Camera Array Calibration with Color Rendition Charts

Studienarbeit

HUMBOLDT-UNIVERSITÄT ZU BERLIN
MATHEMATISCH-NATURWISSENSCHAFTLICHE FAKULTÄT II
INSTITUT FÜR INFORMATIK

eingereicht von: Alexander Behringer
geboren am: 04.01.1987
in: Berlin

Gutachter: Prof. Peter Eisert

eingereicht am:

Contents

1 Introduction

1.1 Problem Description

A common problem, when capturing a scene using multiple cameras, are certain deviations of color reproduction, even if the cameras are of the same type. This is caused by (partly unavoidable) differences in capture settings or by variances of physical properties causing different cameras to never produce perfectly identical results. The definition of camera here includes the lens attached to it, for which the same restrictions apply.

In this thesis an approach developed by the author to compensate for these issues is presented, which works by measuring the color differences between different shots using the color data of a color rendition chart[1] placed in the captured scene to then color correct the images or to correct the camera settings or both.

The objective was to explore and study models and methods to (semi-)automatically detect the color chart within images and to extract the color data from it, as well as to establish a model to explain the color deviations and to correct them.

This thesis is accompanied by a prototype implementation called `caccrc.py` which implements the algorithms described later. It is itself implemented in the Python[2] programming language (version 2.7) using mainly the NumPy[3] and SciPy[4] modules for numerical scientific calculations among other libraries.

1.2 Outline and Overview

This work can be split up into three major section:

First, in Section 2 ("Chart Discovery Algorithm"), the algorithms to locate the color rendition chart within an image and to extract the color data from it are described.

Then, in Section 3 ("Color Analysis and Correction"), the models and algorithms explaining the color differences between different shots and the ones used for compensating for them are presented.

Last, in Section 4 ("Practical Experiments and Results"), the results of applying the described algorithms in form of the prototype implementation `caccrc.py` on test footage and the influence of varying different processing parameters onto the quality of the results are shown.

[1] The color rendition chart used for this entire work was a "Color Checker" chart from X-Rite.
[2] see `https://www.python.org`
[3] see `https://www.numpy.org`
[4] see `https://www.scipy.org`

2 Chart Discovery Algorithm

2.1 Overview

The first objective of the whole algorithm chain is to find a color chart within an image. The idea from the beginning was to look for certain specific 'features' of the chart or its patches, which would allow to distinguish them from the background or other objects in the image.

An initial but unsuccessful try was to look for statistical abnormalities in the histogram of the image, converted to a different color space if necessary, which did not work reliable enough. Another idea, that was considered but not pursued, was to detect edges in the image to reconstruct the charts location. Although this idea was discarded, the edges are still used someway by the current algorithm by analyzing the shape of *regions* or the respective *region masks* described later.

Because each patch is homogeneous in color and texture, the idea ultimately pursued was to look for 'homogeneous' areas within the image and then to 'puzzle' them together to form the chart.

The entire process is divided into the following steps:

- A short preprocessing phase
- A *patch discovery* (Section 2.3) phase further divided into
  - a *region discovery* (2.3.1) phase to find interesting areas within the image and
  - a *region selection* (2.3.2) phase to remove non-relevant areas found so far.
- A *chart discovery phase* (Section 2.4) to 'puzzle' the selected areas together to obtain a chart, further divided into
  - *axes orientation guessing* (2.4.1),
  - *region alignment* (2.4.2),
  - *chart orientation correction* (2.4.3) and
  - a final *coordinate transform calculation* (2.4.4).

The details of these phases will be explained in the following sections.

2.1.1 Related Work

A current paper on the topic discussed in this section has been written by A. ERNST, A. PAPST, T. RUF and J.-U. GARBAS, in which they propose a robust method of finding and tracking a color chart within an image (see [CheckMyChart]). Their method differs from the one presented in this paper, as their approach tries to fit a complete model of the chart onto the image using an general-purpose optimization algorithm adjusting the parameters of a homography

(and a color correction), while the method presented here takes an constructive approach, reconstructing the chart 'bottom-up'.

