Ensuring Color Consistency across Multiple Cameras

Adrian Ilie and Greg Welch
University of North Carolina at Chapel Hill, Computer Science Department
Sitterson Hall, CB# 3715, Chapel Hill, NC 27599, USA
{adyilie,welch}@cs.unc.edu

Abstract

Most multi-camera vision applications assume a single common color response for all cameras. However different cameras—even of the same type—can exhibit radically different color responses, and the differences can cause significant errors in scene interpretation. To address this problem we have developed a robust system aimed at inter-camera color consistency. Our method consists of two phases: an iterative closed-loop ***calibration*** *phase that searches for the per-camera hardware register settings that best balance linearity and dynamic range, followed by a* ***refinement*** *phase that computes the per-camera parametric values for an additional software-based color mapping.*

1. Introduction

Many of the computer vision and computer graphics applications that have emerged during the last decade make use of multiple images. Some applications involve the acquisition of multiple images using a single camera [14, 10, 22]. While using a single camera ensures a consistent color response between images, the approach limits the applicability of these methods to static scenes. Alternatively one can capture dynamic scenes using multiple cameras [12, 25, 23]. However such applications require consistent inter-camera color responses to produce artifact-free results. Figure 2 illustrates some artifacts in a reconstruction produced by an implementation of the 3D reconstruction system in [26].

Unfortunately most cameras—even of the same type—do not exhibit consistent responses. Figure 1 illustrates the differences between the responses of 8 cameras to the 24 colors of the $GretagMacbeth$ [5] $ColorChecker^{\text{TM}}$ chart imaged under the same illumination conditions and using the same hardware settings. The data shows that color values are significantly different from camera to camera. This is due for example to aperture variations, fabrication variations, electrical noise, and interpolation artifacts arising from the reconstruction of a full-resolution color image from a half-resolution Bayer pattern image [3].

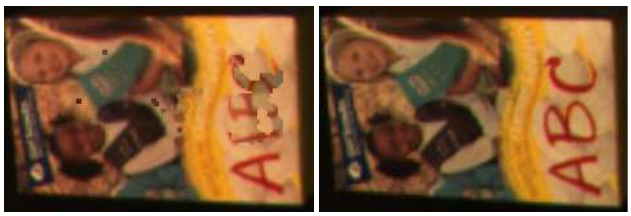

Figure 2. Artifacts in the 3D reconstruction of a physical cookie box. Left: A reconstruction using uncalibrated cameras with the same hardware settings. Right: A reconstruction with color-calibrated cameras. Artifacts are eliminated and the colors are more natural.

To address the color matching problem we have devised a two phase process: an iterative closed-loop *calibration* phase that searches for the per-camera hardware register settings that best balance linearity and dynamic range, followed by a *refinement* phase that computes the per-camera parametric values for an additional software-based color mapping. Variations of these phases have previously been explored separately, however we believe the hardware and software approaches offer complementary benefits that can yield better results when combined. Our goal is to bring the response curves of several cameras closer together and as close as possible to a desired reference image, while also minimizing the amount of noise in the images.

Note that in color science the term *photometric calibration* is typically defined as setting a device to a particular state characterized during a profiling process such as the one described in [16] and later taken into account in order to achieve a desired behavior of the device. In computer graphics, the same term is typically used to describe the process of tuning a general model of the physical device to best describe the specific instance of the device [4]. The first definition corresponds to our iterative closed-loop hardware *calibration* phase, and the second definition corresponds to our software *refinement* phase.

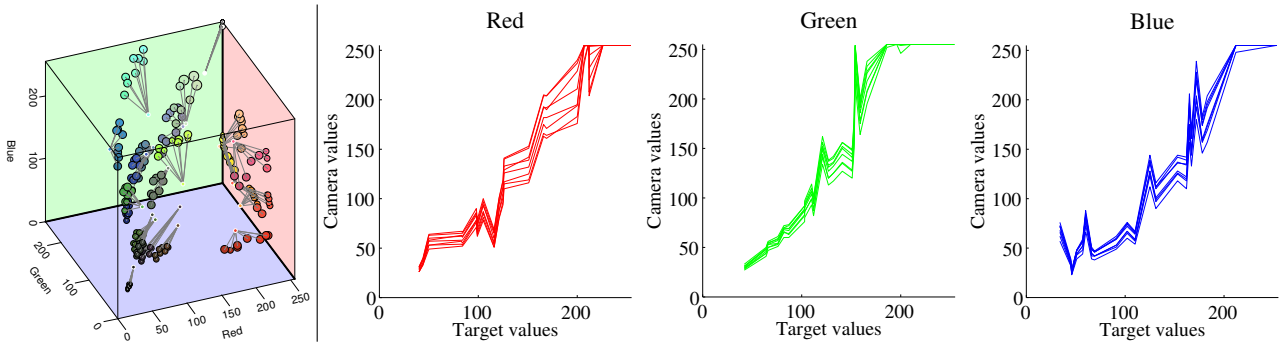

Figure 1. Differences in responses of 8 cameras. Left image: 3D RGB color space plot. Each colored sphere represents the position of a camera sample in the RGB color space. Each connected cluster of colored spheres corresponds to one of the 24 samples in a $ColorChecker^{TM}$ chart. The size of each sphere is proportional to the intra-sample variance. The small white spheres at the origin of each cluster represent the position in the RGB color space of the corresponding target color samples. Right 3 images: The measured color values for each channel, camera and sample, plotted with respect to the corresponding target values. Each individual curve represents samples taken from a particular camera.

