NTIA Technical Memorandum TM-04-406

# Color Correction Matrix for Digital Still and Video Imaging Systems

**Stephen Wolf**


*technical memorandum*

U.S. DEPARTMENT OF COMMERCE · National Telecommunications and Information Administration

NTIA Technical Memorandum TM-04-406

# Color Correction Matrix for Digital Still and Video Imaging Systems

**Stephen Wolf**

**U.S. DEPARTMENT OF COMMERCE**
**Donald L. Evans, Secretary**

Michael D. Gallagher, Acting Assistant Secretary
for Communications and Information

December 2003

## DISCLAIMER

Certain commercial equipment, software packages, and materials are identified in this report to specify adequately the technical aspects of the reported results. In no case does such identification imply recommendations or endorsement by the National Telecommunications and Information Administration, nor does it imply that the material or equipment identified is the best available for this purpose.

The software described within was developed by an agency of the U.S. Government. NTIA/ITS has no objection to the use of this software for any purpose since it is not subject to copyright protection in the United States.

No warranty, expressed or implied, is made by NTIA/ITS or the U.S. Government as to the accuracy, suitability and functioning of the program and related material, nor shall the fact of distribution constitute any endorsement by the U.S. Government.

**CONTENTS**

## FIGURES

Page

## TERMS AND DEFINITIONS

**4:2:2** — A Y, Cb, Cr image sampling format where chrominance planes (Cb and Cr) are sampled horizontally at half the luminance (Y) plane's sampling rate. See ITU-R Recommendation BT.601 [1].

**Chrominance (C, $C_B$, $C_R$)** — The portion of the video signal that predominantly carries the color information (C), perhaps separated further into a blue color difference signal ($C_B$) and a red color difference signal ($C_R$).

**Codec** — Abbreviation for a coder/decoder or compressor/decompressor.

**Gain** — A multiplicative scaling factor applied by a video system to all pixels of an individual image plane (e.g., luminance, chrominance). Gain of the luminance signal is commonly known as contrast.

**Input Video** — Video before being processed or distorted by a video system (see Figure 2). Input video may also be referred to as original video.

**Institute for Telecommunication Sciences (ITS)** — The research and engineering laboratory of the National Telecommunications and Information Administration, U.S. Department of Commerce.

**Luminance (Y)** — The portion of the video signal that predominantly carries the luminance information (i.e., the black and white part of the picture).

**Moving Picture Experts Group (MPEG)** — A working group of ISO/IEC in charge of the development of standards for coded representation of digital audio and video (e.g., MPEG-1, MPEG-2, MPEG-4).

**National Television System Committee (NTSC)** — The 525-line analog color video composite system adopted by the US and most other countries (excluding Europe) [2].

**Offset or Level Offset** — An additive factor applied by a video system to all pixels of an individual image plane (e.g., luminance, chrominance). Offset of the luminance signal is commonly known as brightness.

**Original Video** — Video before being processed or distorted by a video system (see Figure 2). Original video may also be referred to as input video since this is the video input to the digital video transmission system.

**Output Video** — Video that has been processed or distorted by a video system (see Figure 2). Output video may also be referred to as processed video.

**Processed Video** — Video that has been processed or distorted by a video system (see Figure 2). Processed video may also be referred to as output video since this is the video output from the digital video transmission system.

**Rec. 601** — Abbreviation for ITU-R Recommendation BT.601 [1], a common 8-bit video sampling standard that samples the luminance (Y) channel at 13.5 MHz, and the blue and red color difference channels ($C_B$ and $C_R$) at 6.75 MHz.

**sRGB** — Abbreviation for the standard RGB color space supported by most digital cameras and photo-editing software [3].

**Scene** — A sequence of video frames.

**Spatial Registration** — The process that is used to estimate and correct for spatial shifts of the processed video sequence with respect to the original video sequence.

**Temporal Registration** — The process that is used to estimate and correct for the temporal shift (i.e., video delay) of the processed video sequence with respect to the original video sequence.

**White Balance** — The color temperature of white in a photographic or video scene, which normally varies between 2000K and 12000K. Most digital cameras have settings to adjust the white balance.

# COLOR CORRECTION MATRIX FOR DIGITAL STILL AND VIDEO IMAGING SYSTEMS

Stephen Wolf [1]

This document discusses a method for correcting inaccurate color output by digital still and video imaging systems. The method uses a known reference image together with a least-squares algorithm to estimate the optimal color channel mixing matrix that must be applied to the output images in order to correct their color inaccuracies. The techniques presented in this document will provide users of digital photography and video equipment with an automated tool for correcting color output. For instance, digital photography users currently may try to correct color distortions in their images by trial and error using photo editing software. However, these correction procedures are time consuming and subjective and do not normally allow for arbitrary mixing of the color channels. The automated color correction matrix computation presented in this document allows each color component in the corrected image (e.g., red) to be calculated as a linear summation of a DC component and all the color components (e.g., red, green, and blue) in the uncorrected image. Methods to correct non-linearities in the color response of digital imaging systems are also discussed.

Key words: calibration; camera; channel; chart; color; colorspace; component; correction; digital; matrix; non-linear; sRGB; video

## 1. INTRODUCTION AND OVERVIEW

Accurate color output can be particularly critical for some user applications. For instance, accurate representation of flesh tones might be important for proper medical diagnosis when video systems are used in telemedicine applications. Other applications where accurate color might be important include online shopping, video surveillance, and inventory documentation. Users of digital still and video cameras may experience frustration as to what to do when the final color renditions of their prints and videos do not match the original scene.

A considerable amount of effort has been expended in recent years in developing color management techniques for digital images in computer systems (see for example [4]). These techniques allow one to specify color profiles for devices (such as input, display, and output devices) that specify how to convert between native device color spaces and device independent color spaces. In theory, proper image color management should enable documents and images with embedded color profiles to be freely moved between different computers and operating systems without changing their perceived colors. In practice, even knowledgeable users may be confronted with missing or inaccurate color profiles for their devices and find it difficult to obtain accurate color output. For instance, users of digital cameras might find that bright reds take on an orange color cast. Users may try to correct their images by trial and error using photo editing software. As an alternative, this document presents a simple deterministic procedure for estimating the color correction matrix for a device or imaging system. This color correction matrix allows

[1] The author is with the Institute for Telecommunication Sciences, National Telecommunications and Information Administration, U.S. Department of Commerce, 325 Broadway, Boulder, CO 80305.

each color component in the corrected image (e.g., red) to be calculated as a linear summation of a DC component and all the color components (e.g., red, green, and blue) in the uncorrected image. Methods to correct non-linearities in the color response of digital imaging systems are also discussed. MATLAB[2] code is given for implementing the color correction procedure.

## 2. COMMON APPLICATIONS

### 2.1 Digital Cameras

Figure 1 illustrates a very common application where the user desires photographic prints that have the same colors as those originally observed. Many digital cameras produce files in the RGB color space (red, green, and blue) and popular photographic manipulation software is able to read and translate images in this color space to other color spaces. For instance, a different color space may be used internally by the photographic manipulation software, by the computer monitor used to display the images, and by the printer. Color translation errors can occur at any point in the chain, from the digital camera to the printer. Some color spaces do not have as wide a color gamut, or range of colors that can be represented, and this also contributes to color errors. What is desired is a method to calibrate or correct color errors for the system shown in Figure 1.

While Figure 1 makes the color calibration situation look fairly simple and deterministic (and hence straightforward), in reality the color rendition output by the digital camera depends upon the camera's internal settings and the color temperature of the light illuminating the scene. We have all observed the changing color temperature of ambient light from sunrise (low color temperature between 2000k and 4000k) to mid-day (up to 11000K or 12000K, depending upon elevation). Most digital cameras have multiple choices for manually setting white balance (e.g., sunny, cloudy, fluorescent, incandescent, flash) and also letting the camera's internal processing estimate the correct white balance setting (e.g., automatic mode). An error in camera white balance setting (i.e., a setting different than the actual color temperature of the ambient illumination) will produce color errors. In addition, many digital cameras do not accurately record colors even when the white balance is properly set. What is needed is a procedure for correcting color that can be easily applied for any combination of ambient light and camera white balance.

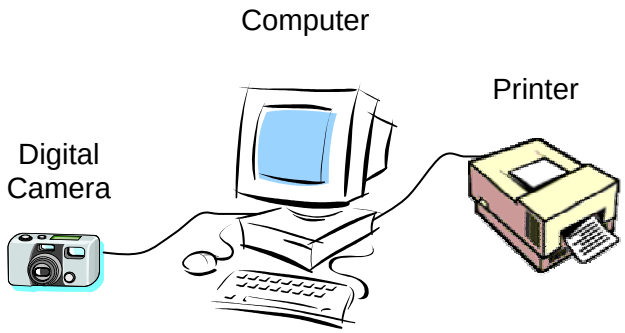


Figure 1. A digital camera system.

[2] MATLAB is a registered trademark of MathWorks, Inc.

### 2.2 Video Transmission Systems

Figure 2 illustrates another application that involves the use of a video transmission system. Here, the user would like to encode (or compress) the original video to save on transmission costs. The original or input video may be in analog form with an associated color space (e.g., NTSC). After transmitting the compressed bit-stream over a digital channel (e.g., Internet), the received bit stream must then be decoded (or uncompressed) before being presented to the user. The processed or output video may have color translation errors from the encoding and decoding processes. What is needed here is a general method for correcting the processed video to make it more closely approximate the original video. For Figure 2, one would hope that the color translation errors would be independent of video scene content so that the color correction could be calculated once for each video transmission system and then applied to all processed video scenes.

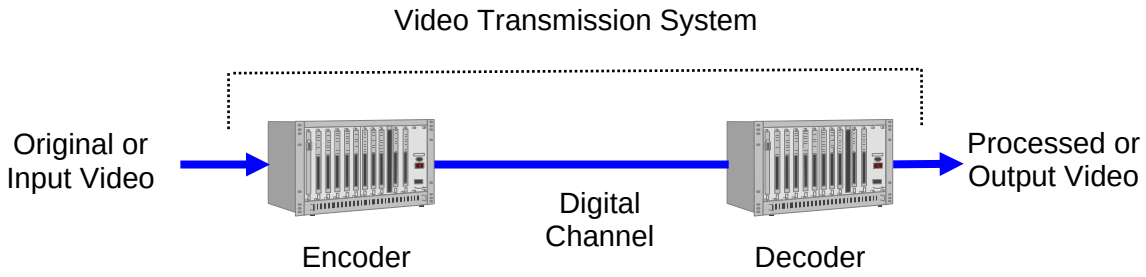


Figure 2. A video transmission system.