They as well use the reference colors of the chart, matching them against the colors currently 'masked' by their model using an affine color transform to compensate for illumination and camera effects. The error measure used is the root mean square of the color differences and of the color variance within the masked regions (as they try to exploit the homogeneity of the patches, similar to this work). Color correction was not the focus of their work.

Additionally, their paper contains a comprehensive list of references to other papers on the topic of find color charts in images.

2.2 Preprocessing

The preprocessing phase currently only consists of one operation, namely down-scaling of the image, to improve processing performance, which therefore is optional.

Let *FullImage* be the original full resolution color image, then *DownScaled* should be the rescaled image:

$$DownScaled \;:=\; \text{rescale}(FullImage)$$

The `caccrc.py` currently uses the `scipy.misc.imresize` function with a `"cubic"` interpolation and per default images are scaled-down to have their longer side have a length of 1024 px (the `processing_image_size_bounds` parameter) while preserving the aspect ratio.

2.3 Patch Discovery

2.3.1 Region Discovery

The first objective of the *region discovery* procedure is to find sets of connected pixels within the image called **regions**, which correspond to the perspectively projected and lens-distorted square patches of the color chart. Because the patches are homogeneous in color and texture the idea is to look a the '(color) gradient' of the image or rather its absolute value or magnitude, since the 'direction' of the gradient is not important and at totally flat points not even determinable.

Color Channel "Fusion"

As a preprocessing step the third dimension, the color dimension, is first removed from the *DownScaled* image. The result of that operation is called the *ProcessingImage*. Currently a simple point-wise mean along the color dimension is used:

$$ProcessingImage[row, col] \;:=\; \sum_{c} \frac{DownScaled[row, col, c]}{3}$$

There are of course plenty of other options to do that, for example choosing a single channel (the green one for example) or calculating a weighted mean and of course it is possible to transform the image to a different color space first, but the simple mean worked quite well. An example of a *ProcessingImage* is given in Figure 1a on the following page.

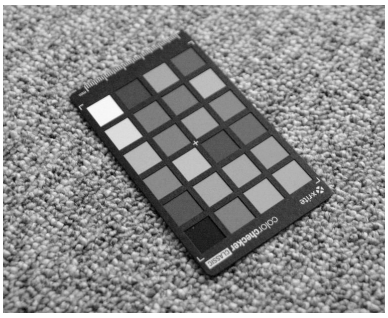
Gradient Analysis

To reduce the influence of high frequency noise onto the gradient, the image is Gaussian filtered first. This is accomplished by convolving the image with a Gaussian kernel. Because of the properties of the convolution operator it is possible to combine the Gaussian filtering and the gradient calculation into a single convolution with a Gaussian-derivative kernel.

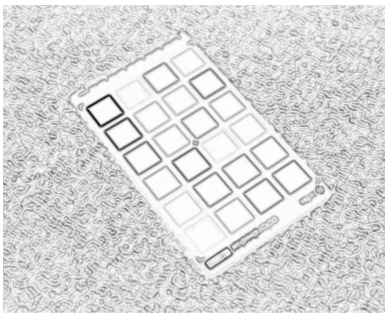
The prototype implementation uses the `gaussian_gradient_magnitude` function that resists in the `scipy.ndimage.filters` module to calculate *GradientMagnitude* from *ProcessingImage*, which is implemented by one-dimensional convolving the image successively with a one-dimensional Gaussian kernel or the derivative of a Gaussian kernel and combining the result as follows:

$$\begin{aligned}
Gradient_{vertical} &= (ProcessingImage *_{vertical} GaussianDerivative) *_{horizontal} Gaussian \\
Gradient_{horizontal} &= (ProcessingImage *_{vertical} Gaussian) *_{horizontal} GaussianDerivative \\
GradientMagnitude &= \sqrt{Gradient^2_{vertical} + Gradient^2_{horizontal}}
\end{aligned}$$