2. Previous Work

Previous research aimed at color consistency falls mainly in two categories: calibrating cameras in order to obtain some desired response, and processing images after acquisition. Color consistency has also been studied in the context of projector displays [11], but these techniques have not been extended to camera systems. Other well-known calibration techniques for printers, scanners and monitors are described in great detail in [9].

Calibrating cameras is usually performed with respect to a known target, such as a color chart with standardized samples [13]. Color charts have been traditionally used in photography and color research [2]. The closest work to our method is presented in [8]. They acquire images of a color target, compensate for non-uniform lighting, adjust the gains and offsets of each color channel to calibrate each camera to a linear response, and then apply several software post-processing steps. They also address the scalability of calibrating a large number of cameras by automatically detecting the location of the color target and using special hardware attached to each camera in order to minimize traffic over the camera connections. Although their calibration method is different, their other contributions are applicable to our method as well. We use an approach that minimizes the differences between several camera images while also observing goals such as maintaining visual fidelity and minimizing the signal noise.

Other researchers have proposed the use of scene statistics for single camera calibration [6]. Scene statistics are used in the RingCam [15], a system for capturing panoramas using multiple cameras. They change the brightness and gain of each camera to match the desired "black level" and "mean brightness" values, and to match the image colors in the overlapping regions of adjacent cameras. While these methods have the advantage that they do not require a color chart, they are sensitive to the choices of desired values.

Consistency can also be obtained by software post-processing of images. For example, [19] uses pair-wise correlation for modeling transfer functions and a special distance metric based on image color histograms. While this can produce reasonable results, its complexity increases quadratically with the number of cameras. Also, the transfer functions computed by this approach may introduce distortions and quantization errors when some parts of the color spectrum are compressed or stretched.

3. The Calibration Process

Our method consists of two main phases: an iterative closed-loop hardware *calibration* phase, and a software *refinement* phase. In the first phase we search for the per-camera hardware register settings that best balance linearity and dynamic range. We do this in two steps: first we optimize to a known target (a 24-sample $GretagMacbeth$ [5] $ColorChecker^{TM}$), and then we optimize to the average of the results of the previous step. In the second phase we compute the per-camera parametric values for an additional software-based color mapping. These two phases and the intra-phase steps are depicted in Algorithm 1, and described in more detail in the following subsections.

Algorithm 1 Overall process.

Phase 1: Closed-Loop Calibration of Hardware
identify locations of color samples in the target image
Step 1: optimize to target
for each camera **do**
  identify locations of color samples in the camera image
  repeat
    minimize cost function with respect to target
  until (cost $<$ threshold) **or** (no improvement)
end for
Step 2: optimize to average
repeat
  compute average of all camera images
  designate average as the new target image
  identify locations of color samples in the target image
  for each camera **do**
    if cost is higher than a threshold **then**
      minimize cost function with respect to target
    end if
  end for
until for all cameras (cost $<$ threshold) **or** (no improvement))

Phase 2: Software-Based Refinement
for each camera **do**
  perform software refinement
end for

3.1. Closed-Loop Calibration of Hardware

The basic idea of the calibration phase is to use a general optimizer to search the space of each camera's hardware register values for a state with the closest match between the colors of a target image and the camera image of the color chart. For each camera, the optimizer repeatedly adjusts the register values, acquires an image, and computes the cost. We allow the optimizer to run until all the cameras are close enough to the target image, or until there is no significant improvement from the previous iteration. We actually perform a variation of this procedure twice—in two steps.

In the first step we optimize to an image of the $GretagMacbeth\ ColorChecker^{TM}$ chart acquired with a device whose response is designated as ideal (e.g., another camera, or a scanner). The optimizer cost is computed as a function of the differences in color for a predefined number of samples from each camera image. We compute the differences in RGB color space using either an $L1$ or an $L2$ norm. We also include the intra-window sample variance in the cost function as a way to ensure that our calibration simultaneously minimizes image noise. (Noise will increase with certain poor choices for camera register values.) The resulting formula for the cost function is a weighted sum of the color differences and the intra-window sample variances:

$$C = \sum_{s=1}^{NS} \left(w|\vec{I_s} - \vec{T_s}| + (1-w)V_s \right) \quad (1)$$

where C is the value of the cost function, s is the sample number, NS is the total number of samples, $\vec{I_s}$ is the color of camera image sample s, $\vec{T_s}$ is the color of target image sample s, w and $(1-w)$ are weights (we use $w = 0.5$). Note that colors are 3-element vectors, containing the 3 values for the red, green and blue channels: e.g., $\vec{I_s} = [Ir_s\ Ig_s\ Ib_s]$.