## 3. ESTIMATING COLOR CORRECTION

We will first present the least-squares solution for estimating the color correction matrix (section 3.1). Then we will modify this solution to make it more robust and less influenced by outliers (section 3.2). We also introduce the option of applying a non-linear monotonic transfer function before or after the color correction matrix (section 3.3). Finally, we discuss some considerations for picking the color space in which to perform the color matrix correction (section 3.4).

MATLAB code for performing the algorithms in this section is given in the appendix. Example applications of these algorithms to digital cameras and video systems will be discussed in section 4.

### 3.1 Least-squares Solution for Color Correction Matrix

This section will describe the basic matrix transformation for one color space, the RGB color space. The principles detailed here can then be extended to other color spaces. Subsequent sections will outline the application of the basic color correction matrix to the common applications shown in Figure 1 and Figure 2. The nomenclature for "original image" (i.e., the reference image) and "processed image" (i.e., the image to be color corrected) used in Figure 2 will be used to formulate the equation for the color correction matrix. The basic idea that will be described here is an automated least-squares solution for calculating the optimal color correction matrix. This color correction matrix allows each color

component in the corrected image (e.g., red) to be calculated as a linear summation of a DC component and the color components in the uncorrected image (e.g., red, green, and blue).

Let the original image **O** have *N* color samples, with each color sample being represented using red (R), green (G), and blue (B) intensities. Organize these original color samples into a matrix with *N* rows x 3 columns,

$$(1) \qquad \mathbf{O} = \begin{bmatrix} O\_R_1 & O\_G_1 & O\_B_1 \\ O\_R_2 & O\_G_2 & O\_B_2 \\ . & . & . \\ . & . & . \\ . & . & . \\ O\_R_N & O\_G_N & O\_B_N \end{bmatrix} .$$

Let the processed image **P** also have *N* color samples that correspond to those in the original image,

$$(2) \qquad \mathbf{P} = \begin{bmatrix} P\_R_1 & P\_G_1 & P\_B_1 \\ P\_R_2 & P\_G_2 & P\_B_2 \\ . & . & . \\ . & . & . \\ . & . & . \\ P\_R_N & P\_G_N & P\_B_N \end{bmatrix} .$$

What we seek is the optimal linear transformation matrix **A** (4 rows x 3 columns) that best maps the processed color samples **P** into the corresponding original color samples **O**,

$$(3) \qquad \mathbf{O} \approx \hat{\mathbf{O}} = \left[\underline{\mathbf{1}} \quad \mathbf{P}\right]\mathbf{A} \ .$$

In the above equation, $\underline{\mathbf{1}}$ is a column vector of *N* ones that provides a DC offset, or shift, in the brightness level. Thus, each transformed pixel color is a linear combination of a DC offset and the processed red, green, and blue samples. For example, the *red* color of the first transformed pixel would be equal to

$$\hat{O}\_R_1 = A_{1,1} + A_{2,1} * P\_R_1 + A_{3,1} * P\_G_1 + A_{4,1} * P\_B_1 \ ,$$

where the two subscripts on the **A** matrix elements denote their row and column positions, respectively. If we have more than twelve independent RGB samples (the number of unknowns in the **A** matrix), then the set of linear equations is over-determined and the least-squares solution is given by

$$\text{(4)} \qquad \mathbf{A} = \left( \left[ \underline{\mathbf{1}} \quad \mathbf{P} \right]^{\mathbf{T}} \left[ \underline{\mathbf{1}} \quad \mathbf{P} \right] \right)^{-1} \left[ \underline{\mathbf{1}} \quad \mathbf{P} \right]^{\mathbf{T}} \mathbf{O} \ .$$

Equation (4) is the fundamental equation that we will use to estimate the **A** color correction matrix.

### 3.2 Robust Least-squares Algorithm for Color Correction Matrix

The processed image normally contains spatial distortions as well as color calibration problems. In addition, the processed image may not be properly registered, or aligned, to the original image. These problems can create outliers (processed color samples that do not agree with the majority fit) that may unduly influence the estimation of the color correction matrix **A**. In this section, we present an iterative least-squares solution with a cost function to help minimize the weight of outliers in the fit. The basic idea is to reduce the weight of outliers in the fit using a cost function that is inversely proportional to the error, or Euclidean distance, between the original sample $O$ and the fitted processed sample $\hat{O}$. If this error distance is large, then the associated cost of the fitting error will be small and the outlier's influence on the estimate of **A** will be minimal. To implement the robust least-squares solution, the following algorithm is applied to the *N* matching original and processed RGB color samples:

Step 1: Use the normal least-squares solution from equation (4) in section 3.1 to generate an initial estimate of the color correction matrix **A**.

Step 2: Generate an error vector $\underline{E}$ where each element $E_i$ is equal to the Euclidean distance between the original sample $O_i$ (equation (1), section 3.1) and the fitted processed samples $\hat{O}_i$ (equation (3), section 3.1):

$$\text{(5)} \qquad E_i = \sqrt{\left(O\_R_i - \hat{O}\_R_i\right)^2 + \left(O\_G_i - \hat{O}\_G_i\right)^2 + \left(O\_B_i - \hat{O}\_B_i\right)^2}, \quad i = 1,2,...,N$$

Step 3: Generate a cost vector ($\underline{C}$) that is the element-by-element reciprocal of the error vector ($\underline{E}$) from step 2 plus a small epsilon ($\varepsilon$):

$$\text{(6)} \qquad \underline{C} = \frac{1}{\underline{E} + \varepsilon} .$$

The $\varepsilon$ prevents division by zero and sets the relative weight of a point that is on the fitted line versus the weight of a point that is off the fitted line. An $\varepsilon$ of 0.1 is recommended.

Step 4: Normalize the cost vector $\underline{C}$ for unity norm (i.e., each element of $\underline{C}$ is divided by the square root of the sum of the squares of all the elements of $\underline{C}$).

Step 5: Generate the cost vector $\underline{C}^2$ that is the element-by-element square of the cost vector $\underline{C}$ from step 4.

Step 6: Generate an $N \times N$ diagonal cost matrix ($\mathbf{C}^2$) that contains the cost vector's elements ($\underline{C}^2$) arranged on the diagonal, with zeros everywhere else.

Step 7: Using the diagonal cost matrix ($\mathbf{C}^2$) from step 6, perform cost-weighted least-squares fitting to determine the next estimate of the color correction matrix **A**:

$$\mathbf{A} = \left( \left[ \underline{\mathbf{1}} \quad \mathbf{P} \right]^T \mathbf{C}^2 \left[ \underline{\mathbf{1}} \quad \mathbf{P} \right] \right)^{-1} \left[ \underline{\mathbf{1}} \quad \mathbf{P} \right]^T \mathbf{C}^2 \mathbf{O} . \tag{7}$$

Step 8: Repeat steps 2 through 7 until the elements of the color correction matrix **A** converge to four decimal places.

### 3.3 Non-linear Monotonic Transfer Function

Sections 3.1 and 3.2 describe linear methods for calculating the color correction matrix. System non-linearities may arise due to non-linearities in the sensor (e.g., charge coupled devices, CCD) or system under test (e.g., non-linear video system gain). This section presents one method for performing a monotonic non-linear correction to the individual color components.

The order in which non-linear and linear corrections are performed is not in general interchangeable. Hence, if a non-linear correction is performed, one must make a decision to apply the non-linear correction before or after the linear correction. A careful examination of the entire system under test may be used to help make this decision. For instance, if the imaging sensor or signal processing in the camera is known to have non-linearities, then it may be appropriate to first remove these non-linearities before performing the color matrix correction. However, if you only have access to image data after some unknown linear color manipulation was performed, it may be more appropriate to first apply the linear color correction matrix (to undo the color manipulation) and then remove the remaining non-linearities. An examination or plot of the original image color samples versus the processed image color samples may provide clues as to which approach is better.

Since a monotonic increasing nonlinear transformation is desired on color samples from the processed image, where $x_1 < x_2$ implies $f(x_1) \leq f(x_2)$, one must compute a constrained least squares fit where the x-variable contains the processed color samples and the y-variable contains the corresponding original color samples. The processed color samples can then be non-linearly transformed by this fitting function. This constrained optimization problem is easily handled by the MATLAB routine LSQLIN, which is contained in the optimization toolbox.

The choice of the fitting function depends upon the shape of the non-linearities. We have found that a third order polynomial provides sufficient degrees of freedom to track most common non-linearities that seem to result from compression and/or expansion at the extremes of the amplitude scale. The appendix provides MATLAB code for performing a monotonic polynomial fit that utilizes the LSQLIN function.

### 3.4 Color Space Considerations

We have presented the color correction algorithms using the Red, Green, and Blue (RGB) color space formulation. However, many different color spaces have been developed for various reasons. The intent

of this document is not to recommend one color space over another but simply to present general algorithms that can be applied in any color space.

An excellent overview of the different color spaces can be found in [5]. One important consideration is the color space in which to apply the color correction algorithms given above. Some color spaces have been specifically designed to be more perceptually uniform (i.e., calculated differences between two colors represent how different they are from a perceptual standpoint) than the RGB color space. In a perceptually uniform color space, errors calculated by the least squares fitting algorithms should more accurately represent actual *perceptual* changes in color. This should produce final corrected images that look more closely matched to the original image.

Another important consideration is the use of gamma corrected color spaces (see [6]). Gamma correction was originally developed as a compensation for the non-linear responses of cathode ray tubes (CRTs). By adding a gamma correction to the output of a camera that was the inverse of the nonlinear CRT response, the correct overall response could be achieved. While gamma correction was originally intended to compensate for the non-linear response of the CRT monitor, by a remarkable coincidence the non-linear response of the human visual system is approximately the inverse of the nonlinear response of a CRT monitor. Thus, gamma pre-corrected color signals are already *approximately* perceptually coded. For digital camera images, the standard RGB (sRGB) color space [3], supported by most personal computers and photo-editing software, includes gamma pre-correction. For video signals, ITU-R Recommendation BT.601 [1], which specifies luminance (Y), chrominance blue (Cb), and chrominance red (Cr) color component signals, also includes gamma pre-correction.

## 4. APPLYING THE COLOR CORRECTION ALGORITHM

This section provides examples of applying the color correction algorithms in section 3 to digital camera pictures and video transmission systems (as shown in Figure 1 and Figure 2), respectively.

