Super-Resolution

By

Michael Smith

Supervisor: Dr. Richard Dansereau

A report submitted in partial fulfillment of the requirements of SYSC 4907 Engineering Project

Department of Systems and Computer Engineering
Faculty of Engineering
Carleton University

April 8, 2005
ABSTRACT

This report describes a multi-frame super-resolution image enhancement algorithm. Images are first aligned using an image registration routine implemented for the project, or with an open source registration library. The latter provides better accuracy.

Colour-filtered images from digital cameras need to be demosaiced to restore missing colour information before they can be analyzed. Three demosaicing algorithms were implemented and compared; one was found to provide a small improvement over the demosaicing routines embedded in camera hardware and drivers.

After alignment, frames can be scaled up and combined to produce a higher-resolution image. An iterative super-resolution enhancement algorithm is then applied to sharpen details and reduce noise and colour artifacts. The algorithm uses steepest descent to minimize penalty terms corresponding to undesirable image attributes. Comparisons using both contrived and real-world image sequences show a noticeable cosmetic quality improvement when using super-resolution instead of simply magnifying a single frame. However, the algorithm is not usually able to enhance very fine print to allow it to be read.

Many of the algorithms described in the report were developed in GNU Octave. Significant memory and speed improvements were obtained by porting the super-resolution routines to C++. The Octave and C++ libraries have been made available under an open source license to facilitate further research.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Pages</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABSTRACT</td>
<td>ii</td>
</tr>
<tr>
<td>LIST OF FIGURES</td>
<td>v</td>
</tr>
<tr>
<td>I INTRODUCTION</td>
<td>1</td>
</tr>
<tr>
<td>1.1 Problem statement</td>
<td>1</td>
</tr>
<tr>
<td>1.2 Project scope</td>
<td>2</td>
</tr>
<tr>
<td>1.3 Report organization</td>
<td>2</td>
</tr>
<tr>
<td>II IMAGE REGISTRATION</td>
<td>3</td>
</tr>
<tr>
<td>2.1 Motion models</td>
<td>3</td>
</tr>
<tr>
<td>2.2 Iterative Least-Mean-Squares (LMS)</td>
<td>5</td>
</tr>
<tr>
<td>2.2.1 Laplacian Pyramid decomposition</td>
<td>5</td>
</tr>
<tr>
<td>2.2.2 LMS estimate</td>
<td>6</td>
</tr>
<tr>
<td>2.2.3 Coarse-to-fine refinement</td>
<td>7</td>
</tr>
<tr>
<td>2.2.4 Results and complexity</td>
<td>7</td>
</tr>
<tr>
<td>2.3 Robust Multiresolution Algorithm</td>
<td>9</td>
</tr>
<tr>
<td>III COLOUR DEMOSAICING</td>
<td>11</td>
</tr>
<tr>
<td>3.1 Introduction</td>
<td>11</td>
</tr>
<tr>
<td>3.2 Bilinear interpolation</td>
<td>12</td>
</tr>
<tr>
<td>3.3 Kimmel’s algorithm</td>
<td>12</td>
</tr>
<tr>
<td>3.4 Generalized Pei-Tam method</td>
<td>14</td>
</tr>
<tr>
<td>3.5 Algorithm output comparison</td>
<td>16</td>
</tr>
<tr>
<td>3.5.1 Colour results</td>
<td>16</td>
</tr>
<tr>
<td>3.5.2 Application to image registration</td>
<td>18</td>
</tr>
<tr>
<td>3.5.3 Remarks</td>
<td>18</td>
</tr>
<tr>
<td>IV SUPER-RESOLUTION</td>
<td>19</td>
</tr>
<tr>
<td>4.1 Initial estimate</td>
<td>19</td>
</tr>
<tr>
<td>4.2 Greyscale enhancement</td>
<td>22</td>
</tr>
<tr>
<td>4.2.1 Data fidelity term</td>
<td>22</td>
</tr>
<tr>
<td>4.2.2 Regularization term</td>
<td>24</td>
</tr>
<tr>
<td>4.2.3 Results and complexity</td>
<td>25</td>
</tr>
<tr>
<td>4.3 Colour enhancement</td>
<td>30</td>
</tr>
</tbody>
</table>
# LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Bayer colour-filter array</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>Translational motion model</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>Affine motion model</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>Construction of a three-level Laplacian Pyramid</td>
<td>6</td>
</tr>
<tr>
<td>5</td>
<td>Test sequences used for the LMS algorithm</td>
<td>8</td>
</tr>
<tr>
<td>6</td>
<td>Bayer colour-filter array</td>
<td>11</td>
</tr>
<tr>
<td>7</td>
<td>Bilinear interpolation of missing red pixel values</td>
<td>11</td>
</tr>
<tr>
<td>8</td>
<td>Calculation of image gradients in the Kimmel method</td>
<td>13</td>
</tr>
<tr>
<td>9</td>
<td>Pixel grid for the Pei-Tam method</td>
<td>15</td>
</tr>
<tr>
<td>10</td>
<td>Three reconstructions of the sailing image after colour-filtering.</td>
<td>17</td>
</tr>
<tr>
<td>11</td>
<td>Three reconstructions of the lighthouse image after colour-filtering.</td>
<td>17</td>
</tr>
<tr>
<td>12</td>
<td>Block diagram of the initial estimate procedure</td>
<td>19</td>
</tr>
<tr>
<td>13</td>
<td>Two grid-filling methods</td>
<td>20</td>
</tr>
<tr>
<td>14</td>
<td>Initial estimate from four images of a textbook</td>
<td>21</td>
</tr>
<tr>
<td>15</td>
<td>Block diagram of the data fidelity term</td>
<td>23</td>
</tr>
<tr>
<td>16</td>
<td>Super-resolution enhancement of the textbook images</td>
<td>26</td>
</tr>
<tr>
<td>17</td>
<td>Affine and translational enhancement of the textbook images</td>
<td>27</td>
</tr>
<tr>
<td>18</td>
<td>Results from enhancement of a constructed sequence of 16 images</td>
<td>28</td>
</tr>
<tr>
<td>19</td>
<td>Graph of PSNR as a function of enhancement iterations</td>
<td>29</td>
</tr>
<tr>
<td>20</td>
<td>Graph of PSNR as a function of source frames</td>
<td>30</td>
</tr>
<tr>
<td>21</td>
<td>Colour bookshelf source frame and enhancement</td>
<td>33</td>
</tr>
<tr>
<td>22</td>
<td>A region from the bookshelf image, showing effects of each term</td>
<td>34</td>
</tr>
<tr>
<td>23</td>
<td>Unsuccessful license plate reconstruction from a cell phone camera</td>
<td>36</td>
</tr>
<tr>
<td>24</td>
<td>Cell phone camera reconstruction from three source frames</td>
<td>36</td>
</tr>
<tr>
<td>25</td>
<td>Algorithm implementation class diagram</td>
<td>39</td>
</tr>
</tbody>
</table>
CHAPTER I

INTRODUCTION

1.1 Problem statement

Poor image quality in consumer camera devices presents both cosmetic and practical problems. Video sequences captured from modern digital cameras are normally digitized at lower resolutions than still pictures. A “freeze-frame” from such a video would thus be of lower than desirable quality. Similarly, CMOS sensors embedded in cell phones and webcams tend to produce noisy, low-resolution images. In a more utilitarian vein, a distant suspect’s profile or license plate number may be unrecognizable at the available resolution of a surveillance video. Some form of image enhancement would be desirable, but the decreasing size of camera-enabled devices often makes it difficult to improve the hardware.

![Figure 1: The Bayer colour-filter pattern, commonly used in digital cameras.](image)

Another obstacle to image quality originates from the single intensity sensor array used on most consumer cameras. In order to capture red, green, and blue data points, a colour-filter array is placed over the sensor with a pattern similar to the one shown in Figure 1. Each sensor captures either red, green, or blue, and camera electronics or driver software interpolates the missing values in a process known as colour demosaicing. This tends to result in colour artifacts at edges, where the assumptions of the interpolation algorithm break down. It also decreases the number of valid data points available to image enhancement algorithms.

This project aims to improve image quality by combining information from a sequence of low-resolution frames to produce a single, higher-resolution still image. Both cosmetic quality and image functionality (detail recognition, especially text readability) are addressed. To facilitate image enhancement, a comfortable interface for opening source images, selecting regions of interest, and experimenting with algorithm parameters would be desirable.
The project features an implementation of a colour multi-frame super-resolution algorithm. By analyzing small, random camera or object vibrations over the course of multiple frames, super-resolution algorithms are able to compute a still image or a sequence of video frames at a higher resolution than the original camera input. Sub-pixel movements may reveal a different portion of the object to the camera in each frame; if an image registration algorithm can accurately estimate the translational or rotational motion, the views can be interpolated and stitched together to produce a blurry, high-resolution image. A super-resolution algorithm can then be applied to sharpen the image and to mask ghosting effects introduced by inaccurate image registrations.

1.2 Project scope

In the course of this project, two image registration algorithms were compared. One was implemented from scratch in GNU Octave, a MATLAB work-alike, and the other was accessed from an open source library implementation. Octave implementations of a greyscale and a colour super-resolution algorithm were developed, followed by a port of the colour algorithm to C++. A framework was created for rapidly experimenting with algorithm parameters, and a variety of test runs have shown that the super-resolution process provides noticeable quality improvements, both cosmetic and functional – although not necessarily at the same time.

Along the way, three colour demosaicing algorithms were compared, two of which were implemented in GNU Octave for the project. The algorithms were then applied to a series of raw frames from a webcam, prior to super-resolution enhancement.

1.3 Report organization

This report is organized as follows:

- Chapter 2 describes the two motion models and two image registration algorithms used in the project.
- Chapter 3 describes and compares three colour demosaicing algorithms.
- Chapter 4 explains the super-resolution enhancement algorithm.
- Chapter 5 provides details about the Octave and C++ algorithm implementations.
- Chapter 6 presents conclusions.
- Appendix A contains a few equations derived during the course of the project.
- Appendix B documents the open source software package produced as part of this work.
2.1 Motion models

The goal of the image registration process is to determine an alignment between a sequence of images by estimating motion parameters between successive frames. Video codecs such as MPEG-4 divide the image into small blocks and determine translational vectors for each block. For super-resolution applications, a motion model with a set of parameters which can be applied globally to the entire image is usually accurate enough, and offers advantages in efficiency and generality.