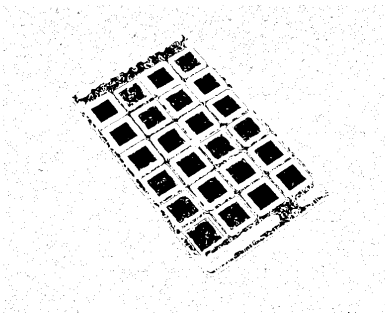
where $*_{vertical}$ and $*_{horizontal}$ denotes the convolution of a two-dimensional image which a one-dimensional kernel along the vertical and horizontal image dimension, respectively. *Gaussian* is a sampled one-dimensional Gaussian curve with its length truncated to approximately four



(a) The *ProcessingImage* resulting (among other things) from color averaging the original input image (*FullImage*).



(b) The *GradientMagnitude* of the *ProcessingImage*. Dark regions have non-zero gradient magnitude.



(c) Image (*Homogeneous*) of the pixels considered 'flat' (dark color).

Figure 1: Examples of the different intermediate results of the patch discovery algorithm.

times the standard deviation. Likewise *GaussianDerivative* is the sampled first derivative of the Gaussian curve.

Per default the Gaussian curve used has a standard deviation of 0.002 (the `gradient_sigma` parameter) times the image width, which is approximately 2 px with default settings and a landscape image.

An Example of a *GradientMagnitude* is given in Figure 1b.

Homogeneous Pixels

Because pixels associated with a patch are assumed to have a small gradient magnitude, these will now be selected using a simple threshold operation:

$$Homogeneous \;:=\; (GradientMagnitude < threshold)$$

Homogeneous masks those pixels for which the gradient is considered 'flat' at their location. An example of how *Homogeneous* looks like is given in Figure 1c. How the *threshold* and the other processing parameters can be tuned is demonstrated in Section 4.2. Per default the *threshold* is set to 0.5 (the `gradient_threshold` parameter).

Since isolated pixels are of no use the next step is to look for sets of connected homogeneous pixels the next step is to find these sets. By considering two pixel to be connected when they share a common edge now the *connected components* of *Homogenous* are enumerated, assigning each pixel the 'number' of its component. This is called *labeling*. Let $Label[row, col]$ be the label of the pixel at (row, col). The background (all pixels not considered 'flat') is assigned the integer 0 while all other labels are positive integers.

The prototype implementation uses the `label` function of the `scipy.ndimage.measurements` module to label the *Homogene* image.

2.3.2 Region Selection

Since not all regions found so far are associated with a patch of the color chart, those who are have to be selected by some criteria. Regions corresponding to patch are called *valid*, the others are called *invalid*.

The process of selecting the valid regions and discarding the invalid is separated into multiple phases:

- ▻ First the regions are filtered based on 'local' features (*preselection phase*).
- ▻ Then statistics about the preselected regions are collected (*statistics phase*).
- ▻ At last the preselected regions are compared against each other and filtered based on the gathered statistics (*selection phase*).

Preselection Phase

In the *preselection phase* first all regions consisting of to few or to many pixels are removed. The upper bound is chosen based on the maximum area a patch of the chart can reasonably occupy within the image assuming that the chart is still completely visible. Similar the lower bound is chosen based on the assumption that the chart will occupy some minimal fraction of the image.

The default for the lower bound is 0.0005 (the `minimum_relative_patch_area` parameter) times the image area (width times height) and the upper bound is 0.02 (the `maximum_relative_patch_area` parameter) times the image area.

Now each region, not already filtered out by some of the previous criteria, is assigned four vertices. This is done by assuming the region is a rasterized quadrilateral. The algorithm used is explained in Section 2.3.3 on page 11. The convex area covered by the quadrilateral spanned by the found four vertices is called a **region mask**.