### 4.1 Digital Cameras

To demonstrate the application of the color correction algorithm to digital camera images, a test chart with 100 randomly distributed color patches was generated using the MATLAB routine generate_chart given in the appendix. This test chart was then loaded into image editing software as an sRGB color space image and converted to the cyan, magenta, yellow, and black (CMYK) color space in preparation for printing. This step assures that all the randomly generated sRGB colors were indeed printable by the limited color gamut of CMYK (the most commonly used printer color space). The test chart was then converted back to sRGB to become the “original” sRGB reference image used for the experiment (see Figure 3).

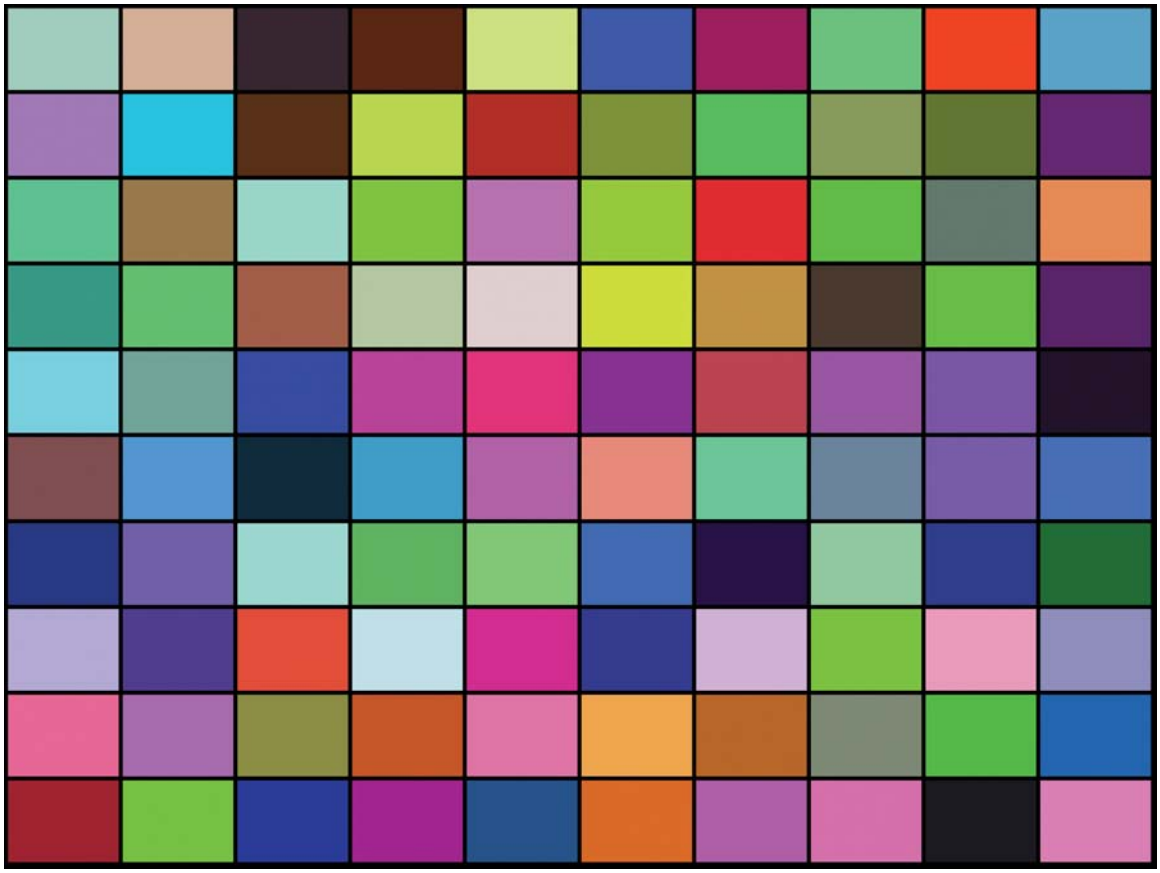

Figure 3. Reference test chart (i.e., the original image).

The CMYK color space test chart was printed and uniformly illuminated with 3200K light. A tripod mounted digital camera with white balance set to "auto" (i.e., the camera tries to pick the best white balance) was used to take the photo shown in Figure 4.

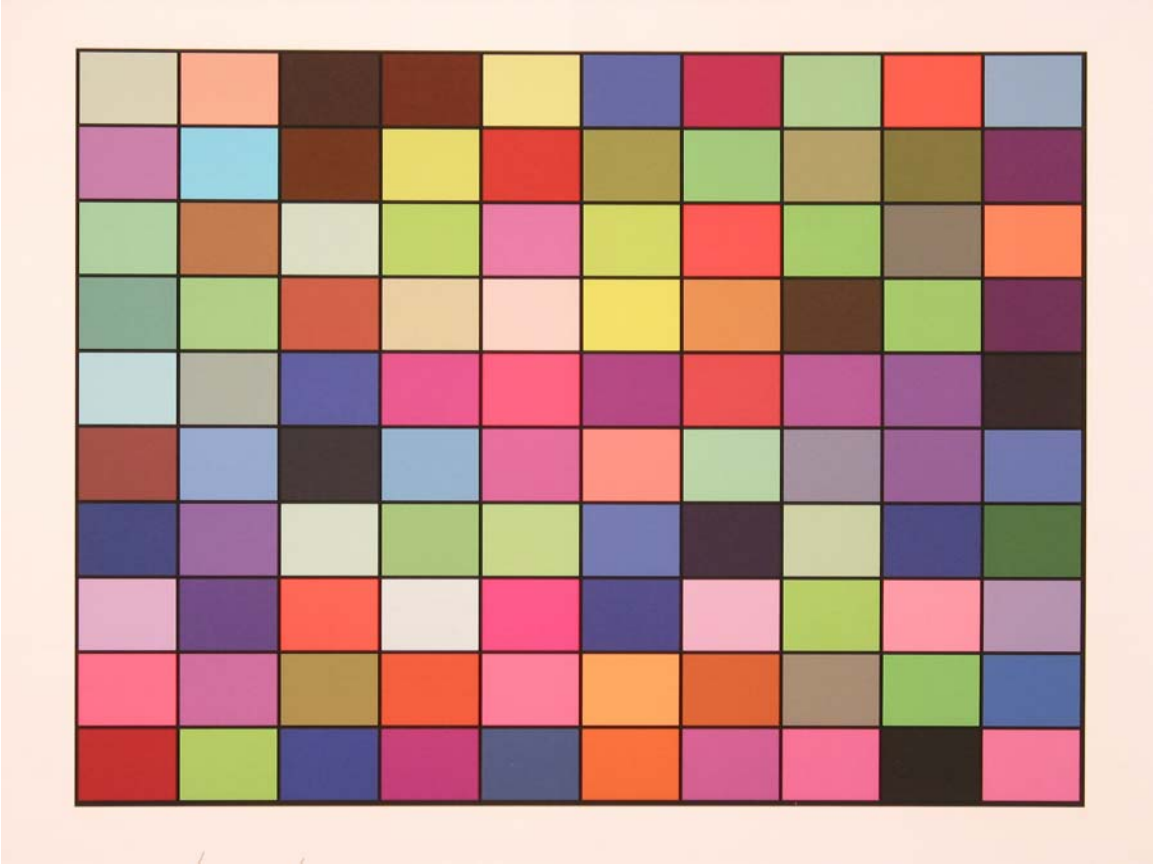

Figure 4. Test chart from camera (i.e., the processed image).

Examination of the means (or average values) of the red, green, and blue color components in the color patches, original versus processed, revealed the presence of non-linearities, particularly in the red component (see Figure 5, Figure 6, and Figure 7, respectively, for the red, green, and blue components). A rectangular sub-region fully contained within each color patch (with some added safety margin) was used by the color_xform routine to calculate these color patch means (see section A.1 in the appendix). The camera's automatic white balance estimation performed poorly in this example and the image white balance is shifted toward red. This is evident by the red border around the test chart, which was printed on white paper. Examination of Figure 5 also reveals what appears to be red level clipping at 255 for several of the color patches. Once clipping occurs, the information cannot be recovered. Hence, we expect that several of the color patches with a high amount of red will not be fully corrected in the final calibrated image.

As a result of these observations, non-linear pre-correction was activated in the color_xform routine given in the appendix. Figure 8 gives the nonlinear pre-correction that was applied to the red color component.

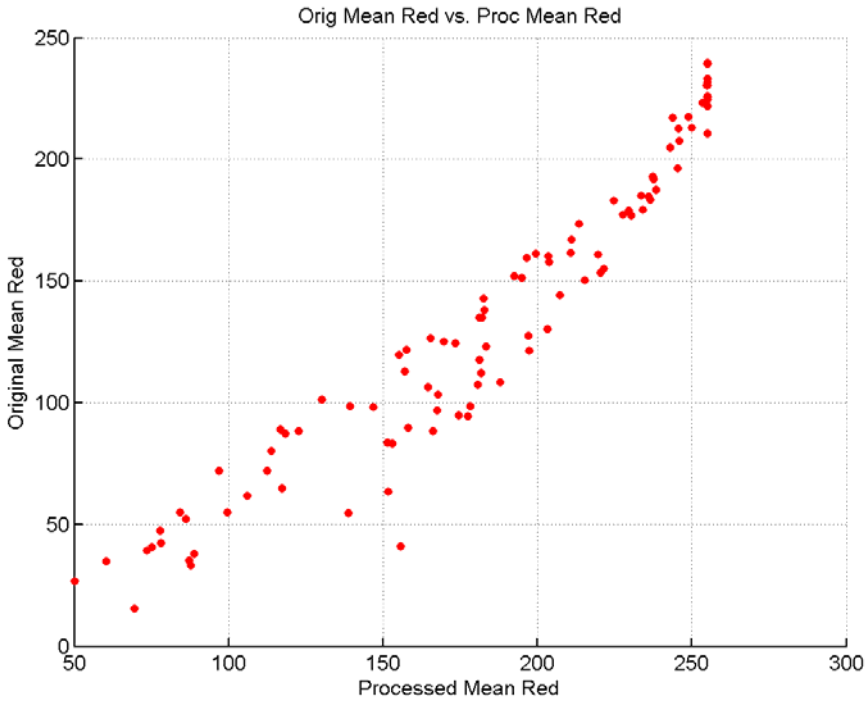


Figure 5. Graph of the red component means, original versus processed.