We use square sampling windows of adjustable size, and compute the sample color as an average in each color channel. The intra-window sample variance V_s is computed as

$$V_s = \sqrt{\sum_{i=1}^{WS} |\vec{I_{si}} - \vec{I_s}|^2} \quad (2)$$

where i is the index of each pixel inside the sampling window, WS is the window size, $\vec{I_{si}}$ is the color of pixel i of sample s, $\vec{I_s}$ is the average pixel color over the window.

During the first step each camera will converge on some minimum cost, but the colors in the final camera images are typically still quite different from the target colors. This is not unexpected, as most cameras would be unable to match the ideal target. However in practice when the cameras are of the same type, their response functions (after this first step) are reasonably similar.

In the second step of the hardware-based calibration phase we use the same cost function, but compare with a *new* target image, computed as the average for all cameras of the final sample colors from the previous step. This guarantees that we have not chosen an outlier for the new target, and increases the probability that it can be matched by all the cameras. We repeat the optimization process for all cameras, and then compute a new average target image. We repeat this process until all the cameras are close enough to the latest average target image, or there is no significant improvement. In our experience, a small number of iterations are usually sufficient. Figure 3 shows the final result of our hardware calibration for 8 cameras of the same type.

In both steps of the hardware-based calibration phase (see Algorithm 1) we minimize the cost function iteratively by using a modified Powell's method, adapted from [20]. We chose this method because it is robust to local minima, it does not require the derivatives of the cost function with respect to the input parameters, and it computes the global minimum in a reasonable number of iterations. In our implementation, we use the cost function callback provided by Powell's method to set the current parameter values on the camera, acquire a new image, then compute and return the corresponding cost value. In order to minimize the chance of choosing a local minimum instead of the global minimum, we randomize the starting values of each parameter and the order in which the parameter domains are explored.

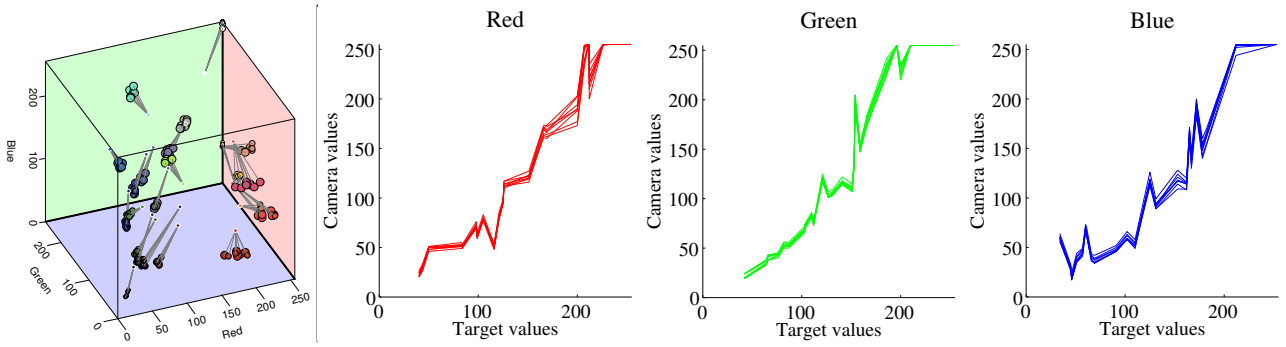

Figure 3. The results of the hardware calibration process. Left image: 3D RGB color space plot. Right 3 images: The measured color for each channel, camera and sample, plotted with respect to the corresponding target values. Compare with Figure 1. The color values in the camera images are still far from the corresponding target values, but they are more consistent (closer together).

3.2. Software-Based Refinement

Hardware settings alone are insufficient to achieve color consistency, because their range and precision are often inadequate. Consequently, a more precise software refinement also needs to be applied to images taken with already calibrated cameras. However, to avoid amplifying noise, clamping and color space distortion errors, we suggest the impact of software refinement should be kept to a minimum. We have explored three different post processing methods to improve our results: linear least squares matching, a 3x3 RGB to RGB linear transform and a general polynomial transform.