All following selection criteria are based on the region mask. The following criteria are currently implemented:

- ▻ Distance of the four vertices of the region mask to the image border:

  If at least one vertex is close to the border it is very likely the the region is part of something which has been cropped by the camera frame and is thus unlikely to be patch of the color chart. Even if it was, it would be considered invalid.

  Per default a vertex is considered to be to close to border, if it has a vertical or horizontal distance of less then 0.03 (the `border_distance_threshold` parameter) of the image width or height to the image border, respectively.

- ▻ Parallelism of the four sides of the region mask:

  The projected patches should be nearly perfect parallelograms. A region mask does not comply with this criterion, if the difference of opposite side length is more than 0.15 (the `parallelism_threshold` parameter) from the average of both side lengths.

- ▻ Aspect ratio of the region mask:

  Because the camera normally looks nearly orthogonal onto the chart, the aspect ratio of the patches should be near one. The aspect ratio is defined as the quotient of the the sum of the lengths of opposite sides. Per default a region mask violates this criterion, if its aspect ratio is smaller than 0.5 (the `aspect_ratio_tolerance` parameter) or larger than the inverse of that, which is 2.0.

- ▻ Similarity of the region with the region mask:

  The projected patch should look like a quadrilateral. By comparing the rasterized region mask with the actual region, it can be checked if it has approximately quadrilateral shape. Per default a region mask violates this criterion, if more than 0.175 (the `mask_coverage_tolerance` parameter) of the pixels of the rasterized region mask differ from the pixels of the region. Note that both are binary images and indicate, if a pixel belongs to the region mask or the region, respectively.

Of course other criteria are possible and easily implementable, but the reference implementation confines itself to aforementioned ones.

Statistics Phase

Assuming that most of the regions selected in the preselection phase are valid, the idea is now to gather statistics to find the remaining outliers, which are assumed to be invalid patches.

As with the preselection criteria there are plenty of options, but the reference implementation currently uses only one criterion, which is the sides lengths of the region masks. The side lengths of the edges are first sorted, each region mask a time, and the the median for the longest, the second longest and so forth side is calculated.

These statistics are used in the final selection phase which follows next.

Selection Phase

The objective in the *selection phase* is to remove the remaining invalid patches and thus to select the regions used for further processing. Since the *chart discovery* process currently cannot handle invalid regions, it is important to remove all of them. It is however acceptable to discard a few valid regions as long as one region per column and one per row remain.

Since the prototype implementation only gathers statistics on the side lengths of the region masks during the statistics phase, the only filtering step possible is based on the comparison of the side lengths, sorted by length, of each region against the median and to remove the regions which deviate to much. The region is discarded, if one of its side lengths deviate more then 0.15 (the `side_length_tolerance` parameter) times the respective median side length from the respective median side length.

Figure 2 on the following page shows an example of all found regions and the criteria of their removal.

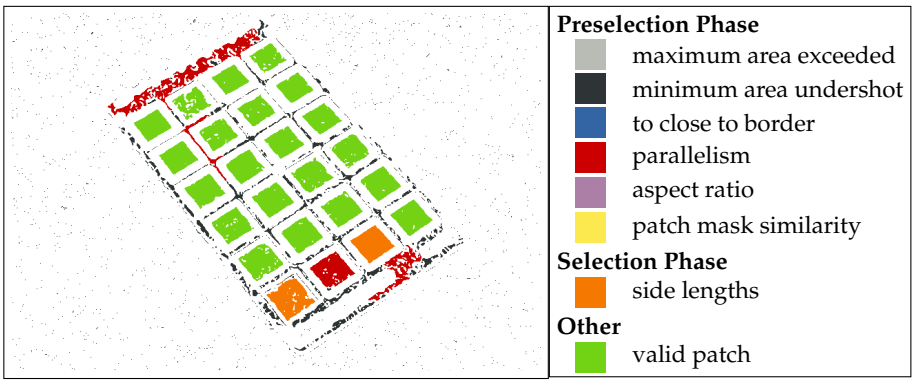


Figure 2: Shown are all regions in the color of the first criteria, which matched the region and led to their removal (except of course for "valid patch").