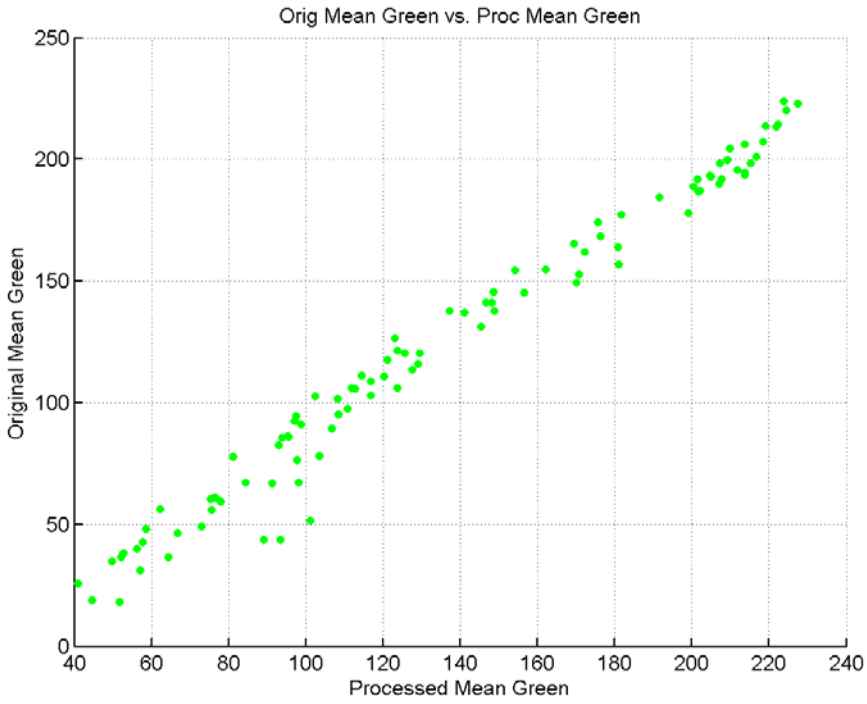


Figure 6. Graph of the green component means, original versus processed.

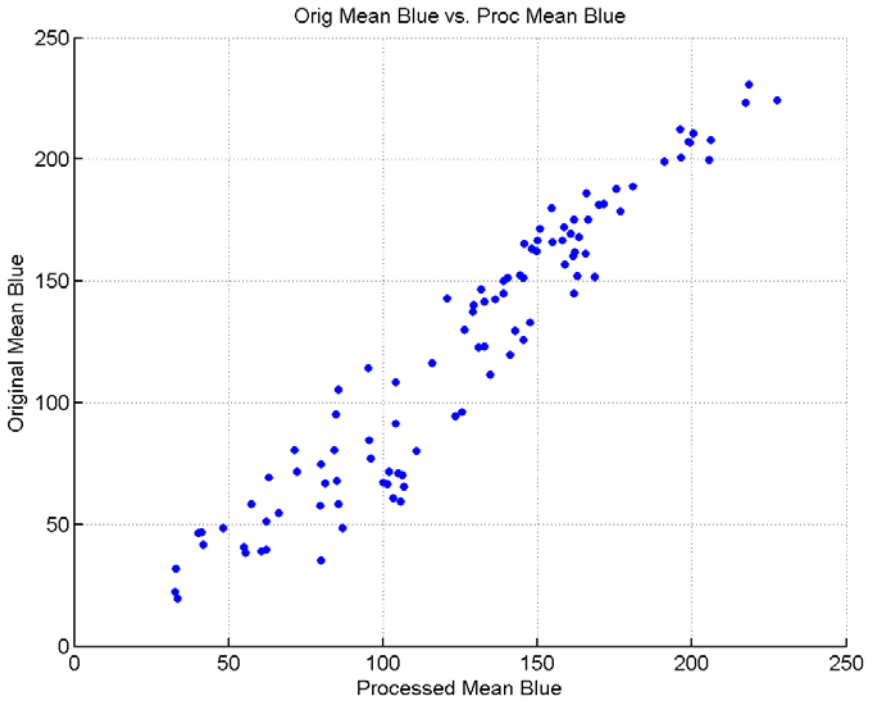


Figure 7. Graph of the blue component means, original versus processed.

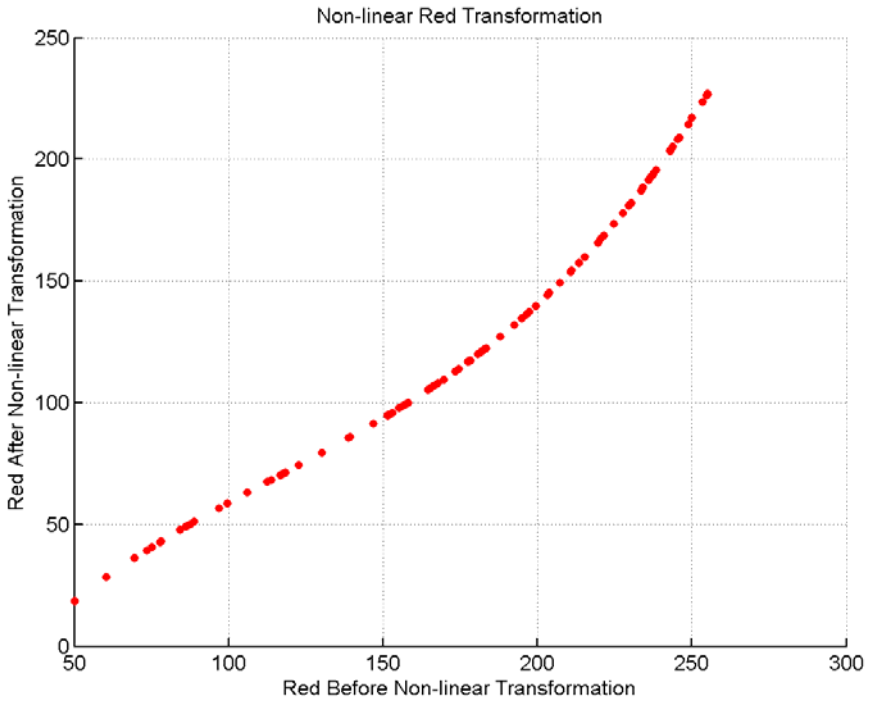


Figure 8. Nonlinear pre-correction applied to the red component.

The color correction matrix algorithm described in section 3.1 was then applied using the color_xform MATLAB routine given in the appendix. The resultant calibrated image is shown in Figure 9. Figure 10, Figure 11, and Figure 12 provide the red, green, and blue color component plots, respectively, of the original image colors versus the final calibrated image colors. These three plots can be compared with the three plots from the un-calibrated image that are shown in Figure 5, Figure 6, and Figure 7, respectively. Most of the colors in the calibrated image look very close to the original image. Several of the color patches in the calibrated image may look slightly different from the original image due to either the high amplitude clipping at 255 in the red component or because the camera cannot reproduce the full gamut of colors in the original test chart.

In conducting this experiment, it was found that the uniformity of the light intensity (over the test chart) was critical. A non-uniform distribution of light intensity shifts the different color patches by varying amounts and can cause an increase in the amount of scatter shown in the plots. The light intensity used for the camera image in Figure 4 was uniform to about 5%. One possible application for the color correction algorithms illustrated here would be to correct images from specific cameras under known lighting conditions (e.g., color temperature of illuminating light) and camera operating conditions (e.g., preset white balance).

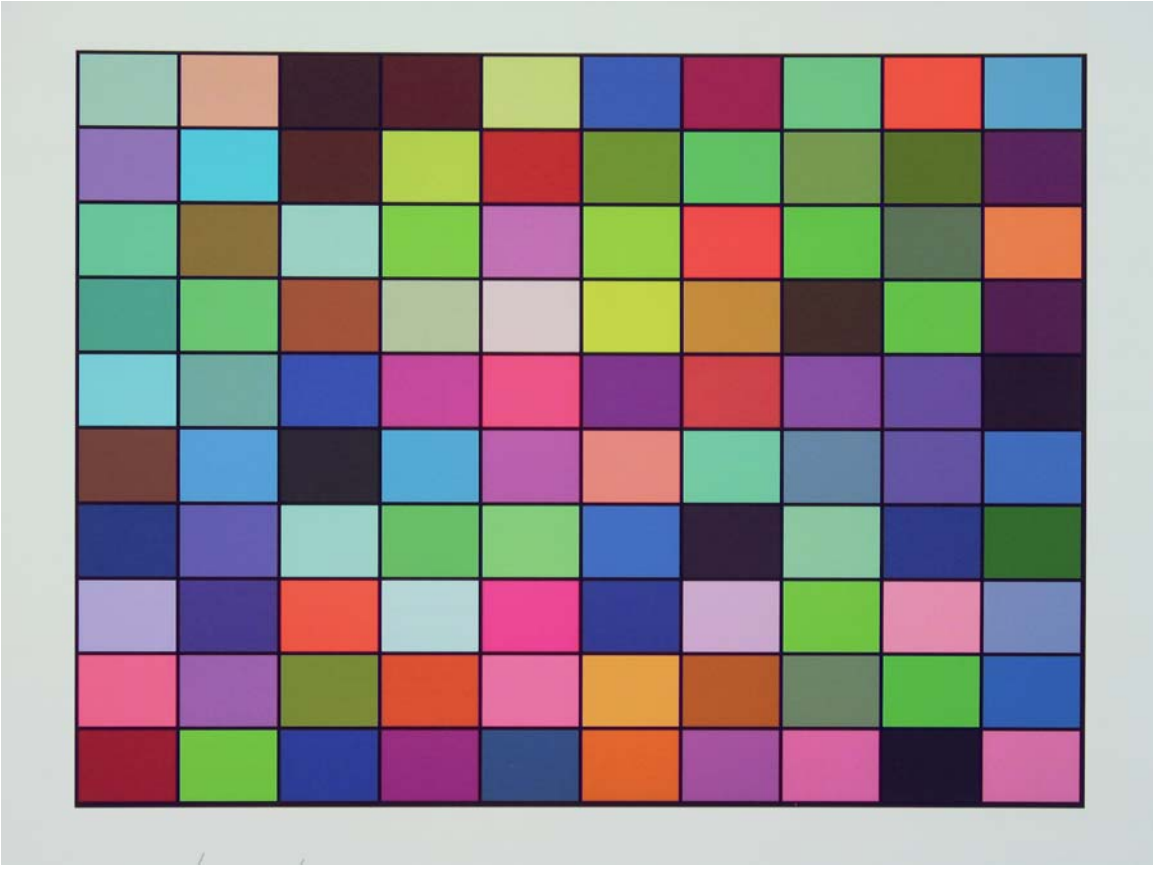

Figure 9. Test chart from camera after color correction algorithms (i.e., the calibrated image).