The super-resolution algorithm described in this report requires a few properties of the motion model:

- It must be possible to “warp” an image to a new position, given the estimated model parameters.
- The model parameters must be invertible, since the enhancement algorithm needs to warp images back from their aligned locations.
- It should be possible to compose successive motion estimates into a cumulative set of parameters. Registration accuracy is improved by determining the motion parameters between successive frames, but the super-resolution algorithm needs to know the motion parameters between the first and \( n^{th} \) frames.

During the course of the project, global translational and global affine models were employed. A global translational model assumes that one image can be warped to the next by shifting it a real number of pixels in the horizontal and vertical directions. There are therefore only two parameters, described by the equations below:

\[
\begin{align*}
    x' &= x + x_t \\
    y' &= y + y_t
\end{align*}
\]

The equations give the transformation between a coordinate \((x, y)\) in the reference image and a coordinate \((x', y')\) in the new image after translation to the right and down by \(x_t\) and \(y_t\) pixels, respectively. Figure 2 demonstrates a translational image warp. Translational models are easily
inverted (by sign-flipping) and composed (by summing).

Global affine models use six parameters to account for uniform translation, shearing, scaling, and rotation:

\[ x' = x + a_1 + a_2 x + a_3 y \]
\[ y' = y + a_4 + a_5 x + a_6 y \]

Figure 3 shows an image under affine warp. Affine models are a superset of translational models, with \( a_1 = x_t \) and \( a_4 = y_t \). They are more difficult to invert and compose, but it is possible. Derivations for inversion and composition are given in Appendix A.1.

There are other models, for example a 12-parameter model involving quadratic relations, but for ease of implementation they were not used for this project. Since the super-resolution literature reports satisfactory results with a translational model \([1]\), it was decided that an affine model would provide more than enough accuracy.

The global translational model is easier to understand and quicker to compute than the affine model. Farsiu et al \([1]\) have even optimized part of the super-resolution algorithm used in this project based on the assumption of translational motion. However, experimental results in Figure 17 (page 27) show that global affine registration, combined with the unoptimized super-resolution algorithm, produces better super-resolution results.
2.2 Iterative Least-Mean-Squares (LMS)

Bergen et al [2] present a family of motion estimators for various global models, including the affine model. The affine estimator was implemented in GNU Octave for this project, and a simplified routine supporting only translational motion was also created, for comparison.

The image registration framework presented in [2] makes the assumption that image intensity is constant between frames; that is, the brightness of an object does not change, even though the object may have moved. The algorithm consists of several steps:

1. Images are decomposed into a multi-resolution pyramid, and the high-frequency components (edges) are extracted.
2. At the smallest level of the pyramid, the parameters are estimated by using an iterative Least-Mean-Squares (LMS) technique to minimize the sum of the squared differences (SSD) between the reference image and the second image after back-warping.
3. To refine the estimate, the parameters are propagated to the next highest level of the pyramid and the LMS routine is reiterated.

The following sections describe the components of the algorithm.

2.2.1 Laplacian Pyramid decomposition

The LMS estimation tracks the movement of object edges. Consequently, some sort of highpass operator needs to be applied to the images before registration. Additionally, the LMS technique is prone to finding local minima, so it is most accurate for small displacements. If the image can be scaled to smaller dimensions, the displacement will also shrink, and accurate estimation of larger displacements will be supported.

The Laplacian Pyramid [3] provides a high-pass filtered, multi-resolution view of the images, fulfilling both requirements at once. The image is first convolved with a blur kernel that is similar to a normalized Gaussian, but constrained in a way that will be described later. The blurred image is then downsampled by a factor of two.

Next, the reduced image is blurred and expanded by a factor of two. The expanded image is subtracted from the original, leaving as an absolute difference the high-frequency components that did not survive the double blur. This difference image provides the first level of the pyramid. The process can be repeated to produce lower levels, beginning with the reduced image from the first step. Figure 4 shows the results of pyramid construction.
Figure 4: Construction of a three-level Laplacian Pyramid. Pyramid residues, at right, are shown brighter than their actual intensity.

The blur kernel is constrained so that each pixel in the reduced image contributes equally to the reconstruction of the higher-level image. A separable 5x5 kernel is normally formulated as

\[
\begin{bmatrix}
\frac{1}{4} - \frac{a}{2}, \frac{1}{4}, a, \frac{1}{4}, \frac{1}{4} - \frac{a}{2}
\end{bmatrix}
\]

where \( a \) is some parameter between \( \frac{1}{4} \) and \( \frac{1}{2} \). When \( a = 0.4 \), the resulting kernel is

\[
\begin{bmatrix}
0.05, 0.25, 0.4, 0.25, 0.05
\end{bmatrix}
\]

which is very close to a 5x5 Gaussian kernel with standard deviation of 1.0. The Laplacian Pyramid gets its name from the fact that the difference of two Gaussian blurs is an approximation to the Laplacian of a Gaussian [4].

2.2.2 LMS estimate

After the two images to be considered have been decomposed into pyramids, the motion parameters are estimated iteratively, by minimizing the error after the first image is warped toward the second. The following process is repeated five times\(^\text{1}\) at each level of the pyramid:

1. Warp the first image toward the second, using the current estimate (which is initially zero).
2. For each pixel \((x, y)\), set up the following matrix: \(^\text{2}\)

\[
X = \begin{bmatrix}
1 & x & y & 0 & 0 \\
0 & 0 & 0 & 1 & x & y
\end{bmatrix}
\]

---

\(^{1}\) In this implementation, stopping conditions are not discussed in [2]; they are considered in [5].

\(^{2}\) To restrict the estimate to translational motion, it was determined that this constant matrix could be used instead:

\[
X = \begin{bmatrix}
1 & 0 \\
0 & 1
\end{bmatrix}
\]
3. Accumulate the sums for each pixel:

\[
X^{T} \begin{bmatrix} I_x & I_y \\ I_x & I_y \end{bmatrix} X \delta a = - \sum X^{T} \begin{bmatrix} I_x \\ I_y \end{bmatrix} I_{t,\text{warp}}
\]

\(I_{t,\text{warp}}\) is the warped spatial gradient at that pixel: the second pyramid image minus the first pyramid image after warping. \(I_x\) is the spatial gradient in the horizontal direction at that pixel. The matrix version of \(I_x\) is defined as the convolution of the second pyramid image and \([-0.5, 0, 0.5]\). Similarly, \(I_y\) is the vertical spatial gradient, defined as the convolution of the second pyramid image and \([-0.5, 0, 0.5]^T\).

4. Use Gaussian elimination or any other technique to solve for \(\delta a\), the model parameter increment.

5. Add the increment to the model parameters.

2.2.3 Coarse-to-fine refinement

The registration process begins at the coarsest level of detail, at the smallest level of the pyramid. Between levels, the estimate is propagated to the new level. For affine models, this simply means the translational parameters are doubled. It is important to note that each propagation doubles any error in the estimate, and estimation at the next level may not be able to correct the error. With too many pyramid levels, the estimation error will be unacceptable; with too few levels, the LMS estimator may find a local minimum rather than the true global motion.

Proper selection of the number of levels depends on image size and motion velocity. Through experimentation, it was determined that three to four levels is a reasonable pyramid size, and that allowing any levels smaller than 30 by 30 pixels can result in completely inaccurate estimates.

2.2.4 Results and complexity

The implementation was tested with a real image that was subjected to contrived translational and affine motion. The translational test was created by cropping two 218x314 pixel areas from an image of a bookshelf, offset from each other by two pixels in the horizontal direction. The areas were then scaled by 0.33, resulting in a sub-pixel displacement motion of approximately 0.66...
pixels at the new scale. The registration algorithm identified the displacement as $-0.708$ pixels in the horizontal direction and $-0.0583$ pixels vertically, which is not far off. For a typical super-resolution applications with a super-resolution factor of 4, the difference between $-0.666$ and $-0.707$ (similarly, 0.0 and $-0.0583$) would not be enough to misalign the image to the wrong position in the high-resolution grid.

The affine test was created using the entire bookshelf image. A copy of the image was rotated 4 degrees about its center, then scaled to 109% and cropped back to its original size to remove black borders. Both the LMS implementation and the Motion2D package from the following section were used to register the frames, yielding similar estimates. While the LMS algorithm came up with $a = [2.92, 0.0847, -0.0951, -29.67, 0.0949, 0.0855]$, the Motion2D package estimated $a = [2.88, 0.0849, -0.0949, -29.52, 0.0950, 0.0861]$. The proximity of the two estimates produced by different algorithms suggest the LMS algorithm produces reasonable results for two-image sequences. Figure 5 shows the translational and affine test sequences used.
The algorithm was then tested against a sequence of 16 images, which were registered, back-warped toward the first image, and viewed in rapid succession to check for jumps caused by registration errors. Since the LMS estimator prefers small displacements, images in a sequence are registered successively, and parameters are then combined as described in Appendix A.1.2 to create cumulative displacements from the first frame. After the first four frames, registration errors began to accumulate noticeably.

For images that are $n$ by $n$ pixels in size, the complexity of the LMS estimation at the largest level of the pyramid is $O(n^2)$. The number of pyramid levels is usually restricted to about $\log n - 5$, giving a total complexity of $O(n^2 \log n)$.

2.3 Robust Multiresolution Algorithm

The “RMRmod” algorithm described in [5] also makes use of pyramidal decomposition. However, it presents several enhancements to the LMS approach:

- It introduces a new parameter to account for changes in global image intensity (illumination) between frames.
- It replaces the LMS estimator with Iteratively Reweighted Least Squares (IRLS). LMS estimates can be thrown off by even just one outlier in the data set (e.g., a noisy pixel, or an object under secondary local motion). IRLS is more robust: on every increment, each pixel in the pyramid is assigned a weight between 0 and 1.0 which controls its relevance to the global estimation process.
- It assumes a translational motion model for the first level or two of the pyramid, to create a stable base for subsequent affine estimates.
- Stopping conditions are better specified. At each pyramid level, the algorithm stops iterating when the norm of the increment dips below a certain maximal value indicating convergence. The increment norm is defined as a linear combination of the affine model parameters, with some of the parameter weights depending on pixel weights in the region of movement.