2.3.3 Rasterized Quadrilateral Vertices Discovery

In this section the algorithm used to assign each region four vertices to form the region mask is described.

Under the assumption that the camera or its lens respectively applies a perspective projection onto the chart, all patches will be transformed into convex quadrilaterals at the image plane and therefore all regions corresponding to patch are the result of perspectively projecting and rasterizing square patches. The goal of the algorithms described here is to find the locations of the vertices of these rasterized quadrilaterals by looking at the shape of each region.

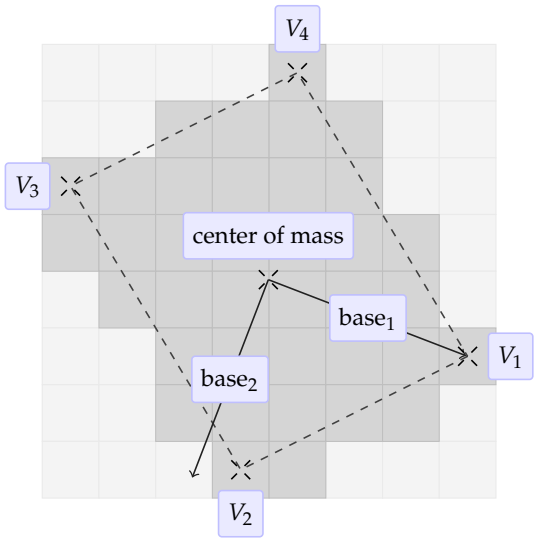


Figure 3: Finding the vertices of a rasterized quadrilateral.

The mathematical objects defined subsequently are shown in Figure 3.

First the *center of mass* C of the region is calculated. In SciPy this can be accomplished easily with the `center_of_mass` function in the `scipy.ndimage.measurements` module.

Next the distance of every pixel of the region to that center is calculated and the farthest pixel is chosen to be the first vertex V_1.

Now an auxiliary orthogonal affine coordinate transform with origin C and first base vector $\overrightarrow{CV_1} =: (b_1, b_2)$ is constructed. The other base vector may be $(-b_2, b_1)$ for example.[5]

[5]The two base vectors should have the same length, but the coordinate transform may but does not have to be orthonormal.

Next, the coordinates of all pixels of the region are recalculated relative to this new coordinate system. Assuming row vectors, this is simple matrix inversion and multiplication:

$$\begin{pmatrix} — newcoords_1 — \\ \vdots \\ — newcoords_n — \end{pmatrix} = \begin{pmatrix} — coords_1 — \\ \vdots \\ — coords_n — \end{pmatrix} \cdot \underbrace{\begin{pmatrix} — base_1 — \\ — base_2 — \end{pmatrix}^{-1}}_{M^{-1}}$$

Since the coordinate system is orthogonal (and thus the transformation matrix M) inverting M is equal to transposing M:

$$M := \begin{pmatrix} b_1 & b_2 \\ -b_2 & b_1 \end{pmatrix} \quad \Rightarrow \quad M^{-1} = M^T = \begin{pmatrix} b_1 & -b_2 \\ b_2 & b_1 \end{pmatrix}$$

The remaining three vertices V_2, V_3 and V_4 are chosen by selecting the farthest pixels along the other three directions using the following three simple distance functions d_2, d_3 and d_4:

$$\begin{aligned} d_2(c_1, c_2) &:= c_2 \\ d_3(c_1, c_2) &:= -c_1 \\ d_4(c_1, c_2) &:= -c_2 \end{aligned}$$

This choice of base vectors and distance functions in this order guaranties that the vertices will be in clockwise orientation.

2.4 Chart Discovery

The patch discovery procedure described in the previous section provides a most likely incomplete and possibly erroneous set of regions in form of the vertices of their region masks. Before a coordinate transform capable of uniquely addressing each patch within the image can be calculated, these regions have to be brought into a grid-like order. Apart from the problem of missing regions and additional erroneous regions not corresponding to a patch but some other object in the picture, this means the alignment of the regions and the orientation of the chart have somehow to be guessed.