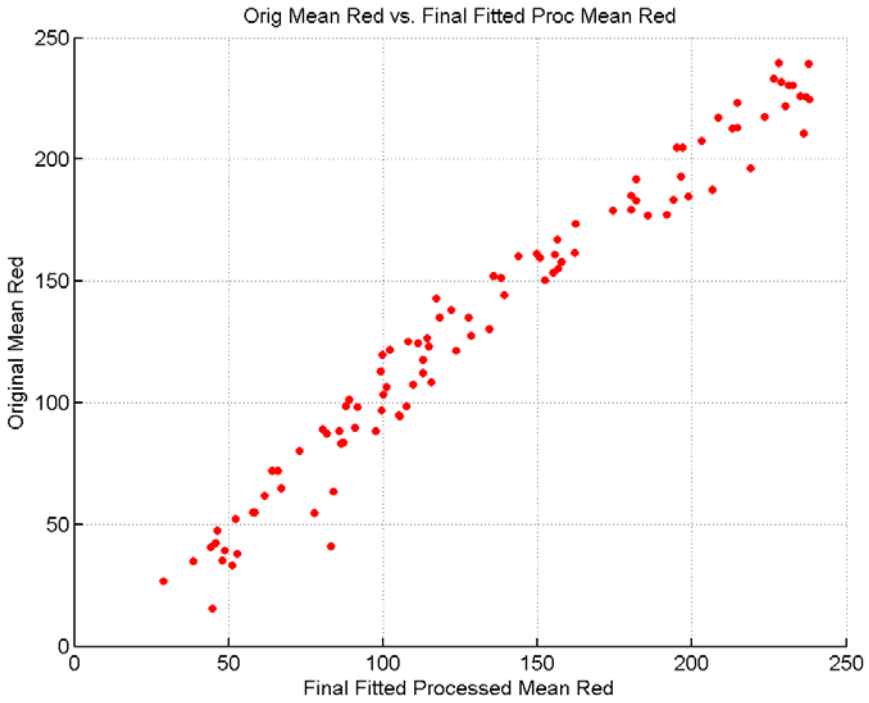


Figure 10. Graph of the red component means, original versus calibrated.

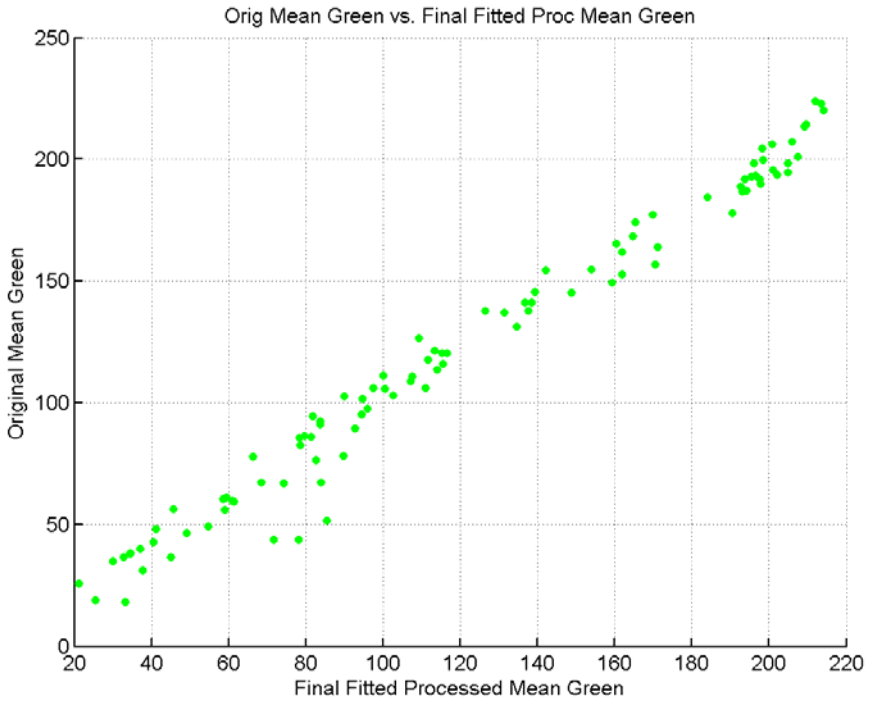


Figure 11. Graph of the green component means, original versus calibrated.

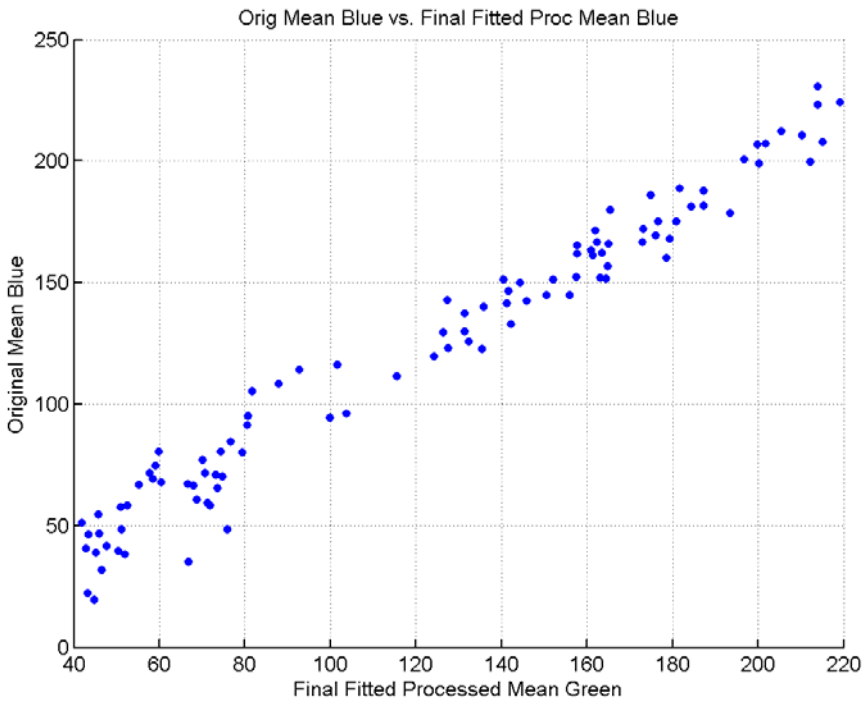


Figure 12. Graph of the blue component means, original versus calibrated.

### 4.2 Video Transmission Systems

To demonstrate the application of the color correction algorithm to video systems, an original video scene was sent through a low bit-rate video conferencing system that introduced color distortions. The original video and processed video (i.e., the video output by the video conferencing system) were sampled according to Rec. 601, which produces a luminance component (Y), a blue chrominance component Cb, and a red chrominance component Cr. Before application of the color correction algorithm, the original and processed video streams were spatially and temporally registered (to remove spatial and temporal shifts). Next, a 640 x 448 sub-region was selected from each video stream. Figure 13 is the sub-region that was selected from the original video stream and Figure 14 is the sub-region that was selected from the processed video stream. Note that the video conferencing system has introduced significant color distortions in the flesh tones, the hair, and the pink foreground object.

The 640 x 448 sub-region was then divided into 8 x 8 blocks, which were treated like color patches. Thus, the selected sub-region produced 80 x 56 = 448 color patches. For each 8 x 8 block, the average Y, Cb, and Cr values were calculated and these values were used by the color correction algorithm. Examination of the original color component values versus the processed color component values did not reveal any nonlinearities. Thus, the color_xform routine given in the appendix was run without nonlinear correction. The color correction matrix algorithm given in section 3.1 produced the image shown in Figure 15. The algorithms were clearly successful in correcting the color distortions.

Figure 13. One frame from an original video scene (i.e., the original image).

Figure 14. The corresponding frame from a video conferencing system (i.e., the processed image).

Figure 15. The processed image after color correction algorithms (i.e., the calibrated image).

One possible application for the color correction algorithms shown here is to correct for video color distortions that are more complicated than a simple gain and DC shift in each of the video signal components (Y, Cb, and Cr). For instance, in NTSC and Y/C systems (i.e., S-video), the chrominance information (Cb and Cr) is on a vector whose phase is interpreted as hue and whose magnitude is interpreted as saturation. If this chrominance vector has the wrong phase (i.e., rotated), the hue will be off and a simple gain and DC offset in the individual Cb and Cr components cannot correct for this distortion. However, the color correction matrix algorithm presented in this report can perform an effective rotation on the chrominance vector and thus compensate for the hue distortion.

## 5. CONCLUSIONS

We have presented a simple method and MATLAB code for performing color correction in digital still and video imaging systems. This method can automatically estimate the optimal color channel mixing matrix that must be applied to output images from digital still cameras or video systems in order to correct their color inaccuracies. In addition, a method and MATLAB code to correct for non-linearities in the color response of digital imaging systems were also presented. We demonstrated the usefulness of these color correction algorithms by applying them to actual output from digital still cameras and video systems that have color distortions in their output images.

## 6. REFERENCES


[1] ITU-R Recommendation BT.601, "Encoding parameters of digital television for studios," Recommendations of the ITU, Radiocommunication Sector.

[2] SMPTE 170M, "SMPTE Standard for Television – Composite Analog Video Signal – NTSC for Studio Applications," Society of Motion Picture and Television Engineers, 595 West Hartsdale Avenue, White Plains, NY 10607.

[3] The sRGB Website, www.srgb.com. Last accessed November 28, 2003.

[4] International Color Consortium (ICC) Specification ICC.1:2001-12, "File Format for Color Profiles," copyrighted 1994-2001, available for download at www.color.org. Last accessed November 28, 2003.

[5] Charles A. Poynton, "Frequently Asked Questions about Color," copyrighted 1997-02-27 [f], available for download at http://www.poynton.com/. Last accessed November 28, 2003.

[6] Charles A. Poynton, "Frequently Asked Questions about Gamma," copyrighted 1999-12-30 [b], available for download at http://www.poynton.com/. Last accessed November 28, 2003.