To compensate for extra local minima in the IRLS estimator as compared to LMS, the RMRmod algorithm makes use of a parameter, $C$, which changes the shape of the minimization problem. For the first incremental calculation, $C$ is set to a value which causes the estimator to reduce the number of local minima while making it more sensitive to outliers. As pixel weights are established in subsequent increments, $C$ is reduced to make the estimator ignore outliers. This intelligent
scaling allows the estimator to avoid local minima when it is most critical, at the coarsest level of the pyramid.

An open source implementation of the RMRmod algorithm, “Motion2D,” [6] is available and was used for this project.

Odobez and Bouthemy [5] compared LMS and RMRmod rigorously, putting the algorithms to work against randomly contrived motion sequences and two short real-life videos. For the purposes of this project, it was enough to feed in a stream of 16 frames of a still scene, captured with a shaky handheld camera. The frames were registered and back-warped toward the first frame. They were then viewed in rapid succession to check for jumps that would be caused by inaccurate registrations. Whereas the LMS algorithm accumulated significant errors after the first four or five frames, the RMRmod implementation was able to align all frames properly. As discussed in [5], the algorithm’s complexity is similar to that of the LMS algorithm; although the IRLS estimation increment takes longer, it converges more quickly.
CHAPTER III

COLOUR DEMOSAICING

3.1 Introduction

Most consumer cameras have a single charge-coupled device sensor array. In order to capture red, green, and blue data points, a colour-filter array is placed over the sensors with a pattern similar to the one shown in Figure 6. Each sensor captures either red, green, or blue, and camera electronics or driver software interpolates the missing values in a process known as colour demosaicing. There are more green values than red or blue, because the human eye is more sensitive to green light than the other primaries. Interpolation algorithms generally make the assumption that colour intensity changes smoothly from one sample to the next. This assumption breaks down at edges, resulting in colour artifacts.

![Figure 6: The Bayer colour-filter pattern.](image)

![Figure 7: Bilinear interpolation of missing red pixel values.](image)

It was hypothesized that these edge artifacts would degrade the accuracy of the image registration, in addition to being visually unpleasant. Since it is possible to extract raw, colour-filtered frames from the Linux USB QuickCam driver, and demosaic them in GNU Octave, it was decided to implement an alternate algorithm and compare it with the driver’s best-quality routine. The goal was to reduce edge artifacts in the demosaiced images and in turn improve the output of the super-resolution enhancement.
The following sections describe the three demosaicing algorithms that were compared.

### 3.2 Bilinear interpolation

Bilinear interpolation is the simplest way to reconstruct colour values. Each colour channel is handled in isolation; neighbouring pixels are averaged to obtain missing colour values. Figure 7 shows how missing red values are interpolated at green and blue pixels. In general either two or four neighbours are available for each missing value, except along the edges and in corners.

This method of interpolation is very computationally efficient, requiring a few additions and a barrel-shift for each missing value. It also produces very poor output at edges, as shown in Figure 10(f).

### 3.3 Kimmel’s algorithm

Kimmel [8] proposes a better method which avoids interpolating across edges. Additionally he makes the assumption that inside any given object, the ratios of green values to red values and green values to blue values should be locally constant. These colour cross ratios are used to reduce colour artifacts in the reconstruction. The green channel is favoured because it contributes most strongly to the luminance of an image, and thus defines the image’s edges.

First, edge indicators are determined based on image gradients, and used to weight a bilinear reconstruction of the green channel. Next, the green channel and the colour cross ratios at neighbouring pixels are used to reconstruct the red and blue channels, with edge indicators again used to avoid cross-edge interpolation. A correction loop is then iterated three times to adjust the green values to make the colour cross ratio locally constant, and to correct the blue and green values to match the adjusted cross ratios. Kimmel also defines a second non-linear enhancement step using inverse colour diffusion along edges. Since the goal of this project was to produce quality super-resolution output and not a demosaicing implementation, the second step was not implemented.

Figure 8 shows how derivatives can be calculated at a typical blue pixel. Using the pixel numbers from the figure,

\[
D_x = (G_6 - G_4)/2 \\
D_y = (G_2 - G_8)/2
\]

---

1 The sailing and lighthouse images used in this chapter were obtained from Ron Kimmel’s website [7].
Figure 8: Calculation of image gradients at pixel B5 in the Kimmel method.

\[
D'_x = (R3 - R7)/(2\sqrt{2})
\]

\[
D'_y = (R1 - R9)/(2\sqrt{2})
\]

Derivatives for every other pixel can be similarly calculated. \(^2\)

To prevent interpolation across edges, an edge indicator must be formulated to provide lower weights to pixels that are on edges. Each pixel has eight neighbours, and consequently there must be eight edge indicators. At any given pixel, the edge indicator for each direction is a function of the derivative in that direction at the current pixel and at the neighbouring pixel. For example, the lower-right edge indicator at pixel B5 in Figure 8 would be a function of \(D'_y\) at pixels B5 and R9. Call these two derivatives \(D_A\) and \(D_B\). Kimmel suggests the use of the following decreasing function as the edge indicator:

\[
e = \frac{1}{\sqrt{1 + D_A^2 + D_B^2}}
\]

In the initialization step of the algorithm, the green channel is bilinearly reconstructed using the edge function to prevent interpolation across edges:

\[
G_{ij} = \frac{e_{ij}^{i-1,j}G_{i-1,j} + e_{ij}^{i+1,j}G_{i+1,j} + e_{ij}^{i,j-1}G_{i,j-1} + e_{ij}^{i,j+1}G_{i,j+1}}{e_{ij}^{i-1,j} + e_{ij}^{i+1,j} + e_{ij}^{i,j-1} + e_{ij}^{i,j+1}}
\]

Here, \(e_{ij}^{i-1,j}\) is the central upper edge indicator at \((x, y) = (j, i)\), \(G_{i-1,j}\) is the green pixel value at \((j, i-1)\), and so on. The missing blue and red pixels are then interpolated using the green channel and the colour cross ratios, \(\frac{B}{G}\) and \(\frac{R}{G}\). For example, missing blue pixels at red locations are interpolated as follows:

\[
B_{ij} = G_{ij} \frac{e_{ij}^{i+1,j+1}B_{i+1,j+1} + e_{ij}^{i-1,j-1}B_{i-1,j-1} + e_{ij}^{i+1,j-1}B_{i+1,j-1} + e_{ij}^{i-1,j+1}B_{i-1,j+1}}{e_{ij}^{i+1,j} + e_{ij}^{i-1,j} + e_{ij}^{i,j-1} + e_{ij}^{i,j+1}}
\]

\(^2\) There is a modification for \(D'_x\) and \(D'_y\) at green pixels, but because of space constraints it is not explained here.
In the correction loop, the interpolated green values are then corrected to fit the green-blue cross ratio:

\[
G_{ij}^B = B_{ij} \left[ e_{ij}^{i,j} \frac{G_{i+1,j} + e_{ij}^{i+1,j} G_{i-1,j} + e_{ij}^{i,j+1} G_{i,j+1} + e_{ij}^{i,j-1} G_{i,j-1}}{B_{i+1,j} + e_{ij}^{i+1,j} + e_{ij}^{i-1,j} + e_{ij}^{i,j+1} + e_{ij}^{i,j-1} + e_{ij}^{i,j+1} + e_{ij}^{i,j-1}} \right] \\
+ e_{ij}^{i+1,j+1} G_{i+1,j+1} B_{i+1,j+1} + e_{ij}^{i-1,j-1} G_{i-1,j-1} B_{i-1,j-1} + e_{ij}^{i+1,j} G_{i+1,j} B_{i+1,j} + e_{ij}^{i-1,j} G_{i-1,j} B_{i-1,j} + e_{ij}^{i,j+1} G_{i,j+1} B_{i,j+1} + e_{ij}^{i,j-1} G_{i,j-1} B_{i,j-1} \\
/ \left[ e_{ij}^{i+1,j} + e_{ij}^{i-1,j} + e_{ij}^{i,j+1} + e_{ij}^{i,j-1} + e_{ij}^{i,j+1} + e_{ij}^{i,j-1} + e_{ij}^{i,j+1} + e_{ij}^{i,j-1} \right]
\]

A similar correction is performed to fit the green-red cross ratio, and the average is taken to obtain the final green channel. The blue and red values are then adjusted to fit the red-green and blue-green colour ratios, respectively.

It was found that green values at edges were being dramatically brightened during the correction loop. At edges where the blue and red pixel values were very close to zero, the cross ratios $G_B^G$ and $G_R^G$ were very large and were not being properly reduced by the edge indicators. Observing that [9] mentions the absolute value operator in conjunction with edge sensing algorithms, the edge indicator function was rewritten as follows:

\[
e = \frac{1}{1 + 4|DA| + 4|DB|}
\]

This modified edge indicator decreases more quickly when $DA$ and $DB$ increase, causing contributions from cross-edge interpolation to be ignored more vigorously.

There are also artifacts near the image borders where entire rows and columns of the green channel are saturated during the correction step. This was worked around by searching the image for value adjustments greater than about 15% and restoring the pre-adjustment value.

All of the operations performed by the algorithm have linear complexity, depending only on the number of pixels processed. However, the algorithm relies heavily on division, traditionally a slow operation. Eight edge indicators also need to be stored in memory for each pixel, unless they are calculated on the fly at each correction step. The following section describes an algorithm which provides comparable quality for a much lower computational cost.

### 3.4 Generalized Pei-Tam method

Pei and Tam [9] propose a high-quality reconstruction algorithm. Their algorithm has been implemented in the Linux USB QuickCam driver, and is activated on the driver’s “best quality” setting.
Pei and Tam extend Kimmel’s assumption that the colour cross ratios, $\frac{B}{C}$ and $\frac{R}{G}$, are locally constant. To avoid division, they transform colours to a “contrast domain”:

\[
K_R = G - R \\
K_B = G - B
\]

It turns out that $K_R$ and $K_B$ are in general locally constant and can be used to interpolate missing values similarly to how the cross ratios are used in Kimmel’s method. Edge sensing is not used; the new algorithm does interpolate across edges. However, Pei and Tam show that the standard deviations of $K_R$ and $K_B$ are much smaller than that of the green channel, which in turn means they vary by smaller amounts across edges. In fact, they are almost constant across edges. Because $K_R$ and $K_B$ are not completely constant, this algorithm gives slightly higher interpolation errors at edges than an edge sensing method such as Kimmel’s. However, it is many times better than the bilinear interpolation method, and requires only about three times the processing power.