The process, constructing the chart form the region data, is called *chart discovery* and consists of first compensating for small rotation (meant is a rotation up to ±45°) of the chart by looking at the orientation of the region edges (Axes Orientation Guessing). This brings the chart in an upright position, which allows row and column numbers to be assigned to each region (Region Alignment). From that an *intermediate coordinate system* can be calculated. Then certain errors like coarse rotation (by 90°, 180°, 270°), flipping and shifting of the chart are corrected (Chart Orientation Correction) and the desired coordinate transform is calculated (Final Coordinate Transform).

The whole processing chain is shown in figure 4.

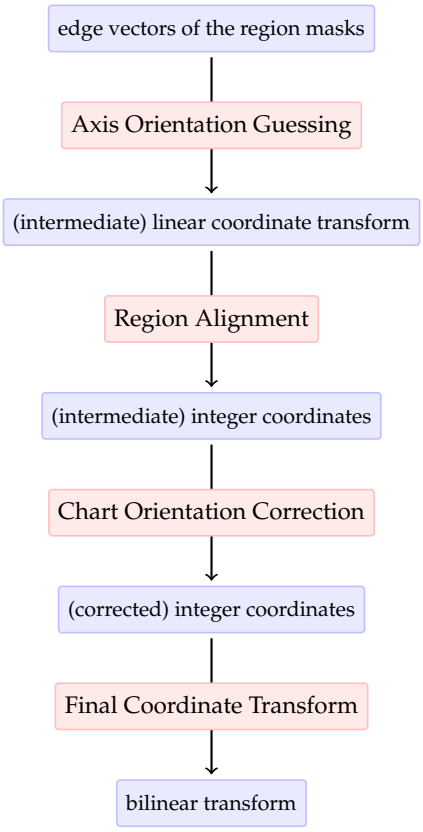


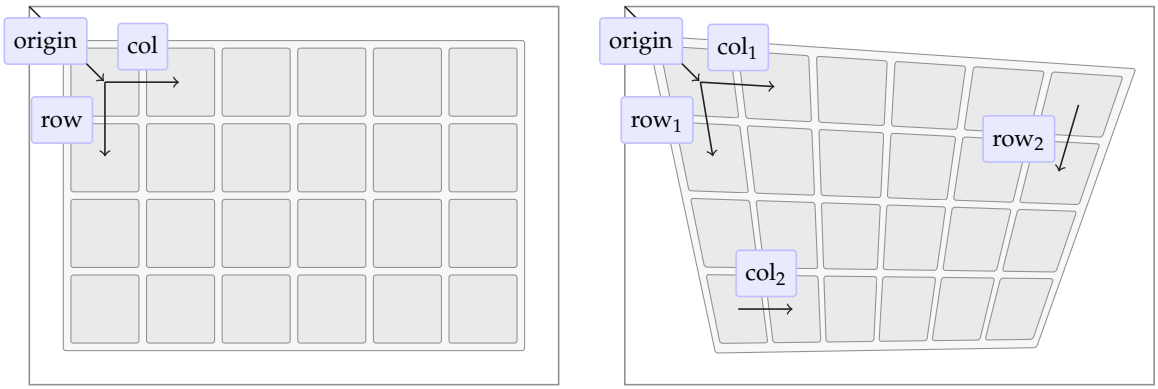
Figure 4: Chart Discovery Algorithm

2.4.1 Axes Orientation Guessing

Because the patches of the chart are aligned in a grid, they can be addressed using an affine coordinate transform using only integer coordinates (Figure 5a on the following page). The projection caused by the camera and the lens respectively transforms that into a bilinear coordinate system[6] (Figure 5b on the next page).