## APPENDIX: MATLAB CODE

This appendix contains MATLAB Release 13 routines for performing the algorithms described in section 3. Code is also provided for generating and storing an RGB color chart with random color samples. Color charts may be useful for testing digital cameras (see for example, section 4.1).

### A.1 ROUTINE COLOR_XFORM.M

MATLAB routine color_xform.m contains the color correction algorithms described in section 3. An example MATLAB command line call that was used to generate the images and plots in section 4.1 is given by:

- [cal, a, fit] = color_xform (orig, 1, 1, 1704, 2272, orig_patch, proc, 104, 150, 1590, 2138, 0, -1);

In this example, "orig" is the original image and "proc" is the processed image. The top, left, bottom, and right rectangular sub-region coordinates to use for the original image are 1, 1, 1704, 2272 while the corresponding coordinates for the processed image are 104, 150, 1590, 2138. Thus, the processed image has undergone spatial scaling and shifting when referenced to the original image. The "orig_patch" variable is a two dimensional matrix that gives the location and size of the color patches in the original image where these quantities are calculated using an added safety margin to eliminate transitional effects between the different color patches (see the description of "patch_info" in section A.3 for more information). The corresponding patch information for the processed image is obtained by spatially scaling and shifting that is deduced from the top, left, bottom, and right coordinates for the original and processed images. The last two variables of the function call "0, -1" specify that the color correction matrix calculation should be computed using a normal least-squares fit (as given in section 3.1, rather than the robust least-squares fit given in section 3.2), and that non-linear pre-correction should be performed on the color components before calculating the color correction matrix. The color_xform function returns the calibrated processed image "cal", the color correction matrix "a", and the non-linear fit "fit". See the header comments in the MATLAB code for more information.

```
function [cal_img, a, fit] = color_xform(orig_img, orig_top, orig_left, orig_bot, orig_right, ...
    orig_patch_info, proc_img, proc_top, proc_left, proc_bot, proc_right, robust, nonlinear);

%  function [cal_img, a, fit] = color_xform(orig_img, orig_top, orig_left, orig_bot, orig_right, ...
%      orig_patch_info, proc_img, proc_top, proc_left, proc_bot, proc_right, robust, nonlinear);
%
%  Calculates and returns the calibrated color corrected processed image (cal_img), given an
%  original image (orig_img) and a processed image (proc_img).  orig_img and proc_img must have
%  the same dimensions and are three dimensional with each third dimension holding the color
%  image planes (e.g., R, G, B).
%
%  If two output arguments are requested, also returns the 4 x 3 least squares color correction matrix
%  transformation [A] that maps a processed image (proc_img) to the original image (orig_img)
%  according to the following transformation:
%
% |orig_r_1 orig_g_1 orig_b_1|       |1 proc_r_1 proc_g_1 proc_b_1|
% |orig_r_2 orig_g_2 orig_b_2|       |1 proc_r_2 proc_g_2 proc_b_2|
% |orig_r_3 orig_g_3 orig_b_3|       |1 proc_r_3 proc_g_3 proc_b_3|
% |    .        .        .   |   =   |.     .        .        .   |  *  [A]
% |    .        .        .   |       |.     .        .        .   |        4 x 3
% |    .        .        .   |       |.     .        .        .   |
% |orig_r_n orig_g_n orig_b_n|       |1 proc_r_n proc_g_n proc_b_n|
%                           n x 3                                  n x 4
%
%  In the above equation, n is the number of color patches that will be used to perform the
%  least-squares estimate of A (see orig_patch_info below).
%
%  If three output arguments are requested, also returns the non-linear pre-correction or post
%  correction fit that was individually applied to the color components (see below).
%
%  The image sub-region examined by the alogorithms is specified by TOP, LEFT, BOT, & RIGHT.  If
```

```
%  using a test chart, this image sub-region should be the top-left and bot-right corners of the
%  test chart and include the black portion of the test chart for both the original image and the
%  processed image.  The top-left pixel in the image is defined as coordinate (1, 1).  The image
%  sub-regions of the original and processed images are compared and used to compute a linear
%  transformation factor between the orig_patch_info and the corresponding patch information for
%  the processed image.  Thus, for example, a test chart contained within the original and
%  processed images does not have to be the same size or located in the same horizontal and vertical
%  position.  However, the two sub-regions must not be rotated with respect to each other as this
%  routine does not perform any rotational correction.
%
%  orig_patch_info is a two dimensional matrix where each row specifies the following information
%  about each color patch in the original image:
%       first_row - the first row number of the color patch
%       first_col - the first col number of the color patch
%       row_size - the number of rows in the color patch
%       col_size - the number of cols in the color patch
%
%  The number of rows in orig_patch_info specifies the number of points n that will be used in the
%  above least squares solution to find the color transformation matrix [A].  The user may want to add
%  a safety margin to each beginning patch location and size since the entire patch area specified by
%  the orig_patch_info matrix is averaged to produce low noise estimates of the color values for each
%  patch before the least squares solution is calculated. Routine generate_chart can be used to obtain
%  the orig_patch_info with a safety margin added.
%
%  The following inputs control the least-squares fitting procedure:
%       robust = 0  Perform a normal least-squares fit
%              = 1  Perform a robust least-squares fit reducing outlier weights.
%
%       nonlinear = 0  Do not perform a non-linear transform on R, G, and B (individually).
%                 = -1  Perform a non-linear transform on R, G, and B before the color matrix
%                 =  1  Perform a non-linear transform on R, B, and B after the color matrix
%

%  Set the order of the non-linear fit.  Use a polynomial of order 3 since this approximates
%  non-linear s-curve responses.
order = 3;
fit = zeros (order+1,3);

%  Check image sizes
if (size(orig_img) ~= size(proc_img))
    disp('The original and processed images must have the same dimensions');
    return
end

%  Find the total number of color patches
n = size(orig_patch_info,1);

%  Compute the translation and scale between the original color
%  chart patches and the processed color chart patches.
proc_patch_info = zeros(n,4);

r_scale = (proc_bot-proc_top)/(orig_bot-orig_top);  % row size scale factor
proc_patch_info(:,3) = orig_patch_info(:,3)*r_scale;

c_scale = (proc_right-proc_left)/(orig_right-orig_left);  % col size scale factor
proc_patch_info(:,4) = orig_patch_info(:,4)*c_scale;

r_shift = proc_top - r_scale*orig_top;  % row shift after size scaling
proc_patch_info(:,1) = r_scale*orig_patch_info(:,1) + r_shift;

c_shift = proc_left - c_scale*orig_left;  % col shift after size scaling
proc_patch_info(:,2) = c_scale*orig_patch_info(:,2) + c_shift;

%  Round processed patch indices to the nearest integer
proc_patch_info = round(proc_patch_info);

%  Calculate the R, G, B mean of each patch.
orig_mean = zeros(n,3);
proc_mean = zeros(n,3);
for i = 1:n
    %  Original image
    or_1 = orig_patch_info(i,1);  % top coordinate of orig patch
    oc_1 = orig_patch_info(i,2);  % left coordinate of orig patch
    or_2 = orig_patch_info(i,1) + orig_patch_info(i,3) - 1;  % bottom coordinate of original patch
    oc_2 = orig_patch_info(i,2) + orig_patch_info(i,4) - 1;  % right coordinate of original patch
    orig_mean(i,:) = ...
        mean(reshape(orig_img(or_1:or_2, oc_1:oc_2, :),orig_patch_info(i,3)*orig_patch_info(i,4),3));

    %  Processed image
    cr_1 = proc_patch_info(i,1);  % top coordinate of processed patch
    cc_1 = proc_patch_info(i,2);  % left coordinate of processed patch
    cr_2 = proc_patch_info(i,1) + proc_patch_info(i,3) - 1;  % bottom coordinate of processed patch
    cc_2 = proc_patch_info(i,2) + proc_patch_info(i,4) - 1;  % right coordinate of processed patch
    proc_mean(i,:) = ...
        mean(reshape(proc_img(cr_1:cr_2, cc_1:cc_2, :),proc_patch_info(i,3)*proc_patch_info(i,4),3));
end
```

```
% Free up memory
clear orig_img

% Plot results and store figures
figure(4)
hold on
plot(proc_mean(:,1), orig_mean(:,1),'r.','markersize', 15);
grid on
set(gca,'LineWidth',1)
set(gca,'FontName','Ariel')
set(gca,'fontsize',12)
xlabel('Processed Mean Red')
ylabel('Original Mean Red')
title('Orig Mean Red vs. Proc Mean Red')
hold off
print -depsc2 -tiff figure4
print -dpng figure4

figure(5)
hold on
plot(proc_mean(:,2), orig_mean(:,2),'g.', 'markersize', 15);
grid on
set(gca,'LineWidth',1)
set(gca,'FontName','Ariel')
set(gca,'fontsize',12)
xlabel('Processed Mean Green')
ylabel('Original Mean Green')
title('Orig Mean Green vs. Proc Mean Green')
hold off
print -depsc2 -tiff figure5
print -dpng figure5

figure(6)
hold on
plot(proc_mean(:,3), orig_mean(:,3),'b.', 'markersize', 15);
grid on
set(gca,'LineWidth',1)
set(gca,'FontName','Ariel')
set(gca,'fontsize',12)
xlabel('Processed Mean Blue')
ylabel('Original Mean Blue')
title('Orig Mean Blue vs. Proc Mean Blue')
hold off
print -depsc2 -tiff figure6
print -dpng figure6

%  Apply a pre monotonic non-linear transformation on R, G, and B (individually) if requested.
if (nonlinear == -1)
    proc_mean_nlin = zeros(n,3);
    [fit(:,1), proc_mean_nlin(:,1)] = polyfit_monotonic(proc_mean(:,1), orig_mean(:,1), 3);
    [fit(:,2), proc_mean_nlin(:,2)] = polyfit_monotonic(proc_mean(:,2), orig_mean(:,2), 3);
    [fit(:,3), proc_mean_nlin(:,3)] = polyfit_monotonic(proc_mean(:,3), orig_mean(:,3), 3);

    % Plot results and store figures.
    figure(10)
    hold on
    plot(proc_mean(:,1),proc_mean_nlin(:,1),'.r', 'markersize', 15);
    grid on
    set(gca,'LineWidth',1)
    set(gca,'FontName','Ariel')
    set(gca,'fontsize',12)
    xlabel('Red Before Non-linear Transformation')
    ylabel('Red After Non-linear Transformation')
    title('Non-linear Red Transformation')
    hold off
    print -depsc2 -tiff figure10
    print -dpng figure10

    figure(11)
    hold on
    plot(proc_mean(:,2),proc_mean_nlin(:,2),'.g', 'markersize', 15);
    grid on
    set(gca,'LineWidth',1)
    set(gca,'FontName','Ariel')
    set(gca,'fontsize',12)
    xlabel('Green Before Non-linear Transformation')
    ylabel('Green After Non-linear Transformation')
    title('Non-linear Green Transformation')
    hold off
    print -depsc2 -tiff figure11
    print -dpng figure11

    figure(12)
    hold on
    plot(proc_mean(:,3),proc_mean_nlin(:,3),'.b', 'markersize', 15);
```

```
    grid on
    set(gca,'LineWidth',1)
    set(gca,'FontName','Ariel')
    set(gca,'fontsize',12)
    xlabel('Blue Before Non-linear Transformation')
    ylabel('Blue After Non-linear Transformation')
    title('Non-linear Blue Transformation')
    hold off
    print -depsc2 -tiff figure12
    print -dpng figure12

    %  Re-assign proc_mean to proc_mean_nlin in preparation for color matrix calculation
    proc_mean = proc_mean_nlin;

end

%  Add col of ones to proc_mean in preparation for linear fit
proc_mean = [ones(n,1) proc_mean];

% Calculate a using least squares fit: all points equally weighted.
a = proc_mean\orig_mean;

% Calculate a using iterative robust fitting option that reduces the weight of outliers.
if (robust)
    max_tries = 1000;  % Maximum number of tries per each robust fit
    max_change = .0001;  % Maximum change allowed on each matrix element before stopping
    i_try = 0;
    this_change = inf; % initial large number to force execution of while loop
    while ((i_try <= max_tries) & (this_change > max_change))
        i_try = i_try + 1;
        last_a = a;
        % Decrease the weight on the outliers and perform new fit
        err = orig_mean - proc_mean*a;  % error on each R, G, and B component, sample by sample
        err = sqrt(err(:,1).^2 + err(:,2).^2 + err(:,3).^2); % Euclidean error for each RGB sample
        cost = 1./(err+.1);  % cost vector is inverse of error, limited to prevent divide by zero
        cost = cost./norm(cost);  % normalize the cost vector
        cost2 = cost.*cost;  % square the cost vector
        cost2_mat = sparse(1:n,1:n,cost2,n,n);  % create the cost matrix
        a = inv(proc_mean'*cost2_mat*proc_mean)*proc_mean'*cost2_mat*orig_mean;  % updated a
        this_change = max(max(abs(a-last_a)));  % calculate them maximum change in the a elements
    end
end

%  Apply a post monotonic non-linear transformation on R, G, and B (individually) if requested.
if (nonlinear == 1)
    orig_mean_hat = proc_mean*a;
    orig_mean_hat_nlin = zeros(n,3);

    [fit(:,1), orig_mean_hat_nlin(:,1)] = polyfit_monotonic(orig_mean_hat(:,1), orig_mean(:,1), 3);
    [fit(:,2), orig_mean_hat_nlin(:,2)] = polyfit_monotonic(orig_mean_hat(:,2), orig_mean(:,2), 3);
    [fit(:,3), orig_mean_hat_nlin(:,3)] = polyfit_monotonic(orig_mean_hat(:,3), orig_mean(:,3), 3);

    % Plot results and store figures.
    figure(10)
    hold on
    plot(orig_mean_hat(:,1),orig_mean_hat_nlin(:,1),'.r', 'markersize', 15);
    grid on
    set(gca,'LineWidth',1)
    set(gca,'FontName','Ariel')
    set(gca,'fontsize',12)
    xlabel('Red Before Non-linear Transformation')
    ylabel('Red After Non-linear Transformation')
    title('Non-linear Red Transformation')
    hold off
    print -depsc2 -tiff figure10
    print -dpng figure10

    figure(11)
    hold on
    plot(orig_mean_hat(:,2),orig_mean_hat_nlin(:,2),'.g', 'markersize', 15);
    grid on
    set(gca,'LineWidth',1)
    set(gca,'FontName','Ariel')
    set(gca,'fontsize',12)
    xlabel('Green Before Non-linear Transformation')
    ylabel('Green After Non-linear Transformation')
    title('Non-linear Green Transformation')
    hold off
    print -depsc2 -tiff figure11
    print -dpng figure11

    figure(12)
    hold on
    plot(orig_mean_hat(:,3),orig_mean_hat_nlin(:,3),'.b', 'markersize', 15);
    grid on
    set(gca,'LineWidth',1)
```

```
    set(gca,'FontName','Ariel')
    set(gca,'fontsize',12)
    xlabel('Blue Before Non-linear Transformation')
    ylabel('Blue After Non-linear Transformation')
    title('Non-linear Blue Transformation')
    hold off
    print -depsc2 -tiff figure12
    print -dpng figure12

end

% Calculate the calibrated processed image (cal_img).
% Find the total number of rows and columns in proc_img
[nr, nc, planes] = size(proc_img);

if (nonlinear == 0)
    %  Apply the matrix fit that was calculated to find the final processed image.
    cal_img = [ones(nr*nc,1) reshape(permute(proc_img,[3,1,2]), 3, nr*nc)'] * a;
end

if (nonlinear == 1)
    %  Apply the color matrix fit first.
    cal_img = [ones(nr*nc,1) reshape(permute(proc_img,[3,1,2]), 3, nr*nc)'] * a;
    % Then apply the nonlinear transform function that was found on each R, G, and B component.
    cal_img(:,1) = polyval(fit(:,1), cal_img(:,1));
    cal_img(:,2) = polyval(fit(:,2), cal_img(:,2));
    cal_img(:,3) = polyval(fit(:,3), cal_img(:,3));
end

if (nonlinear == -1)
    %  Apply the nonlinear transform function that was found on each R, G, and B component first.
    cal_img = zeros(nr*nc,planes);
    proc_img_reshape = reshape(permute(proc_img,[3,1,2]), 3, nr*nc)';
    cal_img(:,1) = polyval(fit(:,1), proc_img_reshape(:,1));
    cal_img(:,2) = polyval(fit(:,2), proc_img_reshape(:,2));
    cal_img(:,3) = polyval(fit(:,3), proc_img_reshape(:,3));
    clear proc_imge_reshape
    %  Then apply the color matrix fit.
    cal_img = [ones(nr*nc,1) cal_img] * a;
end

%  Reformat cal_img
cal_img = reshape(cal_img, nr, nc,3);

%  Calculate the R, G, B mean of each patch in new calibrated image.
proc_mean2 = zeros(n,3);
for i = 1:n
    %  Processed image
    cr_1 = proc_patch_info(i,1);  % top coordinate of processed patch
    cc_1 = proc_patch_info(i,2);  % left coordinate of processed patch
    cr_2 = proc_patch_info(i,1) + proc_patch_info(i,3) - 1;  % bottom coordinate of processed patch
    cc_2 = proc_patch_info(i,2) + proc_patch_info(i,4) - 1;  % right coordinate of processed patch
    proc_mean2(i,:) = ...
        mean(reshape(cal_img(cr_1:cr_2, cc_1:cc_2, :),proc_patch_info(i,3)*proc_patch_info(i,4),3));
end


% Plot results versus original and store figures.
figure(7)
hold on
plot(proc_mean2(:,1), orig_mean(:,1),'r.', 'markersize', 15);
grid on
set(gca,'LineWidth',1)
set(gca,'FontName','Ariel')
set(gca,'fontsize',12)
xlabel('Final Fitted Processed Mean Red')
ylabel('Original Mean Red')
title('Orig Mean Red vs. Final Fitted Proc Mean Red')
hold off
print -depsc2 -tiff figure7
print -dpng figure7

figure(8)
hold on
plot(proc_mean2(:,2), orig_mean(:,2), 'g.', 'markersize', 15);
grid on
set(gca,'LineWidth',1)
set(gca,'FontName','Ariel')
set(gca,'fontsize',12)
xlabel('Final Fitted Processed Mean Green')
ylabel('Original Mean Green')
title('Orig Mean Green vs. Final Fitted Proc Mean Green')
hold off
print -depsc2 -tiff figure8
print -dpng figure8
```

```
figure(9)
hold on
plot(proc_mean2(:,3), orig_mean(:,3), 'b.', 'markersize', 15);
grid on
set(gca,'LineWidth',1)
set(gca,'FontName','Ariel')
set(gca,'fontsize',12)
xlabel('Final Fitted Processed Mean Green')
ylabel('Original Mean Blue')
title('Orig Mean Blue vs. Final Fitted Proc Mean Blue')
hold off
print -depsc2 -tiff figure9
print -dpng figure9
```