<table>
<thead>
<tr>
<th>R1</th>
<th>G2</th>
<th>R3</th>
<th>G4</th>
<th>R5</th>
</tr>
</thead>
<tbody>
<tr>
<td>G6</td>
<td>B7</td>
<td>G8</td>
<td>B9</td>
<td>G10</td>
</tr>
<tr>
<td>R11</td>
<td>G12</td>
<td>R13</td>
<td>G14</td>
<td>R15</td>
</tr>
<tr>
<td>G16</td>
<td>B17</td>
<td>G18</td>
<td>B19</td>
<td>G20</td>
</tr>
<tr>
<td>R21</td>
<td>G22</td>
<td>R23</td>
<td>G24</td>
<td>R25</td>
</tr>
</tbody>
</table>

**Figure 9:** Pixel grid for the Pei-Tam method.

The algorithm will be explained by way of example using the grid from Figure 9. A green value is interpolated for the central pixel, R13, as follows:

\[
G'_{13} = R_{13} + (K'_{R8} + K'_{R18} + K'_{R12} + K'_{R14})/4 \\
K'_{R8} = G_{8} - R_{8}' = G_{8} - (R_{3} + R_{13})/2 \\
K'_{R18} = G_{18} - R_{18}' = G_{18} - (R_{13} + R_{23})/2 \\
K'_{R12} = G_{12} - (R_{11} + R_{13})/2 \\
K'_{R14} = G_{14} - (R_{13} + R_{15})/2 \\
\rightarrow G'_{13} = R_{13} + G_{8}/4 - R_{3}/8 - R_{13}/4 + G_{18}/4 - R_{23}/8
\]
The blue value at R13 is interpolated through a similar process:

\[
B'_{13} = G'_{13} - (K'_{B7} + K'_{B9} + K'_{B17} + K'_{B19})
\]

\[
K'_{B7} = G'_{7} - B_{7}
\]

\[\text{etc.}\]

The implementation in the Linux QuickCam driver adds a set of constants to weight the effect of red neighbours on green values, blue neighbours on green values, green neighbours on blue values, and so on. The constants have been tuned to increase webcam sharpness. This implementation is referred to in the source code as the Generalized Pei-Tam Method (GPTM). It was extracted from the kernel driver and moved into a userspace wrapper program to be tested for this chapter.

### 3.5 Algorithm output comparison

#### 3.5.1 Colour results

Figure 10 shows a full-colour sailing image which has been colour-filtered and passed through the three algorithms for reconstruction. Artifacts are visible on the sailboat masts in all three cases, but Kimmel’s method appears to do the best job of producing a solid black line. Figure 11 presents a more pathological image of a lighthouse: after colour-filtering, the fenceposts and the siding on the house are heavily undersampled, resulting in colour aliasing in all the reconstructed images. Again, Kimmel’s algorithm does the best job of removing the aliases, although it also darkens the red lifesaver slightly.

<table>
<thead>
<tr>
<th>Image</th>
<th>PSNR (dB)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Bilinear</td>
</tr>
<tr>
<td>Sailing</td>
<td>26.797</td>
</tr>
<tr>
<td>Lighthouse</td>
<td>11.940</td>
</tr>
</tbody>
</table>

**Table 1:** Peak signal-to-noise ratio of each reconstruction performed using the three algorithms. The PSNR’s of the three colour channels are averaged.

Table 1 compares the algorithms’ output peak signal-to-noise ratio (PSNR) from the examples where a reference image is available. Interestingly, higher PSNR’s do not necessarily correspond to fewer edge artifacts: for the sailing image, the PSNR of the bilinear interpolation algorithm beats that of the GPTM, despite the clearly visible colour artifacts.
Figure 10: Three demosaiced reconstructions of the sailing image after colour-filtering.

Figure 11: Three demosaiced reconstructions of the lighthouse image after colour-filtering.
3.5.2 Application to image registration

The registration algorithms used in this project operate on greyscale images. It was hypothesized that if a demosaicing algorithm can improve colour image quality, especially at images, it would also improve registration results after conversion to greyscale.

The original sailing image and its Kimmel and GPTM reconstructions were converted to grey. The PSNR of the greyscale GPTM reconstruction was 26.5 dB, whereas for the Kimmel reconstruction it was 32.5 dB. Consequently there was some evidence to support the idea that a higher quality colour reconstruction would affect the registration.

A series of seven raw 320x480 frames were obtained from a Logitech QuickCam and demosaiced using the bilinear, Kimmel, and GPTM algorithms. All three series were converted to greyscale and registered. The absolute difference between the various registrations was under 0.005 pixels for the two translational parameters, and under 0.00003 for the four affine parameters. Apparently, any edge artifacts introduced by the bilinear algorithm do not actually affect the registration to any noticeable degree.

3.5.3 Remarks

The conclusion was made that the GPTM algorithm does not perform significantly worse than Kimmel’s method. Consequently, there is no advantage to obtaining raw images from the QuickCam driver and demosaicing them in a post-processing step. For digital still image cameras, the benefit gained would depend on the quality of the camera’s demosaicing electronics. However, extensive examination of images from consumer digital cameras, even an older model from 1999, found no suitable examples of edge artifacts. This indicates that consumer cameras are already doing a reasonable job of demosaicing.
CHAPTER IV

SUPER-RESOLUTION

After registering a series of low-resolution frames, it is a simple matter to warp them into alignment on a higher-resolution grid. This chapter describes the alignment step and the enhancement step that follows.

4.1 Initial estimate

Once each image’s alignment at the sub-pixel level has been determined with respect to the first image, the images can be individually upsampled by an integer super-resolution factor, $r$, then warped into place. The registration motion models must first be corrected to match the higher resolution. For affine models, this is accomplished by multiplying the translational motion parameters by the super-resolution factor, leaving the other four parameters untouched.

![Figure 12: Block diagram of the initial estimate procedure.](image)

In this project, both upsampling and warping were performed using bilinear interpolation, but nearest neighbour interpolation would also be a valid choice, reducing execution time at the expense of quality. Each source image thus has an interpolated value at each position in the high-resolution grid. The values are then combined by taking the median value at each pixel. Although it would be computationally easier to take the average value, the median operator is more robust to outliers [1], as might be caused by registration inaccuracies, camera sensor noise, or blur. The output from
this value combination is a blurry high-resolution image, ready for further enhancement. Figure 12 shows a block diagram of the initial estimate process.

(a) Full interpolation

(b) Shifting only, as in [1]

Figure 13: Two methods of filling the high-resolution grid from one source image, after warping it to the right and down by one high-resolution pixel (super-resolution factor $r = 2$).

An alternative method is suggested in [1], where pixels from each source image are shifted to their corresponding positions on the high-resolution grid. Figure 13 demonstrates the difference. In cases where multiple source pixels are aligned to the same destination pixel, the median operator is used to select a value. The authors refer to this process as Median Shift and Add (MSA). In the translational motion model they prefer, source frames can be simply shifted into place by a constant offset, using nearest neighbour interpolation. Theoretically, for a super-resolution factor of $r$, $r^2$ source frames are required to fill the high-resolution grid, and each frame must have a sub-pixel displacement corresponding to a different grid position. For the under-determined case where not all grid positions are represented by the source images, the missing positions must be interpolated somehow, either by plucking pixels from nearby positions in one of the source images or by a more complex blending operation. In addition, when working with an affine motion model there is not a simple pattern of unrepresented grid positions, so it becomes harder to determine which positions need to be filled. With the implementation presented in this report, better results were obtained by interpolating values from each source frame to each high-resolution grid position. However, the term “Median Shift and Add” has persisted.

In Figure 14, four pictures of a textbook were captured from approximately one meter away using an HP Photosmart 318 at 1792x1200 resolution and the lowest JPEG compression level available. The images were converted to grey and enlarged by a factor of two using pixel doubling, bicubic interpolation, and the initial estimate routine, for comparison. The outputs were cropped to show a few lines of text from the page. The initial estimate is clearly an improvement over
Figure 14: Initial estimate from four images of a textbook, comparing results with the effect of simple pixel doubling and bicubic interpolation.
simply enlarging the first input frame.

The complexity of the MSA algorithm is determined by the source frame size, the number of source frames, and the super-resolution factor. Although the routine is best implemented by scaling, warping and taking the median of one small slice at a time to reduce memory requirements, analysis is simplified by treating it as a series of full image scale and warp operations followed by a median operation. For source images of size $n$ by $n$, with $m$ input frames and an enhancement factor of $r$, the MSA routine runs in $O(n^2 r^2 m) + O(n^2 r^2 m^2) = O(n^2 r^2 m^2)$ time, if a selection sort is used for the median operation. The selection sort does not scale well, but for the small values of $m$ normally encountered, it beats out sort algorithms with higher fixed costs, especially since only the first $m/2 + 1$ values need to be sorted.

The initial estimate algorithm is easily applied to colour images by running the MSA procedure separately for each channel. The super-resolution enhancement algorithm will later account for inter-colour dependencies.

### 4.2 Greyscale enhancement

Super-resolution is an inverse problem, with a goal of reversing the effects of motion, atmospheric and camera blur, decimation, and noise on a series of input images [10]. Image registration is never perfect, so inaccuracies in frame alignment must also be considered. The super-resolution algorithm presented in [1] treats image enhancement as an iterative minimization process, using steepest descent to converge to a solution. Minimization begins with the initial estimate described in the previous section.

Two minimization terms are described in [1]. The data fusion or data fidelity term combines values from the source frames in an attempt to remove camera blur. Meanwhile, the regularization term helps the minimization procedure converge to a stable solution in the presence of outliers from noise or inaccurate registrations. The following subsections describe each term, then present some greyscale results.

#### 4.2.1 Data fidelity term

The data fidelity term attempts to minimize the sum of the absolute differences between each source frame and the current estimate, after warping the estimate backwards by that frame’s registration, blurring it with the camera’s point spread function, and decimating it by the super-resolution factor,
Figure 15: Block diagram of the data fidelity term. The results for each source frame are summed to obtain the gradient, which is then multiplied by the stepsize and subtracted from the estimate.