Thus the selected regions, assuming they all correspond to patches, 'sit' on a bilinear grid addressable by integer coordinates as well. Since the perspective distortion of the chart should be relatively small, the problem of just finding the grid can be simplified by using an affine coordinate transform as an reasonable approximation of the bilinear one.

[6] It is actually a homography, but a bilinear transformation should be a sufficient approximation.



(a) The patches of the color chart are addressable using an affine coordinate system using only integer coordinates.

(b) In the image the affine coordinate system becomes a bilinear coordinate system. Please note that four vectors would suffice to describe the transform instead of the displayed five (for example by replacing row_2 and col_2 with their sum $\text{row}_2 + \text{col}_2$).

Figure 5: Coordinate Systems

First the orientation of the axes of the affine coordinate transform, that is the direction but not the length of *row* and *col*, is guessed from the edges of region masks. Because the edges of the patches and the thus the edges of the regions masks are parallel to the axes of the coordinate system, the axes orientation can be estimated using the later. The edges of the region masks are shown as arrows in Figure 6a.

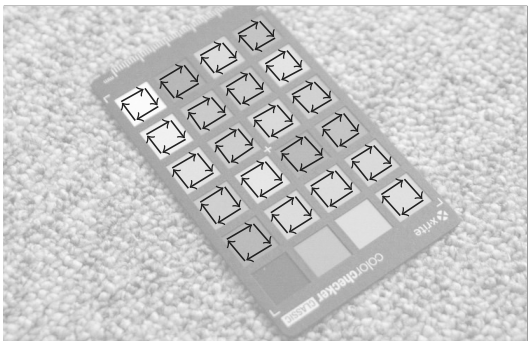
But there is still one problem: Although the "rasterized quadrilateral vertices discovery algorithm" described in Section 2.3.3 guarantees that the vertices and thus the edges of all region mask are always aligned clockwise, there is no guarantee that the 'first' edge of one region mask is on the 'same side' as the 'first' edge of another region mask.

The solution chosen here is to apply four-means clustering on the set of all edge vectors (shown in Figure 6b) which generates four vectors, each one parallel to one side of all region masks. Since only two vectors are needed, the two selected vectors are combined with their respective negated opposite ones by averaging.

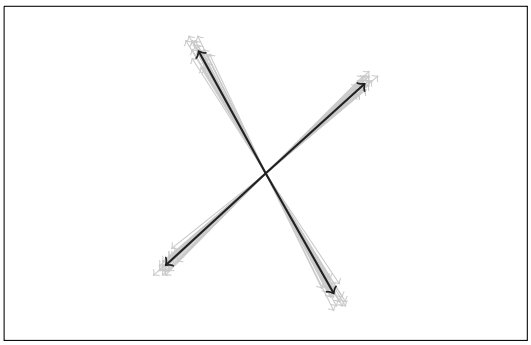
2.4.2 Region Alignment

Although the previous step generates the correct orientation of the base vectors of the affine coordinate transform, it does not provide information about the correct base vector length nor about the origin vector. Therefore the lengths are temporarily set one pixel and the origin is set to be located at the image origin (effectively a linear transform). Now the coordinates of the region masks centers are calculated according to this coordinate system. This is simply a rotation of the coordinates in pixel space to align the region masks edges with the axes of the image coordinate system.

By looking at either the row or the column coordinates of the region mask centers, which is



(a) The edge vectors of the region masks of the selected regions shown as arrows at the location of the respective edge.



(b) In gray the set of all region mask edge vectors and in black the means of all vectors within one of the four clusters.

Figure 6: The base vectors of the intermediate coordinate system are obtained by clustering all region mask edge vectors into four clusters and then combining two of the cluster means with the respective negated opposite cluster mean by average the two.

equal to a projection onto either axis, it is clearly visible, that the coordinates from clusters (see 7a on page 17). Every cluster belongs to one row or one column of the chart, so by assigning integers to the clusters it is possible to address each region by integer row and column coordinates, which is the desired result.