3.2.1. Linear Least Squares Matching

The simplest and fastest transform is linear least squares matching. We compute the coefficients a_c and b_c of the best linear transforms that map the camera image color values to the target image color values for color channel $c \in \{R, G, B\}$. We compute the transforms minimizing the following functions in least square sense [24]:

$$\sum_{s=1}^{NS} (\vec{Ic_s} - (a_c \vec{Tc_s} + b_c))^2, c \in \{R, G, B\} \quad (3)$$

Here $\vec{Ic_s}$ is the component for color channel c of camera image sample $\vec{I_s}$, and $\vec{Tc_s}$ is the component for color channel c of target image sample $\vec{T_s}$. Figure 4 (left) shows the effect of the transformation. In effect, we are scaling and translating the color values of each channel independently. This procedure is fast, but often inadequate in the presence of a significant influence from the other color channels. These inter-channel effects are due to several factors. One factor is the fact that the color filter arrays in front of the sensor arrays let in some light from the other channels. Another factor is the specific arrangement of the sensor cells into color arrays, known as the Bayer pattern [3]. In this arrangement, each sensor cell receives only light from one of the red, green or blue (R,G,B) color channels. The cells are arranged into a mosaic composed of 2x2 RG-GB tiles, and the final RGB image is constructed by interpolation using special de-mosaicing algorithms. Some of these algorithms introduce inter-channel effects, noticeable around edges in the image.

3.2.2. RGB to RGB Transform

A common way to account for inter-channel effects is a 3x3 RGB to RGB transform [8]. We compute the 3x3 matrix that best transforms the 24 color samples of a camera image into the corresponding color samples of a target image. The matrix is the solution to the following over-constrained matrix system:

$$\begin{bmatrix} \vec{I_1} \\ \vec{I_2} \\ \dots \\ \vec{I_{24}} \end{bmatrix}_{24\times 3} \times \begin{bmatrix} t_{rr} & t_{rg} & t_{rb} \\ t_{gr} & t_{gg} & t_{gb} \\ r_{br} & t_{bg} & t_{bb} \end{bmatrix}_{3\times 3} \simeq \begin{bmatrix} \vec{T_1} \\ \vec{T_2} \\ \dots \\ \vec{T_{24}} \end{bmatrix}_{24\times 3} \quad (4)$$

This system can be rewritten as the linear system:

$$\begin{bmatrix} \vec{I_1} & \vec{0_3} & \vec{0_3} \\ \vec{0_3} & \vec{I_1} & \vec{0_3} \\ \vec{0_3} & \vec{0_3} & \vec{I_1} \\ \vec{I_2} & \vec{0_3} & \vec{0_3} \\ \vec{0_3} & \vec{I_2} & \vec{0_3} \\ \vec{0_3} & \vec{0_3} & \vec{I_2} \\ \dots & \dots & \dots \\ \vec{I_{24}} & \vec{0_3} & \vec{0_3} \\ \vec{0_3} & \vec{I_{24}} & \vec{0_3} \\ \vec{0_3} & \vec{0_3} & \vec{I_{24}} \end{bmatrix}_{72\times 9} \times \begin{bmatrix} t_{rr} \\ t_{rg} \\ t_{rb} \\ t_{gr} \\ t_{gg} \\ t_{gb} \\ t_{br} \\ t_{bg} \\ t_{bb} \end{bmatrix}_{9} \simeq \begin{bmatrix} \vec{T_1^T} \\ \vec{T_2^T} \\ \dots \\ \vec{T_{24}^T} \end{bmatrix}_{72}$$

$$\Leftrightarrow A \times \vec{t} \simeq \vec{T} \Leftrightarrow \vec{t} \simeq Pinv(A) \times \vec{T} \quad (5)$$

