithms: steepest descent, conjugate gradient, and Levenberg Marquardt.

The use of GPU for the computation of the homography is at least 50 times faster than using CPU. The computation time in the construction of the super-resolved mosaic can be reduced even more, if more functions are moved from CPU to GPU.

The use of MPEG I-frames reduces the accumulation of error, which is inherent to the projection model, in the computation of the video mosaic.

CHAPTER 7

CONCLUSIONS AND DIRECTIONS FOR FUTURE RESEARCH

This dissertation investigated super-resolution mosaicking of video frames captured by UAS. Additionally, a focus on reducing the computational time using GPU was explained. Since most of the tests were performed with real data and with actual size frames captured by modern video cameras, this research can be applied almost immediately applied to actual commercial or military surveillance data. A summary of the contributions and their importance to image and video processing will be discussed in the next sections.

7.1 Summary of Research Contributions

The background information presented in Chapter 2, combined with a brief tutorial for the regularization methods presented in Chapter 3, provided a complete and accessible introduction to the computation of video mosaic and super-resolution mosaics. Additionally, Chapter 2 provided a comprehensive survey of the literature on multi-frame super-resolution reconstruction and mosaicking.

Chapter 4 explained the reason why SIFT was chosen to be the feature detector. Also, Chapter 4 explained how to compute the computation of the homography with sub-pixel accuracy. This accuracy is important for both super-resolution and mosaicking. Furthermore, a detailed explanation of the construction of video and image mosaicking was given. The use of MPEG I-frames instead of all frames in the construction of video

mosaics was also explained and how this can help to reduce distortion was shown in Chapter 6.

Chapter 5 used a novel framework to compute super-resolution mosaics. This framework is based on the iterative back projection method [24], and image operators including convolution, warping, and down-sampling. Also, this framework adds a regularization term based on cliques over the estimated super-resolution mosaic, using the Huber function to incorporate prior information. The computation of the Lagrange multiplier was performed using cliques and the Huber function, but without the construction of sparse matrices. Also, this computation of the Lagrange multiplier is robust to mis-registration that always occurs when dealing with real UAS surveillance video frames. Three different optimization algorithms were used to find the super-resolution mosaics: 1) steepest descent, 2) conjugate gradient, and 3) Levenberg Marquardt. This last method is a novel algorithm proposed in this dissertation for super-resolution mosaicking. The results showed a great improvement in the resolution in both color and gray pixel levels.

Chapter 6 presented the use of GPU in the computation of super-resolution mosaics. Basically, GPU is used to solve the bottleneck in the entire process of the construction of super-resolution mosaics, which is the image registration step. The results showed that the GPU is more than 50 times faster than the CPU. Then, the CPU takes the results of the registration, called the homography, to continue with the reprojection and the solution to the optimization problem using the three algorithms of Chapter 5. A comparison of the results of video mosaicking and super-resolution mosaicking using GPU-CPU and the CPU alone were provided. Here, the GPU-CPU implementation

reduces significantly the computational time and the visual results are slightly inferior compared to using only CPU.

7.2 Future Research

A number of potential research avenues exist related to the topics addressed in this dissertation for improvements to the presented research:

- The parameters $\alpha^{(n)}$, for the steepest descent algorithm, and the parameter $\beta^{(n)}$, for the conjugate gradient algorithm , were set to a fixed values in every iteration. These parameters can be computed iteratively to create a more adaptive algorithm.
- Different prior information can be used, such as Gaussian, Lorentzian, or others that introduce good prior information for the computation of super-resolution mosaics.
- The framework used allows for the implementation of any optimization algorithm with no need for using sparse matrices. The following algorithms can be also used with the framework proposed in this dissertation: Gauss-Newton, Quasi Newton's, and Davidson Fletcher-Powell.
- The PSF was assumed known a priori, but it can be estimated using "blind" deconvolution as was used in [45,46].
- Geo-referenced super-resolution mosaicking can be performed based on the information detailed in Chapters 4 and 5. The use of the particle filter can also be used to estimate the pose of the mosaic. Geo-referenced mosaics have a number of applications. They can be used for tracking recognition, for target detection, and for mapping a certain area in Google Earth. The advantage over Google Earth is that it will increase the resolution of the data.

- Bundle adjustment can be used to correct the error propagation in the construction of the video mosaics. This bundle adjustment can also be tied to the construction of the geo-referenced mosaic.
- Speed up can be increased with the use of more GPUs in parallel and also if more of the functions are implemented on the GPU side. Therefore, real-time super-resolution mosaicking of actual UAS video frames can be achieved with today's hardware.