The minimization is performed using steepest descent, using the following gradient term:

\[
\hat{X}_{n+1} = \hat{X}_n - \beta \sum_{k=1}^{N} F_k^T H^T D^T \text{sign}(DHF_k \hat{X}_n - \hat{Y}_k)
\]

\(\hat{X}_n\) is the current super-resolution estimate, \(\beta\) is a stepsize controlling the rate and accuracy of convergence, \(N\) is the number of low-resolution source frames, and \(\hat{Y}_k\) are the low-resolution frames. \(D\) and \(H\) represent decimation and camera point spread operators, respectively; \(D^T\) is the interpolation operator, and \(H^T\) is simply the reflection of \(H\). \(F_k\) warps the high-resolution estimate toward the source frame, and \(F_k^T\) warps it back to the reference.

For each source frame, the source frame is subtracted from the back-warped, blurred, decimated high-resolution estimate. The sign of the difference is then taken, giving values in the set \([-1, 0, +1]\). These values are interpolated back up, blurred by the reflection of the original blur kernel, and warped into place on the high-resolution grid. The results from each frame are then summed, multiplied by the stepsize, and subtracted from the current estimate to carry out one iteration of the steepest descent process. Figure 15 shows a graphical representation of the algorithm.

The decimation operator used in the fidelity term retains every \(r^{th}\) row and column, starting
with the first, and discards the rest. Similarly, the interpolation operator inserts \( r - 1 \) zeroes after each row and column. These operators are called “dumb decimate” and “dumb interpolate” in the code to indicate that they perform no inter-pixel interpolation. Bilinear interpolation is used for the warping operation, but little quality would be lost by using nearest neighbour interpolation instead.

The data fidelity term should have the following two effects:

- The value at each pixel location is the median value of the available values after interpolating and scaling each source frame. \(^1\)

- The image has been deblurred.

However, the camera’s point spread function (PSF) must be known or guessed in order for the deblurring to work, as discussed in Section 4.2.3. It should be noted that for symmetric point spreads, such as Gaussian kernels, the reflected PSF is identical to the original.

When only translational motion is captured by the image registration algorithm, the data fidelity term can be optimized to avoid iterating through each source frame in every descent step. Instead, the term operates as a deblurring process. The current estimate is blurred and multiplied element-by-element by a mask matrix, \( \Phi \), containing the square root of the number of measurements at each pixel. The initial estimate, again multiplied by \( \Phi \), is then subtracted. The sign of the difference is taken, multiplied by \( \Phi \), and blurred by the reflection of the blur kernel to obtain the steepest descent gradient. Figure 17 on page 27 shows that restricting registrations to translational motion tends to reduce output quality significantly; consequently, this optimization was not used in the final implementation.

### 4.2.2 Regularization term

The regularization term presented in [1] attempts to minimize the absolute difference between the estimate and several shifted versions of the estimate. The authors call this regularization function “bilateral total variation (TV)”. The gradient from the previous section is augmented with a regularization term:

\[
\hat{X}_{n+1} = \hat{X}_n - \beta \left\{ \sum_{k=1}^{N} F_k^T H^T D^T \text{sign}(DHF_k \hat{X}_n - \hat{Y}_k) \right\}
\]

\(^1\)The initial estimate is already very nearly in this condition.
\[ + \lambda \sum_{l=-P}^{P} \sum_{m=0}^{P} \alpha^{m+|l|}[I - S_{y}^{-m}S_{x}^{-l}] \text{sign}(\hat{X}_{n} - S_{x}^{l}S_{y}^{m}\hat{X}_{n}) \] 

Here, \( \lambda \) is a regularization weight determining the importance of the bilateral TV term as compared to the data fidelity term. The spatial decay constant, \( \alpha \), ranges from 0 to 1 and controls the relevance of more distant pixels. Higher values give blurrier results, requiring pixels to be more similar to their distant neighbours. The kernel radius, \( P \), is normally fixed at 2. \( S_{x}^{l} \) and \( S_{y}^{m} \) are operators that shift the image to the right by \( l \) pixels and down by \( m \) pixels, respectively. In pseudocode, the algorithm is simple:

\[
\text{for } l = -P \text{ through } P \\
\quad \text{for } m = 0 \text{ through } P \\
\quad \quad \text{if } (l + m > 0) \\
\quad \quad \quad \text{temp} = \text{sign(estimate} - \text{shift(estimate, m, l))}; \\
\quad \quad \quad \text{reg} += \alpha^{m + \text{abs}(l)} \times (\text{temp} - \text{shift(temp, -1, -m)}); \\
\quad \quad \end{for} \\
\end{for}

The function \text{shift(image, y, x)} shifts the image to the right by \( x \) pixels and down by \( y \) pixels.

4.2.3 Results and complexity

To obtain the best possible output from the super-resolution algorithm, it is necessary to estimate the camera’s point spread function (PSF) and tune the regularization parameters to smooth out noise. The PSF is usually be assumed to be a Gaussian blur kernel. If the estimated kernel size is off by even 20%, the output quality is noticeably reduced. It is usually worth the effort to begin with a few trial runs using various blur kernels between 3x3 and 13x13, using a sort of manual binary search to find the best estimate. Regularization parameters can then be tweaked.

Figure 16 shows the textbook images from Figure 14 after 600 iterations of super-resolution processing with a super-resolution factor of two, a stepsize of 0.002, and assuming the camera’s point spread function is a 7x7 Gaussian blur kernel with standard deviation of 5.0. In Figure 16(d), the regularization term was enabled with weight 0.025, spatial decay constant 0.6, and kernel radius 2. It can be seen that the data fidelity term counteracts camera blur and the regularization term
Figure 16: Super-resolution enhancement of the textbook images.
smooths out the background while leaving the sharp edges largely untouched.

(a) Super-resolution enhancement (affine motion)

(b) Super-resolution enhancement (translational motion)

Figure 17: Affine and translational super-resolution enhancement of the textbook images. The text is not as readable when translational motion is assumed.

Figure 17 compares the effects of the affine and translational motion models. In Figure 17(b), only translational motion was registered, and the results are not as readable.

Figure 18 presents a more contrived example. A 348x348 image was shifted down and to the right by zero to three pixels, then blurred with a 9x9 Gaussian kernel with standard deviation of 7.0, and downsampled by a factor of four using bicubic interpolation, producing a series of 16 images. The original image was then restored with 600 iterations of the super-resolution algorithm, using a regularization weight of 0.025, spatial decay constant 0.6, and regularization kernel radius 2. This illustrates the abilities of the algorithm when presented with perfect image registrations. The reconstruction error beats that of bicubic interpolation by approximately 4 dB, or more than two times.

The experiment was repeated with errors typical of real-world sequences. Figure 18(d) shows the effects of reconstructing the image when only 11 out of 16 frames are present. Some details are blurred, but the algorithm is mostly able to recover and PSNR remains high. In Figure 18(e), four frame registrations are corrupted by adding or subtracting 0.5, 1.0, 3.75, or 5.75 pixels. The median operation performed at the initial estimate stage is able to avoid the ghosting effects that would have occurred if an average were taken instead, and with 600 iterations the enhancement algorithm is able to keep the PSNR up. Figure 19 graphs the improvement in PSNR as more iterations are performed. The improvement appears to have the general form $y(x) = C(1 - e^{-\lambda x})$: there are
Figure 18: Results from super-resolution enhancement of a constructed sequence of 16 images.
diminishing returns after about 300 iterations.

Finally, Figure 18(f) shows that the algorithm is still able to outperform bicubic interpolation when only 25% of the required translations are available. Figure 20 shows how the PSNR improves as more source frames are added. Again, the function seems to have the form \( y(x) = C(1 - e^{-\lambda x}) \).

It should be noted that this result applies to contrived data-sets with perfect sub-pixel registrations. For real-world data, some frames may contribute redundant information, resulting in no quality gain. Additionally, adding a blurry or inaccurately registered frame could actually decrease quality.

![Graph showing PSNR as a function of super-resolution enhancement iterations for the reconstruction in Figure 18(e), with 11 source frames and some registration errors.](image)

**Figure 19:** Graph showing PSNR as a function of super-resolution enhancement iterations for the reconstruction in Figure 18(e), with 11 source frames and some registration errors.

The complexity of one iteration of the data fidelity term is determined by the source frame size, the number of source frames, the super-resolution factor, and the blur kernel’s size and separability. Like the initial estimate algorithm, its complexity is essentially linear in the number of pixels processed. However, it is much slower, requiring two warps and two blurs per source frame for each iteration.

The complexity of the regularization term is also linear in the number of pixels processed, but it is quicker than the data fidelity term, requiring mainly pixel-by-pixel arithmetic operations and a couple of block image shifts.

The enhancement algorithm provides noticeable benefits over single-image reconstruction when applied to greyscale images. For RGB colour images, some changes are desirable to account for cross-colour correlation. The next section describes these changes.
Figure 20: Graph showing PSNR as a function of number of source frames for the reconstruction in Figure 18(c). For this test, only 300 iterations were performed.

4.3 Colour enhancement

The colour enhancement algorithm presented in [11] treats the image in a few different ways:

- as three independent channels, red, green, and blue, for the data fidelity term
- as a Y’CbCr image for separate regularization of the luminance (Y’) and chrominance (Cb and Cr) channels
- as three dependent channels, to account for inter-colour edge orientation dependencies

The algorithm again consists of a steepest descent minimization process. Minimization terms are derived with respect to red, green, and blue, producing three gradients, one for each channel. The data fidelity term on each channel remains exactly the same as in the greyscale algorithm, while the bilateral TV regularization term is modified slightly. Two other terms are added to smooth the chrominance channel and to account for inter-colour orientation dependencies. The modified and new terms are explained in the following sections.