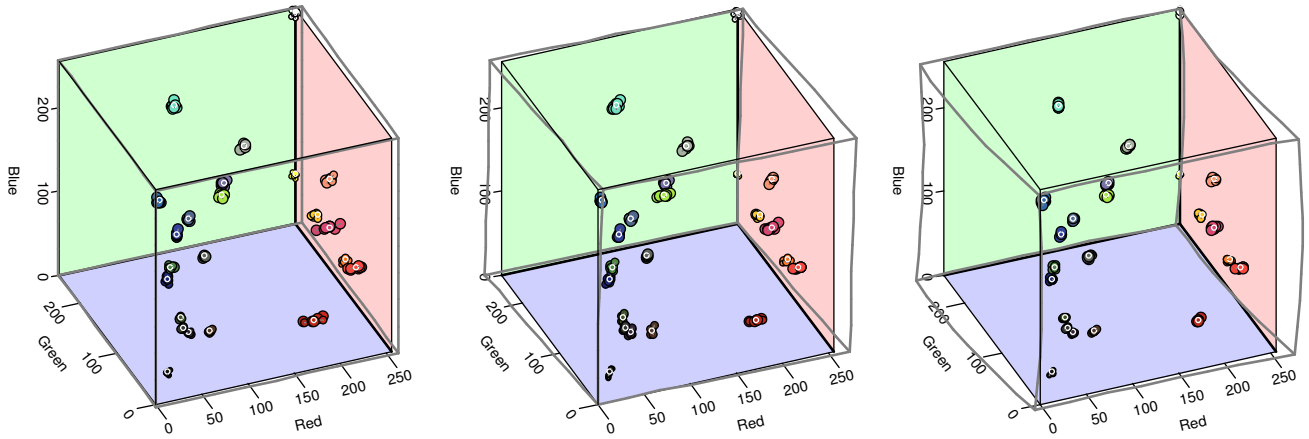

Figure 4. Different types of software refinement. 3D RGB color space plots. Left: Sample colors after linear last squares matching. Middle: Sample colors after applying the RGB to RGB matrix transform. Right: Sample colors after applying the general polynomial transform. The grey outlines show an example of how the color space is distorted by each transform for one of the cameras.

To simplify the notation, we grouped the matrix elements into vectors: $\vec{I_s} = [Ir_s \ Ig_s \ Ib_s]$ is the color for camera image sample s, $\vec{T_s} = [Tr_s \ Tg_s \ Tb_s]$ is the color for target image sample s, and $\vec{0_3} = [0 \ 0 \ 0]$ is a 3-component null vector. t_{xy} is the term that specifies how much the input from color channel x contributes to the output of color channel y. We solve the system using singular value decomposition to compute the pseudo-inverse of matrix A and back substitution to compute the solution $\vec{t}$. Our implementation uses the routines from [20]. Figure 4 (middle) shows the effect of this transform.

3.2.3. General Polynomial Transform

Although the RGB to RGB matrix transform accounts for inter-channel effects, it does not have a translation component and does not compensate for nonlinearities in the response functions. To account for these remaining shortcomings, we have devised a general polynomial transform. We generalize the 3x3 RGB to RGB transform to a nonlinear transform by introducing higher degree terms to compensate for the non-linearities in the response functions and a bias term to allow translations. The general formula for color $c \in \{r, g, b\}$ of sample s is:

$$\sum_{k=1}^{D} \left(t_{rc_k} Ir_s^k + t_{gc_k} Ig_s^k + t_{bc_k} Ib_s^k \right) + t_{c0} \simeq Tc_s \qquad (6)$$

where D is the degree of the polynomial approximation. Ir_s^k, Ig_s^k and Ib_s^k are the red, green and blue values for camera image sample s, raised to power k. Tc_s is the value for color channel $c \in \{r, g, b\}$ of target image sample s. t_{xc_k} is the polynomial coefficient of the k^{th} order term that specifies how much the input from color channel $x \in \{r, g, b\}$ contributes to the output of color channel c. t_{c0} is an additive term that allows translating the output of channel c. Our experiments have shown that $D = 2$ is sufficient to attain the level of precision required by typical applications. For $D = 2$, we can write Equation 6 for all the 24 samples of the color chart in equivalent matrix form as follows:

$$\begin{bmatrix} Ir_1 & Ir_1^2 & Ig_1 & Ig_1^2 & Ib_1 & Ib_1^2 & 1 \\ Ir_2 & Ir_2^2 & Ig_2 & Ig_2^2 & Ib_2 & Ib_2^2 & 1 \\ \dots & & & & & & \\ Ir_{24} & Ir_{24}^2 & Ig_{24} & Ig_{24}^2 & Ib_{24} & Ib_{24}^2 & 1 \end{bmatrix}_{24\times 7} \times$$

$$\begin{bmatrix} t_{rc1} & t_{rc2} & t_{gc1} & t_{gc2} & t_{bc1} & t_{bc2} & t_{r0} \end{bmatrix}_7^T \simeq \begin{bmatrix} Tc_1 \\ Tc_2 \\ \dots \\ Tc_{24} \end{bmatrix}_{24}$$

$$\Leftrightarrow B \times \vec{t_c} \simeq \vec{Tc} \Leftrightarrow \vec{t_c} \simeq Pinv(B) \times \vec{Tc}, c \in \{r, g, b\} \qquad (7)$$

We solve each matrix equation using singular value decomposition to compute the pseudo-inverse of matrix B and back substitution to compute the 3 solutions $\vec{t_r}$, $\vec{t_g}$ and $\vec{t_b}$ Note that matrix B is the same for all 3 color channels, so we only need to perform the inversion once. Our implementation uses the routines from [20]. Figure 4 (right) shows the effect of this transform. Visually, the general polynomial transform gives the best results, yet the amount of distortion (shown by the grey outline) is the largest.