APPENDICES

Appendix A

Use of MPEG I-frames for Video Processing Using FFMPEG

```
/* --Sparse Optical Flow Demo Program--
 * Written by David Stavens (david.stavens@ai.stanford.edu)
 * Modified by Aldo Camargo (aldo.camargo@und.edu) to work with only I frames ( using ffmpeg library ) and generates
 * a video output in avi, the input could be any video format ( I tested with AVI and MPEG formats )
 */
#include <stdio.h>
#include <cv.h>
#include <highgui.h>
#include <math.h>
#include <ffmpeg/avcodec.h>  //add to work with ffmpeg
#include <ffmpeg/avformat.h> //add to work with ffmpeg

static const double pi = 3.14159265358979323846;

inline static double square(int a)
{
	return a * a;
}

/* This is just an inline that allocates images.  I did this to reduce clutter in the
 * actual computer vision algorithmic code.  Basically it allocates the requested image
 * unless that image is already non-NULL.  It always leaves a non-NULL image as-is even
 * if that image's size, depth, and/or channels are different than the request.
 */
inline static void allocateOnDemand( IplImage **img, CvSize size, int depth, int channels )
{
	if ( *img != NULL )	return;

	*img = cvCreateImage( size, depth, channels );
	if ( *img == NULL )
	{
		fprintf(stderr, "Error: Couldn't allocate image.  Out of memory?\n");
		exit(-1);
	}
}

void SaveFrame(AVFrame *pFrame, int width, int height, int iFrame) {
  FILE *pFile;
  char szFilename[32];
  int  y;

  // Open file
```

```
  sprintf(szFilename, "frame%d.ppm", iFrame);
  pFile=fopen(szFilename, "wb");
  if(pFile==NULL)
    return;

  // Write header
  fprintf(pFile, "P6\n%d %d\n255\n", width, height);

  // Write pixel data
  for(y=0; y<height; y++)
    fwrite(pFrame->data[0]+y*pFrame->linesize[0], 1, width*3, pFile);

  // Close file
  fclose(pFile);
}
int main(int argc, char *argv[])
{

    AVFormatContext *pFormatCtx;
    AVCodecContext  *pCodecCtx;
    AVCodec         *pCodec;
    AVFrame         *pFrame;
    AVFrame         *pFrameRGB;
    int             numBytes;
    uint8_t         *buffer;
    int             i, videoStream;
    AVPacket        packet;
    int             frameFinished;
    int j;
    float framerate;

/////
/////  This is the first part of the OpenCV Initialization
/////


          if (argc != 2)
          {
                fprintf(stderr, "usage: %s input.avi\n", argv[0]);
                return -1;
          }
          /* Step 1: Open Input Video */
          /* Create an object that decodes the input video stream. */
          CvCapture *input_video = cvCaptureFromFile( argv[1] );
          if (input_video == NULL)
          {
```

```
                /* Either the video didn't exist OR it uses a codec OpenCV
                 * doesn't support.
                 */
                fprintf(stderr, "Error: Can't open video.\n");
                return -1;
        }

  ////
  ////   Initialization of ffmpeg
  ////

     // Register all formats and codecs
      av_register_all();

// Open video file
  if(av_open_input_file(&pFormatCtx, argv[1], NULL, 0, NULL)!=0)
    return -1; // Couldn't open file

  // Retrieve stream information
  if(av_find_stream_info(pFormatCtx)<0)
    return -1; // Couldn't find stream information

  // Dump information about file onto standard error
  dump_format(pFormatCtx, 0, argv[1], 0);

  // Find the first video stream
  videoStream=-1;
  for(i=0; i<pFormatCtx->nb_streams; i++)
    if(pFormatCtx->streams[i]->codec->codec_type==CODEC_TYPE_VIDEO) {
      videoStream=i;
      break;
    }
  if(videoStream==-1)
    return -1; // Didn't find a video stream

  // Get a pointer to the codec context for the video stream
  pCodecCtx=pFormatCtx->streams[videoStream]->codec;

  // Find the decoder for the video stream
  pCodec=avcodec_find_decoder(pCodecCtx->codec_id);
  if(pCodec==NULL) {
    fprintf(stderr, "Unsupported codec!\n");
    return -1; // Codec not found
  }
  // Open codec
  if(avcodec_open(pCodecCtx, pCodec)<0)
```

```
  return -1; // Could not open codec

 // Allocate video frame
 pFrame=avcodec_alloc_frame();

 // Allocate an AVFrame structure
 pFrameRGB=avcodec_alloc_frame();
 if(pFrameRGB==NULL)
  return -1;

 // Determine required buffer size and allocate buffer
 numBytes=avpicture_get_size(PIX_FMT_RGB24, pCodecCtx->width,
                             pCodecCtx->height);
 buffer=(uint8_t *)av_malloc(numBytes*sizeof(uint8_t));

 // Assign appropriate parts of buffer to image planes in pFrameRGB
 // Note that pFrameRGB is an AVFrame, but AVFrame is a superset
 // of AVPicture
 avpicture_fill((AVPicture *)pFrameRGB, buffer, PIX_FMT_RGB24,
                pCodecCtx->width, pCodecCtx->height);


////
////  End of ffmpeg initialization
////

/////
/////  OpenCV Initialization
/////
        cvQueryFrame( input_video );

        /* Step 2: Read AVI Properties */
        /* Read the video's frame size out of the AVI. */
        CvSize frame_size;
        frame_size.height =
                (int) cvGetCaptureProperty( input_video,
CV_CAP_PROP_FRAME_HEIGHT );
        frame_size.width =
                (int) cvGetCaptureProperty( input_video,
CV_CAP_PROP_FRAME_WIDTH );
        framerate = (float)cvGetCaptureProperty( input_video, CV_CAP_PROP_FPS );

        /* Determine the number of frames in the AVI. */
        long number_of_frames;
        /* Go to the end of the AVI (ie: the fraction is "1") */
        cvSetCaptureProperty( input_video, CV_CAP_PROP_POS_AVI_RATIO, 1. );
```

```
        /* Now that we're at the end, read the AVI position in frames */
        number_of_frames = (int) cvGetCaptureProperty( input_video,
CV_CAP_PROP_POS_FRAMES );

        printf("\n The total number of frames is %d \n", number_of_frames);

        /* Return to the beginning */
        cvSetCaptureProperty( input_video, CV_CAP_PROP_POS_FRAMES, 0. );
///
///   Optical flow
///
        /* Create a windows called "Optical Flow" for visualizing the output.
         * Have the window automatically change its size to match the output.
         */

        cvNamedWindow("Optical Flow", CV_WINDOW_AUTOSIZE);
        long current_frame = 0;

/// ffmpeg ////
  // Read frames and save first five frames to disk
  i=0;
  j=0;

CvVideoWriter  *output  =  cvCreateVideoWriter("../../video2test/output.avi",
CV_FOURCC('D', 'I', 'V', 'X'), framerate, cvSize(frame_size.width,frame_size.height),1);

  while(av_read_frame(pFormatCtx, &packet)>=0) {