## A.2 ROUTINE POLYFIT_MONOTONIC.M

MATLAB routine polyfit_monotonic.m contains the non-linear monotonic polynomial fitting routines described in section 3.3. An example MATLAB function call that was used by routine color_xform.m to generate the images and plots in section 4.1 is given by:

- [fit, proc_mean_nlin] = polyfit_monotonic (proc_mean, orig_mean, 3);

In this example, "proc_mean" is a column vector that holds the mean color patch values of the processed image, "orig_mean" is a column vector that holds the mean color patch values of the original image, and "3" is the order of the polynomial fit. The function returns the polynomial fit (i.e., "fit") and the proc_mean values after the non-linear transformation is applied (i.e., "proc_mean_nlin").

```
function [fit, yhat] = polyfit_monotonic(x, y, order);

%  function [fit, yhat] = polyfit_monotonic(x, y, order);
%
%  Perform a monotonic polynomial fit of column vector x to column vector y and returns
%  the fit and estimated yhat.  The order of the polynomial fit must be >= 1.
%
%  Requires the use of optimization toolbox routine lsqlin.
%

% Catch to make sure order >= 1
if (order < 1)
    disp('Fitting order must be greater than or equal to 1');
end

% Find the size of the column vectors.
n = size(x,1);

% Create x and dx arrays.  For the dx slope array (holds the derivatives of y with respect
% to x, the slope_sign specifies the direction of the slope that must not change over the
% x range.
x_temp = ones(n,1);
dx_temp = zeros(n,1);
slope_sign = 1;

for col = 1:order
    x_temp = [x_temp x.^col];
    dx_temp = [dx_temp col*x.^(col-1)];
end

% The lsqlin routine uses <= inequalities.  Thus, if slope_sign is -1 (negative
% slope), we are correct but if slope_sign is +1 (positive slope), we must
% multiple dx by -1.
if (slope_sign == 1)
    dx_temp = -1*dx_temp;
end

% x fitted to y
fit = lsqlin(x_temp, y, dx_temp, zeros(n,1));
fit = flipud(fit); % organize this fit the same as what is output by polyfit

% yhat calculated
yhat = polyval(fit,x);
```

## A.3 ROUTINE GENERATE_CHART.M

MATLAB routine generate_chart.m can be used to generate color charts with random red, green, and blue color samples. An example MATLAB command line call that was used to generate the color chart in section 4.1 is given by:

- [chart, patch_info] = generate_chart (1704, 2272, 100, 16, 240, 16, 240, 16, 240, 8, 16);

The function returns the color chart ("chart") and a "patch_info" matrix that contains the spatial locations and sizes of the color patches. The "patch_info" is used by routine color_xform to compute the mean of each color patch. The chart generation is controlled by eleven input parameters. The first two input parameters in the above example (1704 and 2272) specify the number of rows pixels and number of columns pixels in the chart image. The third input parameter (100) specifies the number of colors in the chart. Parameters four to nine specify the lowest and highest red (16, 240), green (16, 240), and blue (16, 240) color samples to use. Parameter ten (8) specifies the amount of black border to include around each color patch. Parameter eleven (16) specifies a safety margin to be used when computing "patch_info". This safety margin is useful to eliminate transitions between the different color patches when computing the mean red, green, and blue value of each color patch.