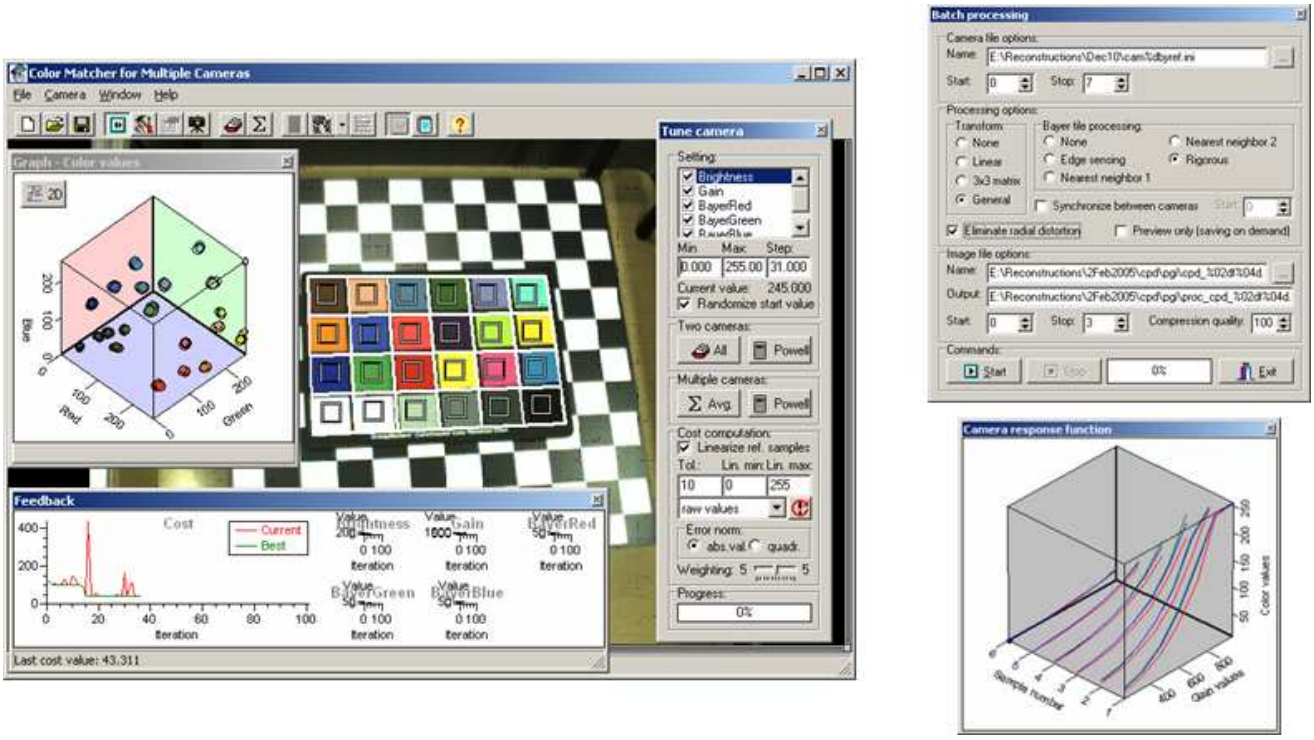

Figure 5. The graphical user interface of the color matching application. Left: The main window, which shows a camera image with the samples highlighted. Also visible are the RGB space representation of all the samples (top left), the hardware optimization settings window (right) and the real time feedback window that shows the progress of Powell's method's cost function during hardware calibration (bottom left). Top right: The batch processing window for applying the software transformations after capture. Bottom right: An example response function graph: color values from the 6 samples of a 6-step gray scale, plotted against the gain of the camera.

4. Implementation and Results

This section describes our calibration application and discusses the results of some of our experiments.

4.1. Application

We have implemented a complete calibration system as an easy to use, stand-alone, extensible application. A few elements of the user interface are shown in Figure 5.

We use an approach similar to the one described in [8] and [23] to automatically detect the locations of the camera image samples: we place the color target at a known location on top of a checkerboard pattern, detect the checkerboard corners in the camera images using OpenCV [7] and employ their positions to compute the location of each sample.

The closed-loop hardware calibration phase is flexible, offering the possibility to choose which hardware settings are tuned, and within what interval. By default, optimization is performed using Powell's method [20]. If time is not critical, the entire hardware settings space can be explored exhaustively, with a specified step in the domain of each setting. The cost can be computed on raw or software-transformed color values, using either an $L1$ or an $L2$ norm and flexible weighting between the color differences and intra-sample variances. The user is given real-time feedback showing the evolution of the color sample values in RGB space and of the value of the cost function. The best hardware setting values are saved in configuration files that are later used during the acquisition process.

The application also allows the visualization of a camera's response function with respect to a chosen hardware setting, as shown in Figure 5 (bottom right). This can provide insight into the limits within which the setting should be constrained during calibration to avoid undesirable effects such as color saturation or excessive noise.

The software refinement phase is performed on demand, and the effect of each transform on the color values can also be visualized as shown in Figure 4. The computed values for the coefficients of the linear, RGB to RGB, and general polynomial transforms are also saved in configuration files that are later used during the post-processing shown in Figure 5 (top right).