    // Is this a packet from the video stream?
    if(packet.stream_index==videoStream) {
///
// Decode video frame
                avcodec_decode_video(pCodecCtx, pFrame, &frameFinished,
packet.data, packet.size);
// Did we get a video frame?
     if(frameFinished) {

          if(pFrame->pict_type ==  FF_I_TYPE) { // is the frame I frame ?

//        while(1)
//        {
      static IplImage *frame = NULL, *frame1 = NULL, *frame1_1C = NULL,
*frame2_1C = NULL, *eig_image = NULL, *temp_image = NULL, *pyramid1 =
NULL,                          *pyramid2 = NULL;

                /* Go to the frame we want.  Important if multiple frames are queried in
                 * the loop which they of course are for optical flow.  Note that the very
```

```
                * first call to this is actually not needed. (Because the correct position
                * is set outsite the for() loop.)
                */
            current_frame = pFrame->coded_picture_number/number_of_frames;
                //cvSetCaptureProperty( input_video, CV_CAP_PROP_POS_FRAMES,
current_frame );

                cvSetCaptureProperty( input_video, CV_CAP_PROP_POS_AVI_RATIO,
current_frame );

                printf("\n The current frame is %d: \n " , pFrame-
>coded_picture_number);

                /* Get the next frame of the video.
                * IMPORTANT!  cvQueryFrame() always returns a pointer to the _same_
                * memory location.  So successive calls:
                * frame1 = cvQueryFrame();
                * frame2 = cvQueryFrame();
                * frame3 = cvQueryFrame();
                * will result in (frame1 == frame2 && frame2 == frame3) being true.
                * The solution is to make a copy of the cvQueryFrame() output.
                */
                frame = cvQueryFrame( input_video );
                if (frame == NULL)
                {
                        /* Why did we get a NULL frame?  We shouldn't be at the end. */
                        fprintf(stderr, "Error: Hmm. The end came sooner than we
thought.\n");
                        return -1;
                }
                /* Allocate another image if not already allocated.
                * Image has ONE channel of color (ie: monochrome) with 8-bit "color"
depth.
                * This is the image format OpenCV algorithms actually operate on
(mostly).
                */
                allocateOnDemand( &frame1_1C, frame_size, IPL_DEPTH_8U, 1 );
                /* Convert whatever the AVI image format is into OpenCV's preferred
format.
                */
                cvConvertImage(frame, frame1_1C, 0);

                /* We'll make a full color backup of this frame so that we can draw on it.
                * (It's not the best idea to draw on the static memory space of
cvQueryFrame().)
                */
```

```
		allocateOnDemand( &frame1, frame_size, IPL_DEPTH_8U, 3 );
		cvConvertImage(frame, frame1, 0);

		/* Get the second frame of video.  Same principles as the first. */
		frame = cvQueryFrame( input_video );
		if (frame == NULL)
		{
			fprintf(stderr, "Error: Hmm. The end came sooner than we
thought.\n");
			return -1;
		}
		allocateOnDemand( &frame2_1C, frame_size, IPL_DEPTH_8U, 1 );
		cvConvertImage(frame, frame2_1C, 0);

		/* Shi and Tomasi Feature Tracking! */
		/* Preparation: Allocate the necessary storage. */
		allocateOnDemand( &eig_image, frame_size, IPL_DEPTH_32F, 1 );
		allocateOnDemand( &temp_image, frame_size, IPL_DEPTH_32F, 1 );

		/* Preparation: This array will contain the features found in frame 1. */
		CvPoint2D32f frame1_features[400];

		/* Preparation: BEFORE the function call this variable is the array size
		 * (or the maximum number of features to find).  AFTER the function call
		 * this variable is the number of features actually found.
		 */
		int number_of_features;

		/* I'm hardcoding this at 400.  But you should make this a #define so that
you can
		 * change the number of features you use for an accuracy/speed tradeoff
analysis.
		 */
		number_of_features = 400;

		/* Actually run the Shi and Tomasi algorithm!!
		 * "frame1_1C" is the input image.
		 * "eig_image" and "temp_image" are just workspace for the algorithm.
		 * The first ".01" specifies the minimum quality of the features (based on
the eigenvalues).
		 * The second ".01" specifies the minimum Euclidean distance between
features.
		 * "NULL" means use the entire input image.  You could point to a part of
the image.
		 * WHEN THE ALGORITHM RETURNS:
		 * "frame1_features" will contain the feature points.
```

```
                * "number_of_features" will be set to a value <= 400 indicating the
number of feature points found.
                */
                cvGoodFeaturesToTrack(frame1_1C, eig_image, temp_image,
frame1_features, &number_of_features, .01, .01, NULL,3,0,0.04);

                /* Pyramidal Lucas Kanade Optical Flow! */

                /* This array will contain the locations of the points from frame 1 in frame
2. */
                CvPoint2D32f frame2_features[400];

                /* The i-th element of this array will be non-zero if and only if the i-th
feature of
                * frame 1 was found in frame 2.
                */
                char optical_flow_found_feature[400];

                /* The i-th element of this array is the error in the optical flow for the i-th
feature
                * of frame1 as found in frame 2.  If the i-th feature was not found (see the
array above)
                * I think the i-th entry in this array is undefined.
                */
                float optical_flow_feature_error[400];

                /* This is the window size to use to avoid the aperture problem (see slide
"Optical Flow: Overview"). */
                CvSize optical_flow_window = cvSize(3,3);

                /* This termination criteria tells the algorithm to stop when it has either
done 20 iterations or when
                * epsilon is better than .3.  You can play with these parameters for speed
vs. accuracy but these values
                * work pretty well in many situations.
                */
                CvTermCriteria optical_flow_termination_criteria
                        = cvTermCriteria( CV_TERMCRIT_ITER |
CV_TERMCRIT_EPS, 20, .3 );

                /* This is some workspace for the algorithm.
                * (The algorithm actually carves the image into pyramids of different