A word of caution is in order. It is possible to generate RGB colors that cannot be printed with the limited color gamut of the cyan, magenta, yellow, and black (CMYK) color space used by many printers. Therefore, it is advisable to generate the RGB chart, use photo manipulation software to convert this RGB chart to the CMYK color space (which is printed for camera testing), and then convert the CMYK chart back to the RGB color space before using it as the original image in routine color_xform.

```
function [chart, patch_info] = generate_chart(nrows, ncols, nrgb, low_red, high_red, ...
    low_green, high_green, low_blue, high_blue, spacing, safety);

% function [chart, patch_info] = generate_chart(nrows, ncols, nrgb, low_red, high_red, ...
%     low_green, high_green, low_blue, high_blue, spacing, safety);
%
% The function generate_chart returns two variables:
%   chart - the generated color chart
%   patch_info - a nrgb X 4 matrix that specifies where each individual color patch is
%                   located within the color chart.  For each color patch, this information is
%                   specified as the first row position, the first column position, the row size,
%                   and the column size, in that order.
%
% The random RGB color chart is generated according to user-specified inputs:
%   nrows - total number of pixels, or rows, in the vertical direction (e.g, 1704)
%   ncols - total number of pixels, or cols, in the horizontal direction (e.g., 2272)
%   nrgb - number of red, green, and blue combinations, or color patches (e.g., 4)
%   low_red - the lowest red value to include (e.g., 0)
%   high_red - the highest red value to include (e.g., 255)
%   low_green - the lowest green value to include (e.g., 0)
%   high_green - the highest green value to include (e.g., 255)
%   low_blue - the lowest blue value to include (e.g., 0)
%   high_blue - the highest blue value to include (e.g., 255)
%   spacing - number of black border pixels between different color patches (e.g., 8)
%   safety - the safety margin (e.g., 16) to add to the patch_info to eliminate transitions
%               between the color patches in the least squares estimate routine.  Safety
%               is added to the patch start locations and 2*safety is subtracted from the
%               patch sizes to provide a safety border all the way around the patch.

chart = zeros(nrows, ncols, 3);  %  Initial chart will contain all black

% The code will compute nrgb color samples randomly spaced from their respective
% low to high values.  These colors are then mapped to a two dimensional color chart.

% Initialize random number generator and generate random color patches
rand('state',sum(100*clock));
rgb = zeros(nrgb,3);  % holds the RGB color of each patch
for i = 1:nrgb
    red = round(low_red-0.5+(1+high_red-low_red)*rand);
    green = round(low_green-0.5+(1+high_green-low_green)*rand);
    blue = round(low_blue-0.5+(1+high_blue-low_blue)*rand);
    rgb(i,:) = [red green blue];
end
```

```
% Distribute patches equally in row and col directions of color chart.  If these values are
% not integers, then the chart will not be completely filled.
nc_p = ceil(sqrt(nrgb));  % number of patches in the col direction, round up
nr_p = sqrt(nrgb);  % number of patches in the row direction (will round as needed)
if (floor(nr_p)*nc_p >= nrgb)
    nr_p = floor(nr_p);
else
    nr_p = ceil(nr_p);
end

sc_p = floor((ncols - spacing*(nc_p+1))/nc_p);  % col size of each patch (in pixels)
sr_p = floor((nrows - spacing*(nr_p+1))/nr_p);  % row size of each patch (in pixels)

%  Contains the first row, first col, row size, and column size of each color patch, in that
%  order.
patch_info = zeros(nrgb,4);

% Fill two dimensional grid from left to right by rows
this_col = 0;
this_row = 1;
for i=1:nrgb  % step through color patches
    this_col = this_col +1;
    if (this_col > nc_p)  % go to next row, this one is full
        this_row = this_row+1;
        this_col = 1;
    end
    index = (this_row-1)*nc_p + this_col;
    beg_col = spacing*this_col + sc_p*(this_col-1) + 1; % beginning col coordinate for patch
    beg_row = spacing*this_row + sr_p*(this_row-1) + 1; % beginning row coordinate for patch
    patch_info(index,:) = [beg_row beg_col sr_p sc_p];
    chart(beg_row:beg_row+sr_p-1, beg_col:beg_col+sc_p-1, :) = ...
        cat(3,repmat(rgb(i,1),sr_p,sc_p),repmat(rgb(i,2),sr_p,sc_p),repmat(rgb(i,3),sr_p,sc_p));
end

%  Add safety margin to the color patch information
patch_info = patch_info + repmat([safety safety -2*safety -2*safety], nrgb,1);

%  Uncomment this line to display the chart
%image(chart/255);

%  Uncomment this line to write out the test chart in tif format.
%imwrite(uint8(chart),'chart.tif','tif');
```

FORM **NTIA-29** U.S. DEPARTMENT OF COMMERCE
(4-80) NAT'L. TELECOMMUNICATIONS AND INFORMATION ADMINISTRATION

# BIBLIOGRAPHIC DATA SHEET

| 1. PUBLICATION NO.<br>TM-04-406 | 2. Government Accession No. | 3. Recipient's Accession No. |
|---|---|---|
| 4. TITLE AND SUBTITLE<br>Color Correction Matrix for Digital Still and Video Imaging Systems | | 5. Publication Date<br>December 2003 |
| | | 6. Performing Organization Code<br>NTIA/ITS.T |
| 7. AUTHOR(S)<br>Stephen Wolf | | 9. Project/Task/Work Unit No.<br><br>3141000-300 |
| 8. PERFORMING ORGANIZATION NAME AND ADDRESS<br>National Telecommunications & Information Administration<br>Institute for Telecommunication Sciences<br>325 Broadway<br>Boulder, CO 80305 | | 10. Contract/Grant No. |
| 11. Sponsoring Organization Name and Address<br>NTIA, Herbert C. Hoover Bldg.<br>14th & Constitution Ave., NW<br>Washington, DC 20230 | | 12. Type of Report and Period Covered |
| 14. SUPPLEMENTARY NOTES | | |

15. ABSTRACT (A 200-word or less factual summary of most significant information. If document includes a significant bibliography or literature survey, mention it here.)

This document discusses a method for correcting inaccurate color output by digital still and video imaging systems. The method uses a known reference image together with a least-squares algorithm to estimate the optimal color channel mixing matrix that must be applied to the output images in order to correct their color inaccuracies. The techniques presented in this document will provide users of digital photography and video equipment with an automated tool for correcting color output. For instance, digital photography users currently may try to correct color distortions in their images by trial and error using photo editing software. However, these correction procedures are time consuming and subjective and do not normally allow for arbitrary mixing of the color channels. The automated color correction matrix computation presented in this document allows each color component in the corrected image (e.g., red) to be calculated as a linear summation of a DC component and all the color components (e.g., red, green, and blue) in the uncorrected image. Methods to correct non-linearities in the color response of digital imaging systems are also discussed.

16. Key Words (Alphabetical order, separated by semicolons)
calibration; camera; color; colorspace; channel; chart; component; correction; digital; matrix; non-linear; sRGB; video

| 17. AVAILABILITY STATEMENT<br><br>UNLIMITED. | 18. Security Class. (This report) | 20. Number of pages |
|---|---|---|
| | 19. Security Class. (This page) | 21. Price: |

# NTIA FORMAL PUBLICATION SERIES

## NTIA MONOGRAPH (MG)

A scholarly, professionally oriented publication dealing with state-of-the-art research or an authoritative treatment of a broad area. Expected to have long-lasting value.

## NTIA SPECIAL PUBLICATION (SP)

Conference proceedings, bibliographies, selected speeches, course and instructional materials, directories, and major studies mandated by Congress.

## NTIA REPORT (TR)

Important contributions to existing knowledge of less breadth than a monograph, such as results of completed projects and major activities. Subsets of this series include:

### NTIA RESTRICTED REPORT (RR)

Contributions that are limited in distribution because of national security classification or Departmental constraints.

### NTIA CONTRACTOR REPORT (CR)

Information generated under an NTIA contract or grant, written by the contractor, and considered an important contribution to existing knowledge.

### JOINT NTIA/OTHER-AGENCY REPORT (JR)

This report receives both local NTIA and other agency review. Both agencies' logos and report series numbering appear on the cover.

## NTIA SOFTWARE & DATA PRODUCTS (SD)

Software such as programs, test data, and sound/video files. This series can be used to transfer technology to U.S. industry.

## NTIA HANDBOOK (HB)

Information pertaining to technical procedures, reference and data guides, and formal user's manuals that are expected to be pertinent for a long time.

## NTIA TECHNICAL MEMORANDUM (TM)

Technical information typically of less breadth than an NTIA Report. The series includes data, preliminary project results, and information for a specific, limited audience.

For information about NTIA publications, contact the NTIA/ITS Technical Publications Office at 325 Broadway, Boulder, CO, 80305 Tel. (303) 497-3572 or e-mail info@its.bldrdoc.gov.

*This report is for sale by the National Technical Information Service, 5285 Port Royal Road, Springfield, VA 22161,Tel. (800) 553-6847.*