The application is written in C++, so extending it by adding new types of cameras and cost metrics is easy by design. Camera hardware settings are mapped to register values, and the mappings are saved in initialization files. Support for other types of cameras can be added by writing subclasses of a base camera class, linking with the appropriate libraries and creating the appropriate initialization files.

4.2. Results

We use 8 FireWire $DragonFly$ cameras and capture libraries provided by PointGrey, Inc. [18] for an implementation of the 3D reconstruction system described in [26], and employ our calibration system to ensure optimal performance.

As shown in Figure 1, even though or cameras are of the same type and we have set their registers to the same values, their responses are quite different and these differences lead to noticeable artifacts in the 3D reconstruction.

For comparison purposes, we first calibrated one camera to a scanned image of the color chart then applied its setting values to all the other cameras. We used a scanned image as our target because even though the chart manufacturer [5] provides color values for the chart we use, the values do not correspond to any color space used in practice [17] and are impossible to match using our cameras. Of all the available hardware camera settings, we used the gain, brightness and per-channel gains during hardware calibration. We chose appropriate values for the other settings and turned off camera features such as auto white balance and auto exposure. Table 1 shows the impact of the hardware calibration process for all cameras.

The inter-sample standard deviation measures how far apart the color samples in the camera images are with respect to each other. This is the error we are trying to minimize, and the hardware calibration accomplishes this task for all 3 color channels. The intra-sample standard deviation measures the level of noise in the camera images. While the noise in the blue channel increases slightly, the noise in the red and green channels decreases significantly.

We then applied the software refinement process. Table 2 shows the impact of the 3 refinement methods we tested (linear least squares, 3x3 matrix transform and general polynomial transform) on the error measured as the mean inter-sample standard deviation for each channel, compared to the values after hardware calibration.

The general polynomial transform performs best according to this error criterion. The 3x3 matrix transform performs worse than the linear transform in this particular case, due to the fact that the 3x3 matrix transform does not have a translation component, and overestimates the inter-channel effects to compensate.

There is a trade-off between the error and the amount of color space distortion a transform induces. Choosing the appropriate software refinement method is dependent upon the application and the scene content. Applications that are more sensitive to differences between camera images and deal with scenes of average colors should choose the general polynomial transform. However, if a scene contains very dark or very bright colors that are already close to the limits of the color space, more distortion can lead to more clamping errors, and the linear method should be chosen instead.

We have also experimented with calibrating configurations of heterogenous cameras. Our first experiment was with a $Flea$ camera from the same manufacturer [18]. The available settings were not the same as for $DragonFly$ cameras: the gain setting for the green channel was missing, but $Flea$ cameras implement gamma correction. We were able to integrate the camera into our application with very little effort, and the result of the calibration was indistinguishable from a $DragonFly$ camera. (Both cameras use the same type of imaging sensor.)

For a second experiment, we used a $DFW-VL500$ camera from $Sony$ [21], which we integrated into our application using the generic capture driver and libraries from [1]. This camera was also missing the gain setting for the green channel, but had many other settings, of which we chose to use brightness, gain, white balance, hue, saturation and gamma correction. Table 3 shows the result of calibrating this camera and one $DragonFly$ camera to the same target.

The errors are approximately 5 times larger than when using cameras of the same type, but within the usability

Table 1. Results of Hardware Calibration

	Channel	Before	After
Mean *inter*-sample st. dev.	R	7.6488	3.0524
	G	6.1958	2.3559
	B	7.9980	3.5444
Mean *intra*-sample st. dev.	R	0.3220	0.2637
	G	0.3000	0.1932
	B	0.2598	0.2695

Table 2. Software Refinement Methods

Channel	Hardware	Linear	Matrix	General
R	3.0524	2.4438	2.3992	1.5170
G	2.3559	1.5098	1.7392	1.0580
B	3.5444	1.7275	1.8078	1.0547

Table 3. Matching of Sony and PointGrey

	Channel	Hardware	Software
Mean inter-sample st. dev.	R	11.1369	5.1854
	G	15.1733	5.8926
	B	12.6101	6.1872

threshold for many applications. We conclude that using cameras of different types is possible, but if high-quality results are desired the best way to obtain them is to use cameras of the same type or at least with the same type of imaging sensor.

5. Conclusion and Future Work

We have shown that it is possible to calibrate several cameras to a known target with high accuracy, which brings their response curves closer together while also minimizing the noise in their images. This enables correlation-based computer vision applications to obtain high quality results. We have presented a complete calibration system in the form of an easy to use, stand-alone, extensible application. Our system implements a two phase process: an iterative closed-loop hardware *calibration*, followed by a single-stage software *refinement*.