resolutions.)
                */
                allocateOnDemand( &pyramid1, frame_size, IPL_DEPTH_8U, 1 );
                allocateOnDemand( &pyramid2, frame_size, IPL_DEPTH_8U, 1 );
```

```
                /* Actually run Pyramidal Lucas Kanade Optical Flow!!
                 * "frame1_1C" is the first frame with the known features.
                 * "frame2_1C" is the second frame where we want to find the first frame's
features.
                 * "pyramid1" and "pyramid2" are workspace for the algorithm.
                 * "frame1_features" are the features from the first frame.
                 * "frame2_features" is the (outputted) locations of those features in the
second frame.
                 * "number_of_features" is the number of features in the frame1_features
array.
                 * "optical_flow_window" is the size of the window to use to avoid the
aperture problem.
                 * "5" is the maximum number of pyramids to use.  0 would be just one
level.
                 * "optical_flow_found_feature" is as described above (non-zero iff
feature found by the flow).
                 * "optical_flow_feature_error" is as described above (error in the flow for
this feature).
                 * "optical_flow_termination_criteria" is as described above (how long the
algorithm should look).
                 * "0" means disable enhancements.  (For example, the second array isn't
pre-initialized with guesses.)
                 */
                cvCalcOpticalFlowPyrLK(frame1_1C, frame2_1C, pyramid1, pyramid2,
frame1_features, frame2_features, number_of_features, optical_flow_window, 5,
optical_flow_found_feature, optical_flow_feature_error,
optical_flow_termination_criteria, 0 );

                /* For fun (and debugging :)), let's draw the flow field. */
                for(i = 0; i < number_of_features; i++)
                {
                        /* If Pyramidal Lucas Kanade didn't really find the feature, skip it.
*/
                        if ( optical_flow_found_feature[i] == 0 )       continue;

                        int line_thickness; line_thickness = 1;
                        /* CV_RGB(red, green, blue) is the red, green, and blue
components
                         * of the color you want, each out of 255.
                         */
                        CvScalar line_color;                    line_color =
CV_RGB(255,0,0);

                        /* Let's make the flow field look nice with arrows. */
```

```
                        /* The arrows will be a bit too short for a nice visualization
because of the high framerate
                         * (ie: there's not much motion between the frames).  So let's
lengthen them by a factor of 3.
                         */
                        CvPoint p,q;
                        p.x = (int) frame1_features[i].x;
                        p.y = (int) frame1_features[i].y;
                        q.x = (int) frame2_features[i].x;
                        q.y = (int) frame2_features[i].y;

                        double angle;           angle = atan2( (double) p.y - q.y, (double)
p.x - q.x );
                        double hypotenuse;     hypotenuse = sqrt( square(p.y - q.y) +
square(p.x - q.x) );

                        /* Here we lengthen the arrow by a factor of three. */
                        q.x = (int) (p.x - 3 * hypotenuse * cos(angle));
                        q.y = (int) (p.y - 3 * hypotenuse * sin(angle));

                        /* Now we draw the main line of the arrow. */
                        /* "frame1" is the frame to draw on.
                         * "p" is the point where the line begins.
                         * "q" is the point where the line stops.
                         * "CV_AA" means antialiased drawing.
                         * "0" no fractional bits in the center cooridinate or radius.
                         */
                        cvLine( frame1, p, q, line_color, line_thickness, CV_AA, 0 );
                        /* Now draw the tips of the arrow.  I do some scaling so that the
                         * tips look proportional to the main line of the arrow.
                         */
                        p.x = (int) (q.x + 9 * cos(angle + pi / 4));
                        p.y = (int) (q.y + 9 * sin(angle + pi / 4));
                        cvLine( frame1, p, q, line_color, line_thickness, CV_AA, 0 );
                        p.x = (int) (q.x + 9 * cos(angle - pi / 4));
                        p.y = (int) (q.y + 9 * sin(angle - pi / 4));
                        cvLine( frame1, p, q, line_color, line_thickness, CV_AA, 0 );
                }
                /* Now display the image we drew on.  Recall that "Optical Flow" is the
name of
                 * the window we created above.
                 */
                cvShowImage("Optical Flow", frame1);
                /* And wait for the user to press a key (so the user has time to look at the
image).
```

```
                    * If the argument is 0 then it waits forever otherwise it waits that number
of milliseconds.
                    * The return value is the key the user pressed.
                    */
                    //int key_pressed;
                    //key_pressed = cvWaitKey(0);

                    cvWriteFrame( output, frame1);

                    /* If the users pushes "b" or "B" go back one frame.
                     * Otherwise go forward one frame.
                     */
          //        if (key_pressed == 'b' || key_pressed == 'B')  current_frame--;
          //        else      current_frame++;
                    /* Don't run past the front/end of the AVI. */
                    //if (current_frame < 0)          current_frame = 0;
                    //if (current_frame >= number_of_frames - 1) current_frame =
number_of_frames - 2;
           } // end of the if for the I frame
             } //end of  if(frameFinished)
//         } //end while(1) of OpenCV
    } // end of if ffmpeg

 // Free the packet that was allocated by av_read_frame
    av_free_packet(&packet);
  } // end of while ffmpeg

  // Free the RGB image
  av_free(buffer);
  av_free(pFrameRGB);

  // Free the YUV frame
  av_free(pFrame);

  // Close the codec
  avcodec_close(pCodecCtx);

  // Close the video file
  av_close_input_file(pFormatCtx);

  cvReleaseVideoWriter( &output );

  return 0;

}
```