4.3.1 Spatial luminance regularization

The luminance channel is regularized using the bilateral TV term from Section 4.2.2. Each pixel is first converted to an intensity level, using the NTSC constants for R’G’B’ to Y’ conversion. The regularization calculation proceeds as before, but the result is multiplied by the current channel’s NTSC conversion constant before being added to the gradient.
4.3.2 Spatial chrominance regularization

Humans perceive the intensity of an image more adeptly than its colours, so Farsiu et al [11] propose the use of a more computationally efficient regularization term for the chrominance channel. They first determine the magnitude of the high-frequency components of the current estimate’s chrominance channels, Cb and Cr:

\[ \| \Gamma X_{Cb} \|_2 + \| \Gamma X_{Cr} \|_2 \]

The operator \( \Gamma \) is a high-pass filter\(^2\). These components are then minimized using steepest descent minimization. The derivatives of the chrominance channels are calculated with respect to red, green, and blue, as shown in Appendix A.2.1. The derivatives are convolved with a high-pass filter, then convolved again, to obtain the gradient. This has a smoothing effect on the colours of the image, noticeable after three hundred iterations or so, enhancing the stability of the iterative enhancement.

4.3.3 Inter-colour orientation dependence

Along edges, the assumptions of some demosaicing algorithms break down, producing colour artifacts. Artifacts can also be produced by undersampling; examples of both types of artifacts are shown in Figures 10(f) and 11(b) on page 17. The orientation dependence term introduced in [11] penalizes these artifacts by requiring that edges in the red, green, and blue channels have similar locations and directions. The term minimizes the norm of the vector product of neighbouring pixels, where each pixel is a three-element vector of red, green, and blue values. The term itself is rather long and is left to Appendix A.2.2. It shows its greatest effect during the first hundred iterations, allowing the algorithm to remove edge artifacts early on in the enhancement process.

4.3.4 Results and complexity

In this project, full-colour frames, having been demosaiced by camera electronics or in software, are converted to grey and registered. The full-colour frames are then used to build an initial estimate,

\[^2\text{In this project, a Laplacian kernel was used:}\]

\[
\begin{bmatrix}
0 & -1 & 0 \\
-1 & 4 & -1 \\
0 & -1 & 0
\end{bmatrix}
\]
and again in the data fidelity term to deblur each channel of the image. According to [11] the enhancement algorithm should be able to demosaic images on the fly, when colour-filtered images are passed to the initial estimate routine (or perhaps to the data fidelity term). Experiments have shown that this codebase does not work with colour-filtered images, possibly due to improper implementation. Nonetheless, positive results have been achieved using demosaiced input images.

Figure 21 shows a super-resolution enhancement of a series of five colour 320x340 pictures of a bookshelf. The series was acquired from a Fuji DX-10 digital camera with its composite A/V output connected to capture card, resulting in increased colour artifacts. The colour algorithm was applied for 600 iterations with a super-resolution factor of two, a stepsize of 0.002, and assuming the camera’s point spread function is a 9x9 Gaussian blur kernel with standard deviation of 7.0. The spatial luminance term was enabled with weight 0.1, spatial decay constant 0.6, and kernel radius 2. Spatial chrominance and orientation dependence weights were set to 25.0 and 5.0, respectively. The enhancement algorithm was able to correct the colour artifacts and improve the level of detail in the image.

Figure 22(a) shows part of the lower shelf from the bookshelf series. The initial estimate from five frames, in Figure 22(b), softens some of the colour artifacts at edges, but they are still visible. Figure 22(c) shows the effect of 600 iterations of the data fidelity term. The image has been sharpened, but looks poor. In Figure 22(d) the spatial luminance regularization term has been enabled, removing the artifacts introduced by the data fidelity term. However, “rainbow” effects are visible on the wall in the upper-right corner. These are removed in Figure 22(e) by activating the spatial chrominance term. Finally, in Figure 22(f), the orientation dependence term is enabled. This term corrects the red haze over the black edge of the leftmost book, but it introduces other colour artifacts on the vertically stacked books to the right. The orientation dependence term is the most problematic of the four, often creating very bright or very dark lines on existing edges.

Figure 23 shows an image of a car and its license plate taken with a Nokia 3220 cell phone camera (640x480, best quality setting). Three images were combined, but the super-resolution reconstruction does nothing to enhance the readability of the plate. In fact, it smears some of the right-hand digits. The algorithm is handicapped by the fact that the text on Ontario license plates is blue, a colour that is poorly represented in a colour-filtered digital camera. However, experiments have shown similarly poor results with other text colours. When presented with fine
Figure 21: Colour bookshelf source frame and super-resolution enhancement.
Figure 22: A region from the bookshelf image, showing the cumulative effect of each term.
print, the regularization term treats the character edges as noise and smooths them out. Even without regularization, the data fidelity term cannot sharpen the text enough to make it legible.

For this example the colour algorithm was applied for 600 iterations with a super-resolution factor of two, a stepsize of 0.002, and assuming the camera’s point spread function is a 7x7 Gaussian blur kernel with standard deviation of 5.0. The spatial luminance term was enabled with weight 0.04, spatial decay constant 0.6, and kernel radius 2. Spatial chrominance and orientation dependence weights were set to 25.0 and 1.0, respectively.

Figure 24 shows a cosmetic super-resolution enhancement of three images taken with a Nokia 3220 cell phone camera (640x480, best quality setting). The algorithm is able to deblur the elephant, but the face in the small picture at right is left mostly unchanged. The algorithm was iterated 600 times with a super-resolution factor of two, a stepsize of 0.002, and assuming the camera’s PSF is a 7x7 Gaussian blur kernel with standard deviation of 5.0. The spatial luminance term was enabled with weight 0.03, spatial decay constant 0.6, and kernel radius 2. Spatial chrominance and orientation dependence weights were set to 25.0 and 2.5, respectively.

Aside from the necessity of processing three channels instead of one, the complexity of the data fidelity and spatial luminance regularization terms remains as discussed in Section 4.2.3. The spatial chrominance term requires a few constant multiplications per pixel, and two convolutions with an inseparable 3x3 kernel. The orientation dependence term performs subtractions and multiplications between several shifted versions of the image, but complexity is still linear in the number of pixels processed. Overall, the colour enhancement algorithm’s complexity remains linear with respect to the number of pixels processed and the number of source frames, with no quadratic terms.
(a) Original image after bicubic scaling  

(b) Super-resolution enhancement from three frames

**Figure 23:** Unsuccessful license plate reconstruction from a cell phone camera.

---

(a) Original image, bicubically scaled to 200%  

(b) Super-resolution enhancement, factor of 2

**Figure 24:** Cell phone camera reconstruction from three source frames.
CHAPTER V

TECHNICAL ENVIRONMENT

During the course of the project, nearly 4000 lines of GNU Octave code and 5400 lines of C++ code were produced and released under an open source license. A description of the contents of the open source package would be outside the scope of this report, and is left to Appendix B. However, some notes about the completed work and design decisions are relevant.

5.1 GNU Octave implementation

GNU Octave is an open source command-line mathematical prototyping environment. It is largely compatible with MATLAB in capability, syntax, and body of available library functions.

Building on the octave-forge function library, routines were written for loading, saving, and displaying colour and greyscale images. In addition, the following routines were implemented as a foundation for the image registration, demosaicing, and super-resolution algorithms, and to support experimental work:

- conversion from colour to grey, according to the ITU standard
- image decimation and interpolation by zero-filling
- image scaling by bilinear interpolation
- image shifting
- Gaussian blur kernel generation
- image spatial gradient calculation
- addition of salt and pepper noise to an image
- calculation of mean square error (MSE) and peak signal to noise ratio (PSNR)
- affine warp by bilinear interpolation
- inversion and composition of affine motion models
- Bayer colour-filtering of full-colour images

Unit tests exist for these low-level routines, with the intention that test oracles could be reused for the C++ implementation. The following high-level routines were implemented:

- affine and translational image registration using the Least-Mean-Squares algorithm
• creation of initial estimates for colour and greyscale image sequences
• greyscale and colour super-resolution

In addition, interfaces were created to process images using the following external tools:

• the Motion2D image registration library
• the Generalized Pei-Tam Method (GPTM) colour demosaicing routine extracted from the Linux USB QuickCam driver

As compared with the C++ implementation, the Octave initial estimate algorithm produces slightly offset, blurrier results due to the way in which image scaling is performed. Subsequent enhancement steps correct the initial estimate output toward the interpretation of the C++ implementation. Since the C++ initial estimate routine runs about 85 times faster than the equivalent in Octave, correcting the Octave routine was deemed unnecessary.

The Octave implementation was created over a period of four months, as algorithms were chosen and understood. In terms of memory and time requirements, its performance is quite poor, often requiring an overnight run for a colour super-resolution experiment. Because each image channel is represented as a 64-bit floating point matrix to allow manipulation by Octave primitives, memory requirements can easily balloon. Many implicit copies are created during the initial estimate and enhancement phases, so over a gigabyte of memory may be required when enhancing large images. The C++ implementation was created to resolve these issues and to allow for easier experimentation with algorithm parameters.

5.2 C++ implementation

The goal of the C++ implementation was to facilitate a reasonably efficient procedure for repeatedly running the super-resolution algorithm against a data set with varying parameters. A list of images is dropped into a .ini-style file, along with coordinates for a region of interest (usually obtained from the crop tool in the GIMP), the algorithm parameters, and a location to write output files. A command-line driver is then used to process a series of these .ini files.

5.2.1 Package structure

The implementation is structured as the following four packages:

• a library containing the initial estimate and super-resolution algorithms, and an interface to the Motion2D image registration package
• a library for parsing .ini files, donated by my employer, Canadian Bank Note Co. Ltd.
• a class for loading images using the ImageMagick library; this allows the algorithm library to be used independently without introducing an ImageMagick dependency into client applications that do not need it
• a simple command-line driver, linked against the above three libraries

The super-resolution library contains the following support functions:
• affine model inversion and composition
• bilinear interpolation
• affine warp, image scaling, and combined scaling and warping by bilinear interpolation
• generation of Gaussian blur kernels
• bidirectional communication with a subprocess
• two-dimensional convolution by separable and non-separable kernels
• zero-fill image scaling

Figure 25: Class diagram for the algorithm package.

Figure 25 shows a class diagram for the algorithm package. The general flow is to load a set of Images, fill in a set of AffineModels using the Registrar, build a MedianShiftAdd to create an
initial high-resolution estimate, and pass the lot to a new ColourSR for enhancement.

5.2.2 Design decisions

Many simple operations exist on the Image class, such as pixel access, element-by-element addition, zero-fill scaling, and greyscale conversion. For ease of implementation, each pixel is a 32-bit floating point number; on modern processors, floating point performance is reasonable enough to allow this. Multiple-channel images are stored with their channels interleaved, because the algorithms often access the channels together. It is hoped that channel interleaving increases the likelihood of cache hits.