The main limitation of our work is that cameras have to be re-calibrated when the lighting conditions change dramatically. While we have not been affected by this problem in our reconstructions, we think re-calibrating may become impractical in some specific circumstances. Re-calibrating without imaging the color chart is not straightforward. We plan to investigate methods that use scene statistics [6] as a way to make incremental adjustments to the cameras to compensate for small changes in lighting.

Another area we plan to explore is more detailed profiling of the cameras. At this time, only the best hardware settings values and the corresponding software transforms coefficients are saved for later use, and only for particular lighting conditions. Profiling the cameras in more detail and under several different lighting conditions may help avoid the need to re-calibrate when the lighting changes.

References

[1] C. Baker. CMU 1394 Digital Camera Driver. `http://www-2.cs.cmu.edu/~iwan/1394/`.

[2] K. Barnard and B. Funt. Camera characterization for color research. *Color Research and Applications*, 27(3):153–164, 2002.

[3] B. Bayer. Color imaging array. US Patent 3,971,065, 1976.

[4] M. Goesele. *New Acquisition Techniques for Real Objects and Light Sources in Computer Graphics*. PhD thesis, Max-Planck-Institut fur Informatik, Saarbrucken, Germany, 2004.

[5] GretagMacbeth Color Management Solutions. `http://www.gretagmacbeth.com`.

[6] M. Grossberg and S. Nayar. What can be known about the radiometric response function from images? In *Proceedings of ECCV*, pages 189–205, 2002.

[7] Intel Corporation. Open Source Computer Vision Library. `http://www.intel.com/technology/computing/opencv/`.

[8] N. Joshi. Color calibration for arrays of inexpensive image sensors. Master's thesis, Stanford University Department of Computer Science, 2004.

[9] Kang, H. *Color Technology for Electronic Imaging Devices*. SPIE-International Society for Optical Engineering, 1997.

[10] M. Levoy and P. Hanrahan. Light field rendering. In *Proceedings SIGGRAPH*, pages 31–42, 1996.

[11] A. Majumder, Z. He, H. Towles, and G. Welch. Achieving color uniformity across multi-projector displays. In *Proceedings of IEEE Visualization*, pages 117–124, 2000.

[12] A. Majumder, W. Seales, M. Gopi, and H. Fuchs. Immersive teleconferencing: A new algorithm to generate seamless panoramic video imagery. In *Proceedings of the Seventh ACM International Conference on Multimedia*, pages 169–178, 1999.

[13] C. McCamy, H. Marcus, and J. Davidson. A color-rendition chart. *Journal of Applied Photographic Engineering*, 2(3):95–99, 1976.

[14] L. McMillan and G. Bishop. Plenoptic modeling: An image-based rendering system. In *Proceedings of SIGGRAPH 1995*, pages 39–46, Los Angeles, CA, 1995.

[15] H. Nanda and R. Cutler. Practical calibrations for a realtime digital omnidirectional camera. In *Proceedings of CVPR, Technical Sketch*, 2001.

[16] M. Nielsen and M. Stokes. The Creation of the sRGB ICC Profile. In *Proceedings of IS&T Sixth Color Imaging Conference: Color Science Systems and Applications*, 1998.

[17] D. Pascale. RGB Coordinates of the Macbeth Color Checker. `http://www.babelcolor.com/main_level/download.htm`, 2003.

[18] Point Grey Research Inc. `http://www.ptgrey.com/`.

[19] F. Porikli. Inter-camera color calibration by cross-correlation model function. In *IEEE International Conference on Image Processing*, volume 2, pages 133–136, 2003.

[20] Press, W. and Teukolsky, S. and Vetterling, W. and Flannery, B. *Numerical Recipes in C: The Art of Scientific Computing, Second Edition*. Cambridge University Press, 1993.

[21] Sony Corporation. Digital Color Camera Module DFW-VL500. Available at: `http://www.sony.net/Products/ISP/products/interface/DFWV500.html`.

[22] R. Szeliski and H.-Y. Shum. Creating full view panoramic image mosaics and environment maps. In *Computer Graphics, 31 (Annual Conference Series)*, pages 251–258, 1997.

[23] Vaish, V. and Wilburn, B. and Levoy, M. Using plane + parallax for calibrating dense camera arrays. In *Proceedings of CVPR*, volume 1, pages 2–9, 2004.

[24] E. Weisstein. Least squares fitting. MathWorld-A Wolfram Web Resource, `http://mathworld.wolfram.com/LeastSquaresFitting.html`.

[25] B. Wilburn, N. Joshi, V. Vaish, M. Levoy, and M. Horowitz. High-speed videography using a dense camera array. In *Proceedings of CVPR*, volume 2, pages 294–301, 2004.

[26] R. Yang. *View-Dependent Pixel Coloring - A Physically-Based Approach for 2D View Synthesis*. PhD thesis, University of North Carolina at Chapel Hill, Computer Science Department, 2004.