REFERENCES

[1] Levmar : "Levenberg-Marquardt nolinear least squares algorithms in C/C++". http://www.ics.forth.gr/~lourakis/levmar/

[2] General-Purpose Computation on Graphics Hardware (GPGPU).http://gpgpu.org/.

[3] Mårten Björkman. School of Computer Science and Communication. "A CUDA implementation of SIFT". http://www.csc.kth.se/~celle/

[4] Rob Hess. School of EECS at the Oregon State University. "SIFT feature detector". http://web.engr.oregonstate.edu/~hess/.

[5] CUDA Zone. NVIDIA. http://www.nvidia.com/object/cuda_home_new.

[6] OpenCL - The open standard for parallel programming of heterogeneous systems. Khronos group.http://www.khronos.org/opencl/

[7] S. Borman. Topics in Multiframe Superresolution Restoration. PhD dissertation, University of Notre Dame, Notre Dame, Indiana, May 2004.

[8] M. C. Chiang and T. E. Boulte, “Efficient Super-Resolution Image via Image Warping,” Image Vis. Comput., vol. 18, no. 10, July 2000.

[9] M. Elad and Y. Hel-Or, “A Fast Super-Resolution Reconstruction Algorithm for Pure Translational Motion and Common Space Invariant Blur.” IEEE Trans. Image Processing, vol. 10, Aug. 2001.

[10] M. Irani, B. Rousso, and S. Peleg, “Computing Occluding and Transparent Motions.” International Journal of Computer Vision, vol. 12, no. 1, pp. 5-16, Feb. 1994.

[11] D. Keren, S. Peleg, and R. Brada, “Image Sequence Enhancement Using Sub-Pixel Displacements.” Proceedings of IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR ‘88), pp. 742–746, Ann Arbor, Michigan, June 1988.

[12] A. Zomet, A. Rav-Acha, and S. Peleg, “Robust Superresolution.” Proceedings of IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR ‘01), vol. 1, pp. 645–650, Kauai, Hawaii, Dec. 2001.

[13] R. Y. Tsai and T. S. Huang, "Multiframe Image Restoration and Registration." In Advances in Computer Vision and Image Processing, vol. 1, chapter 7, pp. 317–339, JAI Press, Greenwich, Connecticut 1984.

[14] P. Vandewalle, S. Susstrunk, and M. Vetterli, "A Frequency Domain Approach to Registration of Aliased Images with Application to Super-Resolution." EURASIP Journal on Applied Signal Processing, vol. 2006, pp. 1-14, Article ID 71459.

[15] Sung Cheol Park, Min Kyu Park, and Moon Gi Kang, "Super-Resolution Image Reconstruction: A Technical Overview." IEEE Signal Processing Magazine, May 2003.

[16] Sina Farsiu. "A Fast and Robust Framework for Image Fusion and Enhancement." PhD dissertation, University of California Santa Cruz, December 2005.

[17] J. Hadamard, Lectures on the Cauchy Problem in Linear Partial Differential Equations. Yale University Press, New Haven, CT (1923).