Polymorphism is used where appropriate, but only when the overhead is acceptable. In particular, the overhead of a virtual function call is minimal compared with the execution time of a convolution operation, so separable and non-separable convolution kernels have convolve() methods inherited polymorphically from their parent.

As a library, it should be easy to reuse the super-resolution functionality, as well as the support routines. These routines, such as convolution, are not tied to super-resolution, although the algorithm as a whole could potentially be made more efficient if they were rewritten to be less general. For example, the convolution operation could be hard-coded for three-channel images and symmetric blur kernels.

5.2.3 The porting experience

The C++ implementation was created over a period of approximately three weeks, during which time approximately 3000 lines of production code and 1500 lines of unit tests were created. It was hoped that a general open source image processing library could be tapped to provide services such as convolution and bilinear interpolation, but a suitable library was not found which would also allow easy pixel-by-pixel arithmetic operations for the super-resolution step. Consequently these routines had to be implemented from scratch, using up a few days of development time and necessitating a few hundred extra lines of test cases.

Early on, the CxxTest unit testing framework was chosen because of its ease of test creation, automatic test registration, and comprehensive set of assert functions [12]. A few classes were instrumented with printing and equality operators, but otherwise the library code was undisturbed.

The existence of Octave test oracles facilitated the testing of the lower-level routines, but many
new tests were needed to cover infrequent but legal inputs at the boundaries of expected values. For example, it was necessary to ensure that the convolution operator did not crash when presented with an image smaller than the convolution kernel, and that the median operator could properly handle sequences of lengths one and two. Consequently, there was a lot of tedious entering of oracle and test input matrices. For some routines, significantly more time was spent debugging the test cases than debugging the implementation. However, the unit tests quickly proved themselves invaluable, catching scores of careless errors shortly after they were committed.

Unit tests were not created for higher-level routines, such as the super-resolution algorithm itself, because it was considered too difficult to test a converging iterative algorithm. Quality suffered: for several days, the orientation dependence term (see Section 4.3.3) did not add its results to the image delta, and the spatial luminance term added when it should have subtracted.

The effort to rewrite the algorithms in C++ was entirely worth it for the memory usage and execution speed improvements alone. Four 1792x1200 greyscale images can be combined into an initial estimate in approximately 3 seconds, using 76 MB RAM, on an AMD Opteron 240 at 1400 MHz. The same operation would peak at 1564 MB RAM and 255 seconds in Octave, nearly 85 times slower. Part of the improvement may result from processing images in smaller slices to avoid bringing multiple full-sized scaled-up images into memory at once, and from scaling and warping in one interpolation step, but it seems most of it comes from switching away from an interpreted language with many implicit copies.

The colour super-resolution algorithm itself experienced more modest speed gains, in the range of ten to fifteen times, with memory usage not much larger than enough to store five or six temporary copies of the full-size estimate. On an Intel Pentium 4 at 3400 MHz, it takes about four and a half minutes to run 100 enhancement iterations of a full 320x340 colour image, with five source frames and a super-resolution factor of two. This is fast enough to support exploration of the algorithm’s parameters. Combined with the ease of experimentation provided by the parameter description file parser, the performance gains of the C++ implementation have made life significantly easier.
CHAPTER VI

CONCLUSION

Over the course of seven months, a number of image processing algorithms have been implemented. An image registration algorithm was implemented, and it worked reasonably well, but better results were achieved with a pre-existing open source library. Two colour demosaicing algorithms were implemented and compared with a pre-existing open source routine, which was found to give comparable results.

The success of the project lies in the actual super-resolution implementation. Using real-life and contrived image sequences, the super-resolution algorithm was able to reconstruct details, enhance text readability, and improve visual quality. To support the project and further research, a comfortable .ini-file interface was created for experimenting with algorithm parameters. This interface and the reasonably fast C++ implementation behind it allow for the rapid execution and refinement of new reconstructions.¹

Several areas for further research have been identified. First, a method to evaluate super-resolution output would be extremely helpful when refining algorithm parameters. For contrived data sets, the mean squared error or peak signal to noise ratio can be calculated between the input and the output, but this technique is not applicable to real-world reconstructions. Additionally, stopping conditions for the iterative enhancement could be developed to allow the algorithm to stop executing when it has converged.

Secondly, it would be desirable to specialize the regularization term for various purposes, such as text enhancement. As shown in Figure 23, the regularization terms used in this project tend to smear text rather than sharpening it. Additionally, even with more simplistic techniques, text may be altered during the enhancement process. For instance, it was found in [13] that a ‘B’ on a known license plate may appear as a ‘D’ after super-resolution enhancement. More research is needed to correct these issues.

Blind deconvolution algorithms, which can deblur single images by determining the camera’s

¹ Rather, they did, until the author’s Athlon was destroyed in a tragic super-resolution accident.
point spread function, seem like natural candidates for a pre-processing stage, to determine the PSF, or for post-processing, to further enhance a super-resolution estimate. However, as noted in [11], there has not been much research toward the effects of integrating deconvolution and multi-frame image reconstruction, although [14] does propose a regularization term for dealing with inaccurate PSF estimates. Further, it would be useful to compare the results of single-frame blind deconvolution with multi-frame super-resolution, to determine whether the super-resolution algorithm provides an improvement in text readability or visual quality.

Finally, the use of a global parametric motion model restricts the applicability of the implementation to scenes with more than one motion component. The use of non-parametric models, such as the motion vectors used in MPEG-4 encoding, should be explored. As long as the model is invertible and composable, and provides sub-pixel accuracy, there is nothing in the super-resolution algorithm to preclude its use.

The algorithms implemented in this project show promise for cosmetic image enhancement. Although they are only sometimes successful with textual enhancement, their implementation provides a vehicle for further research. With some thought, a process for enhancing text could be developed. Super-resolution will have wide application in the forensic, commercial, scientific and entertainment domains, even before its techniques have reached maturity.
APPENDIX A

MATHEMATICAL DERIVATIONS

A.1 Affine motion model

A.1.1 Inversion

The affine motion model relates original to warped coordinates by the following two equations:

\[ x' = x + a_1 + a_2x + a_3y \]  
\[ y' = y + a_4 + a_5x + a_6y \]  

To create a set of parameters that can reverse the effect of an affine warp, it is necessary to solve the coordinate transformation equations for \( x \) and \( y \), and isolate the coefficients on \( x' \) and \( y' \):

\[ x + a_2x + a_3y = x' - a_1 \]  
\[ y + a_5x + a_6y = y' - a_4 \]  

Solving for \( y \):

\[ y' - a_4 - a_5x = y(1 + a_6) \]
\[ y = \frac{y' - a_4 - a_5x}{1 + a_6} \]  

Substituting (5) into (3):

\[ x + a_2x + a_3\left(\frac{y' - a_4 - a_5x}{1 + a_6}\right) = x' - a_1 \]
\[ x(1 + a_2) + a_3\frac{y'}{1 + a_6} - a_3\frac{a_4}{1 + a_6} - \frac{a_3}{1 + a_6}a_5x = x' - a_1 \]
\[ x(1 + a_2 - \frac{a_3}{1 + a_6}a_5) = x' - a_1 - \frac{a_3}{1 + a_6}y' + \frac{a_3}{1 + a_6}a_4 \]
\[ x = \frac{1}{1 + a_2 - \frac{a_3}{1 + a_6}a_5}x' + \frac{1}{1 + a_2 - \frac{a_3}{1 + a_6}a_5}(-a_1 + \frac{a_3a_4}{1 + a_6}) - \frac{1}{1 + a_2 - \frac{a_3}{1 + a_6}a_5}\frac{a_3}{1 + a_6}y' \]
\[ x = (\frac{1}{1 + a_2 - \frac{a_3}{1 + a_6}a_5} - 1)x' + x' + \frac{1}{1 + a_2 - \frac{a_3}{1 + a_6}a_5}(-a_1 + \frac{a_3a_4}{1 + a_6}) - \frac{1}{1 + a_2 - \frac{a_3}{1 + a_6}a_5}\frac{a_3}{1 + a_6}y' \]  

44
From (6), the first three parameters can be extracted:

\[ a_1' = \frac{1}{1 + a_2 + \frac{a_3}{1+a_6} a_5}(-a_1 + \frac{a_3 a_4}{1 + a_6}) \]
\[ a_2' = \frac{1}{1 + a_2 - \frac{a_3}{1+a_6} a_5} - 1 \]
\[ a_3' = -\frac{1}{1 + a_2 - \frac{a_3}{1+a_6} a_5} \frac{a_3}{1 + a_6} \]

A similarly tedious procedure is used to find the last three parameters:

\[ a_4' = -\frac{1}{1 + a_6 + \frac{a_3}{1+a_6} a_5}(-a_4 + \frac{a_3 a_5}{1 + a_2}) \]
\[ a_5' = -\frac{1}{1 + a_6 - \frac{a_3}{1+a_2} a_5} \frac{a_5}{1 + a_2} - 1 \]
\[ a_6' = \frac{1}{1 + a_6 - \frac{a_3}{1+a_2} a_5} \]

### A.1.2 Composition

It is possible to combine the cumulative effect of two affine warp operations into one set of parameters. This is more efficient than performing several successive warp operations on a real image. To solve for the cumulative parameters \( a_1, \ldots, a_6 \), the effect of two successive warps, \( a_{11}, \ldots, a_{16} \) and \( a_{21}, \ldots, a_{26} \), is first determined:

\[ x' = x + a_{11} + a_{12} x + a_{13} y \]
\[ y' = y + a_{14} + a_{15} x + a_{16} y \]
\[ x'' = x' + a_{21} + a_{22} x' + a_{23} y' \]
\[ = x + a_{11} + a_{12} x + a_{13} y + a_{21} + a_{22}(x + a_{11} + a_{12} x + a_{13} y) \]
\[ + a_{23}(y + a_{14} + a_{15} x + a_{16} y) \]
\[ x'' = x + (a_{11} + a_{21}) + x(a_{12} + a_{12} a_{22} + a_{22} + a_{15} a_{23}) \]
\[ + a_{11} a_{22} + a_{14} a_{23} + y(a_{13} + a_{13} a_{22} + a_{23} + a_{11} a_{23}) \] (7)