[18] R.Y. Tsai and T.S. Huang, "Multipleframe image restoration and registration." Advances in Computer Vision and Image Processing. Greenwich, CT:JAI Press Inc., 1984, pp. 317-339.

[19] Lyndsey C. Pickup. Machine Learning in Multiframe Image Super-resolution. PhD dissertation, University of Oxford, Oxford, United Kingdom, 2007.

[20] B. C. Tom and A. K. Katsaggelos. "Reconstruction of a high-resolution image by simultaneous registration, restoration, and interpolation of low-resolution images." In Proceedings of the IEEE International Conference on Image Processing (ICIP), pages 2539–2542, 1995.

[21] B. C. Tom, A. K. Katsaggelos, and N. P. Galatsanos. "Reconstruction of a highresolution image from registration and restoration of low resolution images." In Proceedings of the IEEE International Conference on Image Processing(ICIP), pages 553–557, 1994.

[22] D. Keren, S. Peleg, and R. Brada. "Image sequence enhancement using subpixel displacements." In IEEE Conference on Computer Vision and Pattern Recognition, pages 742–746, Ann Arbor, MI, June 1988.

[23] S. Peleg, D. Keren, and L. Schweitzer. "Improving image resolution using subpixel motion." Pattern Recognition Letters, 5(3):223–226, 1987..

[24] M. Irani and S. Peleg. "Improving resolution by image registration." Graphical Models and Image Processing, 53:231–239, 1991.

[25] M. Irani and S. Peleg. “Motion analysis for image enhancement:resolution, occlusion, and transparency.” Journal of Visual Communication and Image Representation, 4:324–335, 1993.

[26] A. Zomet, A. Rav-Acha, and S. Peleg. “Robust super-resolution”. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Kauai, Hawaii, pages 645–650, December 2001.

[27] H. Stark and P. Oskoui. “High resolution image recovery from image-plane arrays, using convex projections.” J. Opt. Soc. Am. A, 6(11):1715–1726, 1989.

[28] M. Elad and A. Feuer. “Super-resolution reconstruction of continuous image sequences.” In ICIP, pages 817–834, Kobe, Japan, October 1999.

[29] M. Elad and A. Feuer. “Super-resolution reconstruction of image sequences.” IEEE Transactions on Pattern Analysis and Machine Intelligence, 21(9):817–834, 1999.

[30] M. Elad and A. Feuer. “Super-resolution restoration of continuous image sequence adaptive filtering approach.” IEEE Tranactions on Image Processing, 8(3):387–395, March 1999.

[31] D. P. Capel. “Image Mosaicing and Super-resolution.” PhD thesis, University of Oxford, Oxford, United Kingdom, 2001.

[32] A. J. Patti, M. I. Sezan, and A. M. Tekalp. “Super resolution video reconstruction with arbitrary sampling lattices and nonzero aperture time.” IEEE Transactions on Image Processing, pages 1064–1078, August 1997.

[33] R. R. Schultz and R. L. Stevenson. “A bayesian approach to image expansion for improved definition.” IEEE Transactions on Image Processing, 3(3):233–242, 1994.

[34] R. R. Schultz and R. L. Stevenson. “Extraction of high-resolution frames from video sequences.” IEEE Transactions on Image Processing, 5(6):996–1011, June 1996.

[35] S. Farsiu, M. Elad, and P. Milanfar. “A practical approach to super-resolution.” In Proc. of the SPIE: Visual Communications and Image Processing, San-Jose, 2006.

[36] S. Farsiu, D. Robinson, M. Elad, and P. Milanfar. “Robust shift and add approach to super-resolution.” In Proc. of the 2003 SPIE Conf. on Applications of Digital Signal and Image Processing, pages 121–130, August 2003.

[37] S. Farsiu, D. Robinson, M. Elad, and P. Milanfar. “Advances and challenges in super-resolution.” International Journal of Imaging Systems and Technology, Special Issue on High Resolution Image Reconstruction, 14(2):47–57, August 2004.

[38] S. Farsiu, M. D. Robinson, M. Elad, and P. Milanfar. “Fast and robust multiframe super resolution.” Image Processing, 13(10):1327–1344, October 2004.

[39] D. Capel and A. Zisserman. “Automated mosaicing with super-resolution zoom.” In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Santa Barbara, pages 885–891, June 1998.

[40] D. P. Capel and A. Zisserman. “Super-resolution enhancement of text image sequences.” In Proceedings of the International Conference on Pattern Recognition, pages 600–605, 2000.

[41] S. Baker and T. Kanade. “Super-resolution: Reconstruction or recognition?” In IEEE-EURASIP Workshop on Nonlinear Signal and Image Processing, Baltimore,Maryland, June 2001. IEEE.

[42] S. Baker and T. Kanade. “Limits on super-resolution and how to break them.” IEEE Transactions on Pattern Analysis and Machine Intelligence, 24(9):1167–1183, 2002.

[43] N. Nguyen. “Numerical Algorithms for Image Superresolution.” PhD dissertation, Stanford University,California, July 2000.

[44] Filip Sroubek, Michael Sorel, Jiru Boldys, and Jan Flusser. “PET Image Reconstruction Using Prior Information from CT or MRI.” Proceedings of the IEEE 16th International Conference on Image Processing ICIP 2009. Cairo.

[45] Filip Sroubek and Jan Flusser. “Multichannel Blind Iterative Image Restoration.” IEEE Transactions on Image Processing, Vol. 12. N0 9. September 2003.

[46] Filip Sroubek and Jan Flusser. “Multichannel Blind Deconvolution of Spatially Misaligned Images.” IEEE Transactions on Image Processing, Vol. 14. N0 7. July 2005

[47] Jiang Tian and Kai-Kuang Ma. “Markov Chain Monte Carlo Super-resolution Image Reconstruction With Artifacts Suppression.” APCCAS 2006. IEEE Asia Pacific Conference on.

[48] M. Irani, P. Anandan, J. Bergen, R. Kumar, and S. Hsu. “Efficient representations of video sequences and their applications.” Signal Processing: Image Communication, 8(4):327-351,May 1996.

[49] M. Irani, P. Anandan, and S. Hsu. “Mosaic based representations of video sequences and their applications.” In Proc. 5th International Conference on Computer Vision, Boston, pages 605-611, 1995.

[50] R. Kumar, P. Anandan, M. Irani, J. Bergen, and K. Hanna. “Representation of scenes from collections of images.” In ICCVWorkshop on the Representation of Visual Scenes, 1995.

[51] S. Peleg. “Panoramic mosaics by manifold projection.” Technical report, Hebrew University of Jerusalem, 1997.

[52] S. Peleg, B. Rousso, A. Rav-Acha, and A. Zomet. “Mosaicing with strips on adaptive manifolds.” In Panoramic Vision: Sensors, Theory, Applications, pages 309?325, 2001.

[53] S.B. Kang and R. Szeliski. “3 D scene data recovery using omnidirectional multibaseline stereo.” In Proc. IEEE Conference on Computer Vision and Pattern Recognition, pages 364-370, 1996.

[54] R. Szeliski. “Image Mosaicing for Tele-Reality Applications.” Technical Report 94/2, Digital Equipment Corporation, Cambridge Research Lab, June 1994.

[55] M. Brown and D.G. Lowe. “Recognising Panoramas,” International Conference on Computer Vision (ICCV 2003), Nice France (October 2003), pp. 1218-25.

[56] W. Triggs, P. McLauchlan, R. Hartley, and A. Fitzgibbon. “Bundle Adjustment: A Modern Synthesis.” In Vision Algorithms: Theory and Practice, number 1883 in LNCS, pages 298–373. Springer-Verlag, Corfu,Greece, September 1999.

[57] P. J. Burt and E. H. Adelson. “A multiresolution spline with application to image mosaics.” ACM Transactions on Graphics, 2(4):217–236, 1983.

[58] J. Keller. “Inverse problems.” American Mathematical Monthly, 83(2): 107{118(February 1976).

[59] A. Tikhonov and V. Arsenin, “Solutions of Ill-Posed Problems.” V. H. Winston & Sons, Washington, D.C. (1977).

[60] M. Hanke and P. Hansen. “Regularization methods for large-scale problems.” Survey on Mathematics for Industry, 3:253-315, 1993

[61] Richard R.Schultz. “Multichannel Stochastic Image Models: Theory, Applications and Implementations.” PhD dissertation. University of Notre Dame, Indiana.

[62] He Qiang, Richard Schultz, Aldo Camargo, Yi Wang and Florent Martel. "Super-resolution image reconstruction from UAS surveillance video through affine invariant interest point-based motion estimation." Proc. SPIE 6968(2008).

[63] M. Fischler and R. Bolles. "Random sample consensus: A paradigm for model fitting with application to image analysis and automated cartography." Communications of the ACM, 24: 381-395, 1981.

[64] Fernando Caballero, Luis Merino Joaquin Ferruz, and Anibal Ollero. "Homography Based Kalman Filter for Mosaic Building. Applications to UAV postion estimation." 2007 IEEE International Conference on Robotics and Automation. Roma, Italy, April 2007.

[65] R. Szeliski, "Bayesian Modeling of Uncertainty in Low-Level Vision." Kluwer Academic Publishers, 1989.

[66] G. Wahba. "Practical approximate solutions to linear operators equations when the data are noisy." SIAM J. Numer. Anal., 14:651-667, 1977.

[67] P. Mcintosh and G. Veronis. "Solving underdetermined tracer inverse problems by spatial smoothing and cross validation." Journal of Physical Oceanography, 23:716 730, Apr. 1993.

[68] C. Tomasi and R. Manduchi, "Bilateral filtering for gray and color images," in Proceedings of IEEE Int.Conf. on Computer Vision, pp. 836–846, Jan. 1998.

[69] M. Elad, "On the bilateral filter and ways to improve it," IEEE Trans. Image Processing 11, pp. 1141–1151, Oct. 2002.

[70] N. Nguyen, P. Milanfar, and G. Golub. "Blind superresolution with generalized cross-validation using gauss-type quadrature rules." In Proceedings of the 33rd Asilomar Conference on Signals, Systems, and Computers, October 1999.