From (7), the first three parameters can be determined:

\[ a_1 = a_{11} + a_{21} + a_{11} a_{22} + a_{14} a_{23} \]
\[ a_2 = a_{12} + a_{22} + a_{12} a_{22} + a_{15} a_{23} \]
\[ a_3 = a_{13} + a_{23} + a_{13} a_{22} + a_{16} a_{23} \]
Similarly, after expanding the equation for $y''$:

\begin{align*}
a_4 &= a_{14} + a_{24} + a_{11}a_{25} + a_{14}a_{26} \\
a_5 &= a_{15} + a_{25} + a_{12}a_{25} + a_{15}a_{26} \\
a_6 &= a_{16} + a_{26} + a_{13}a_{25} + a_{16}a_{26}
\end{align*}

### A.2 Colour super-resolution terms

#### A.2.1 Spatial chrominance regularization term

Farsiu et al [11] define the spatial chrominance term to be minimized as

$$R(X) = \|\Gamma X_{C1}\|_2 + \|\Gamma X_{C2}\|_2$$

where $X$ is the true image, $\Gamma$ is a high-pass filter, and $X_{Cb}$ and $X_{Cr}$ are the Cb and Cr layers, respectively, in the Y’CbCr colour representation. The same authors provide a term with constants to extract Cb and Cr from an R’G’B’ image in [15]:

$$R(X) = \|\Gamma(-0.169X_R - 0.331X_G + 0.5X_B)\|_2 + \|\Gamma(0.5X_R - 0.419X_G - 0.081X_B)\|_2$$

The partial derivative with respect to the red, green, and blue estimates, respectively, can be calculated as follows:

\begin{align*}
\nabla \hat{X}_R &= \Gamma^T \Gamma((-0.169)^2X_R + (-0.169)(-0.331)X_G + (-0.169)(0.5)X_B \\
&\quad + (0.5)^2X_R + (0.5)(-0.419)X_G + (0.5)(-0.081)X_B) \\
&= 0.27846X_R - 0.15328X_G - 0.12515X_B \\
\nabla \hat{X}_G &= \Gamma^T \Gamma(-0.15328X_R + 0.28467X_G - 0.13147X_B) \\
\nabla \hat{X}_B &= \Gamma^T \Gamma(-0.12515X_R - 0.13147X_G + 0.25666X_B)
\end{align*}

$\Gamma^T$ is the transpose of the high-pass filter. For a Laplacian kernel, this is identical to $\Gamma$.

#### A.2.2 Orientation dependence regularization term

Farsiu et al [11] propose to minimize the norm of the vector product of neighbouring pixels, below:

$$R(X) = \sum_{l=-1}^{1} \sum_{m=0}^{1} \left\| X_G \odot S_{x}^l S_{y}^m X_B - X_B \odot S_{x}^l S_{y}^m X_G \right\|_2$$
Here, $\odot$ denotes element-by-element matrix multiplication. $S^l_x$ and $S^m_y$ are operators that shift the image to the right by $l$ pixels and down by $m$ pixels, respectively. The red, green, and blue gradients of this term are as follows:

\[
\nabla \hat{X}_R = 2 \sum_{l=-1}^{1} \sum_{m=0}^{1} \left[ (S^l_x S^m_y X_L - S^{-l}_x S^{-m}_y X_L)(S^l_x S^m_y (X_R) - S^l_x S^m_y (X_R) X_R) \\
+ (S^l_x S^m_y X_B - S^{-l}_x S^{-m}_y X_B)(S^l_x S^m_y (X_B) - S^l_x S^m_y (X_B) X_B) \right]
\]

\[
\nabla \hat{X}_G = 2 \sum_{l=-1}^{1} \sum_{m=0}^{1} \left[ (S^l_x S^m_y X_R - S^{-l}_x S^{-m}_y X_R)(S^l_x S^m_y (X_G) - S^l_x S^m_y (X_G) X_R) \\
+ (S^l_x S^m_y X_B - S^{-l}_x S^{-m}_y X_B)(S^l_x S^m_y (X_B) - S^l_x S^m_y (X_B) X_B) \right]
\]

\[
\nabla \hat{X}_B = 2 \sum_{l=-1}^{1} \sum_{m=0}^{1} \left[ (S^l_x S^m_y X_R - S^{-l}_x S^{-m}_y X_R)(S^l_x S^m_y (X_B) - S^l_x S^m_y (X_B) X_R) \\
+ (S^l_x S^m_y X_G - S^{-l}_x S^{-m}_y X_G)(S^l_x S^m_y (X_G) - S^l_x S^m_y (X_G) X_B) \right]
\]
APPENDIX B

SOFTWARE PACKAGE OVERVIEW

The code and data files produced during the course of the project have been made available at http://www.engsoc.org/~michael/ . The package files are released under the GNU General Public License, with a few exceptions noted in the LICENSE file.

After extracting the package tarball, the following directories are created:

- **cpp**: the C++ implementation, described in Section B.2.
- **data**: some of the input data used to prepare this report.
- **finalreport**: the source for this report.
- **gptm-demosaic**: a user-space wrapper for demosaicing images using code from the Linux QuickCam Express driver, qc-usb. At the driver’s best quality setting, the GPTM algorithm [9] is used to demosaic raw data from the QuickCam; the code for this implementation was extracted from the driver. See feedme.sh for a usage example.
- **grabber**: a utility for grabbing a series of raw (non-demosaiced) frames from the Linux QuickCam Express driver.
- **mo2dseries**: a small command-line client of the Motion2D image registration library [6]. This client is called from the C++ and Octave implementations to align input frames. See Section B.1 for compilation instructions.
- **octave**: the GNU Octave implementation, described in Section B.3.

**B.1 mo2dseries**

The Motion2D library [6] provides an image registration API. mo2dseries is a small wrapper, reading raw images on stdin and writing inter-frame registrations to stdout.

To build mo2dseries, you must first download and compile Motion2D. Next, change to the
mo2dseries directory and edit the Makefile. Set the correct location for the MOTION2D_DIR variable, save the Makefile, and run make to build the executable.

The motion2d binary is not easy to call directly, but feedme.sh shows an example of how it can be done.

B.2 C++ Implementation

The C++ implementation provides several general image processing routines, as described in Section 5.2. More importantly, it provides a command-line application for applying the super-resolution algorithm to a series of images, parameterized through a .ini file.

The C++ implementation depends on the mo2dseries tool, which you should compile according to the previous section. It also makes use of the ImageMagick library’s C++ interface, so ensure that is installed before compiling the C++ implementation.

After compiling mo2dseries and installing ImageMagick, change to the cpp directory and run make. You can also run the CxxTest unit test suite with make test.

The usual procedure is to make a new directory, copy in cpp/cliapp/run.ini, and customize it to point to your source images. Refer to Sections 4.2.3 and 4.3.4 for hints on parameter settings. Generally you will want to begin by queueing up several runs with various blur kernel sizes, from 5x5 to 13x13. After creating a few .ini files, you can process them sequentially:

```bash
$ PATH=../path/to/mo2dseries:$PATH ../path/to/cpp/cliapp/superres *.ini
```

It is best to let the algorithm run for at least a hundred iterations, as controlled by the totaliters setting. If you are impatient you can set iterspersave to a lower value, such as 30, to get initial output sooner. Every iterspersave iterations, an image will be saved to basename-iteration.png, where basename is the path of the .ini file without the extension.

B.3 Octave Implementation

The GNU Octave directory was the original test bed where the registration, demosaicing, and enhancement algorithms were implemented. It is much slower than the C++ implementation, but contains several routines not found there. Refer to Section 5.1 for information about the capabilities and limitations of this codebase.

The Octave interface is more manual than that provided by the C++ superres executable. Images are processed by calling functions; there are a few shortcut scripts in octave/shortcuts that
can be adapted. The first step is to run `octave`, and from the `octave>` prompt, run `setloadpaths` to establish the `.m`-file search path.

Most functions have a few lines of explanatory comments near the top. A few examples of what you can accomplish are listed below:

- Load a list of colour images:
  ```octave
  frames = loadAllColour(glob("/tmp/\*\.png"));
  ```

- Load a list of colour images, converting them to grey:
  ```octave
  greys = loadAllGFC(glob("/tmp/\*\.png"));
  ```

- Display the third image in a list:
  ```octave
  display(nth(frames, 3));
  ```

- Save the third image:
  ```octave
  saveColourImage("/tmp/out.png", nth(frames, 3));
  ```

- Demosaic a list of images using Kimmel’s algorithm [8]:
  ```octave
  demos = callKimmel(frames);
  ```

- Demosaic a list of images using the `gptm-demosaic` tool:
  ```octave
  demos = callGPTM(frames);
  ```

- Demosaic one frame using bilinear interpolation:
  ```octave
  demos1 = bilinDemosaic(nth(frames, 2));
  ```

- Colour-filter one full-colour frame, for use when testing demosaicing algorithms:
  ```octave
  bay = bayerize(nth(frames, 2));
  ```

- Compute the peak signal-to-noise ratio (in dB) of the red channel of two colour frames:
  ```octave
  psnr(orig.r, bilinDemosaic(bayerize(orig)).r)
  ```

- Compute the SHA-1 checksum of a matrix:
  ```octave
  sha1 = sha1sum(orig.g);
  ```

- Add salt-and-pepper noise to a greyscale image:
  ```octave
  noisy = saltAndPepper(input, 0.02);
  ```
- Register a series of images using the algorithm from [2]:
  \[
  \text{reges} = \text{regAll}(\text{frames}, 0);
  \]

- Register a series of images using the Motion2D library:
  \[
  \text{reges} = \text{regAllMo2D}(\text{frames}, 0);
  \]

Greyscale and colour super-resolution are also implemented in the \texttt{octave} directory, but there should be no reason to use them. They are up to 100 times slower than the C++ implementation.
REFERENCES


models,” Journal of Visual Communication and Image Representation, vol. 6, no. 4, pp. 348–
365, December 1995.


Available: http://forensic.to/superresolution.htm

reconstruction with edge adaptive constraints and higher-order interpolants,” IEEE Trans.

sampled color images,” Proc. of the 2004 IS&T/SPIE 16th Annual Symposium on Electronic

52