[71] EURASIP Journal on Advances in Signal Processing Volume 2010 (2010), Article ID 508089, 10 pages

[72] R. I. Hartley and A. Zisserman. "Multiple View Geometry in Computer Vision." Cambridge University Press, ISBN: 0521623049, 2000.

[73] Krystian Mikolajczyk and Cordelia Schmid. "A Performance Evaluation of Local Descriptors." IEEE Transactions of Pattern Analysis and Machine Intelligence. Vol.27 No 10, October 2005.

[74] Benny Rousso, Shmuel Peleg, and Ilan Finci. "Video Mosaicking using Manifold Projection." British Machine Vision Conference.

[75] Burt, P. J., and Adelson, E. H. “The Laplacian pyramid as a compact image code.” IEEE Trans. Communications 31 (1983), 532–540.

[76] M. Brown and D. Lowe. “Automatic Panoramic Image Stitching using Invariant Features.” International Journal of Computer Vision, 74(1), pages 59-73, 2007.

[77] Barry G. Haskell, Atul Puri, and Arun N. Netravali., [Digital Video: An Introduction to MPEG-2], International Thomson Publishing, 149-152 (1997).

[78] Zhingang Zhu, Edward M. Riseman, Allen R. Hanson, and Howard Schultz.”An efficient method for geo-referenced video mosaicing for environmental monitoring,” Journal in Machine Vision and Applications, June 2005.

[79] Evan D. Andersen, and Clark N. Taylor. “Improving MAV Pose Estimation Using Visual Information.” Proceedings of the IEEE/RSJ International Conference on Intelligence Robots and Systems. San Diego, 2007.

[80] Evan D. Andersen. “A Surveillance System to Create and Distribute Geo-Referenced Mosaics from SUAV Video.” Master Thesis, Brigham Young University, UT.

[81] S. J. Julier and J. K. Uhlmann, “A new extension of the kalman filter to nonlinear systems,” in The Proceedings of AeroSense: The 11th International Symposium on Aerospace/Defense Sensing, Simulation and Controls, Multi Sensor Fusion, Tracking and Resource Management II, 1997.

[82] G. H. Golub and C. F. Van Loan. “Matrix Computations.” The JohnsHopkinsUniversity Press, Baltimore, MD, second edition, 1989.

[83] Mark Pickering, Getian Ye, Michael Frater and Jhon Arnold. “A Transform-Domain Approach to Super-Resolution Mosaicing of Compressed Images.” 4th AIP International Conference and the 1st Congress of the IPIA. Journal of Physics: Conference Series 124 (2008) 012039.

[84] Richard R Shultz, Li Meng, and Robert L. Stevenson. “Subpixel motion estimation for multiframe resolution enhancement.” Visual Communication and Image Processing 1997, pp 1317-1328.

[85] Assaf Zomet, and Shmuel Peleg. “Efficient Super-resolution and Applications to Mosaics.” Proc of International Conference of Pattern Recognition, Sept 2000.

[86] Bryce B. Ready, Clark N. Taylor and Randal W. Beard. “A Kalman-filter Based Method for Creation of Super-resolved Mosaicks.” Robotics and Automation, 2006. UCRA 2006.

[87] Aljoscha Smolic and Thomas Wiegand. “High-resolution video mosaicing.” Proc. ICIP2001, IEEE International Conference on Image Processing, Thesaloniki, Greece, October 2001.

[88] D. Terzopoulos, “Regularization of inverse visual problems involving discontinuities.” IEEE Transactions on Pattern Analysis and Machine Intelligence, PAMI-8(4), 1986, 413–424. Reprinted in Computer Vision: Advances and Applications R. Kasturi and R. Jain (eds.), IEEE Computer Society Press, Los Alamitos, CA, 1991, 183–194.

[89] R.I. Hartley. “In defense of the eight-point algorithm.” In IEEE Transactions on Pattern Analysis and Machine Intelligence, pages 580-593, October 1997.

[90] TASE Gimbal Data sheet. Cloud Cap Technology. TASE Gimbal Data sheet. http://www.cloudcaptech.com/gimbal_tase.shtm.

[91] F. Dellaert, S. Thrun, and C. Thorpe, “Jacobian images of superresolved texture maps for model-based motion estimation and tracking,” IEEE Workshop on Applications of Computer Vision (WACV’98), 1998.

[92] Lowe, D. G., “Distinctive Image Features from Scale-Invariant Keypoints”, International Journal of Computer Vision, 60, 2, pp. 91-110, 2004.

[93] Martin A. Fischler and Robert C. Bolles (June 1981). "Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography". Comm. of the ACM 24: 381–395

[94] Yi Wang, Ronald Fevig and Richard R. Schultz. “Super-resolution Mosaicking of UAV Surveillance Video” , ICIP 2008, pp 345-348, 2008.

[95] Zafer Arican and Pascal Frossard. “Joint Registration and Super-resolution with Omnidirectional Images”. IEEE Transactions on Image Processing. 2009.

[96] Julianne Chung and James G. Nagy. “Nonlinear Least Squares and Super Resolution”. Journal of Physics: Conference Series 124 (2008) 012019.

[97] D.W. Marquardt. An Algorithm for the Least-Squares Estimation of Nonlinear Parameters. SIAM Journal of Applied Mathematics, 11(2):431–441, Jun.1963