Special care has to be taken if complete rows or columns of regions are missing. This could be compensated for by looking at the distance between adjacent cluster (i. e. adjacent rows or columns respectively) to detect gaps. The prototype implementation currently does not handle missing rows or columns, but at least there is no problem as long as there is at least one region per row and column.

Now the complete intermediate affine coordinate transform, including the origin vector, is calculated by fitting a affine coordinate transform to the region centers locations and coordinates we have now. A least square solution can be found by expressing the region center locations and coordinates as homogeneous coordinates, which gives an overdetermined linear equation, and by calculating a Moore-Penrose pseudoinverse and thus solving the following equation:

$$\begin{pmatrix} locRow_1 & locCol_1 & 1 \\ \vdots & \vdots & \vdots \\ locRow_n & locCol_n & 1 \end{pmatrix} = \begin{pmatrix} coordsRow_1 & coordsCol_1 & 1 \\ \vdots & \vdots & \vdots \\ coordsRow_n & coordsCol_n & 1 \end{pmatrix} \cdot TransformMatrix$$

But these coordinates might still be wrong, because of missing first or last rows or columns or because the chart is flipped or rotated in the image. This cannot be corrected using the information available at this stage. Instead the colors of the pixels covered by the region masks have to be taken into account, which is the topic of the next section.

2.4.3 Chart Orientation Correction

Although the intermediate coordinate transform calculated in the last section allows to address the patches by integer coordinates, these coordinates might sill be wrong, because the chart could be rotated (by 90°, 180° or 270°) or flipped[7]. The coordinates could as well be shifted because of missing first or last rows or columns, but the reference implementation currently does not regard that.

These errors cannot be corrected using just geometric information, since all patches have the same shape. Therefore the colors of the pixels of each region have to be taken into account by comparing them to some reference data[8]. The idea is to generate all possible (flipped, rotated, shifted) coordinate transforms and to choose the one with least color error while compensating for the illumination.

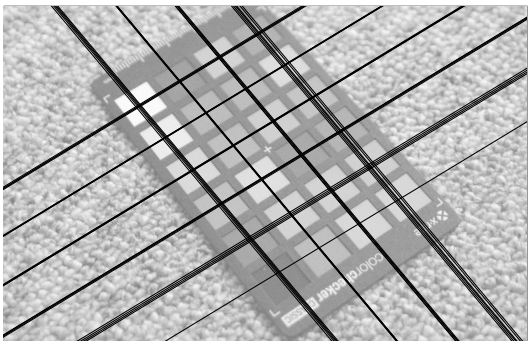
`caccrc.py` partly corrects the rotation by simply making the larger dimension the column dimension. Then it is checked, if reversing one or both dimension gives a better color match, which means currently four possibilities are checked. At the same time, shifting caused by missing first or last rows or columns could be tested for, which is however currently not implemented.

The current prototype implementation uses an affine color transform for illumination compensation, which is not suitable for color correcting the images later, but serves the purpose of finding the correct coordinate transform, and chooses the coordinate transform with the smallest color error.

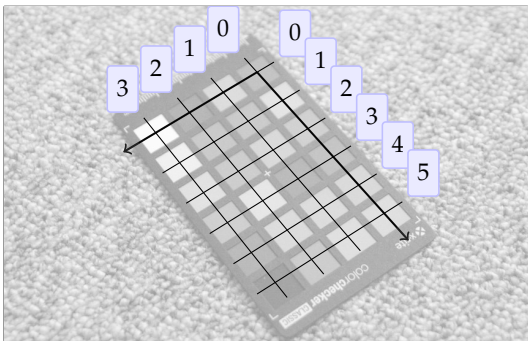
The calculation of this color transform is done exactly like the calculation of the affine coordinate transform in the previous section, just using colors instead of locations. This transform is then applied to the image colors and expressed a non-homogeneous coordinates. Then these colors are subtracted from the reference colors and the root-mean-square of all color components is calculated as the error measure.

[7] Although a flipped chart is due to some 'error' in the image processing chain, because the backside of the chart has no patches. But since the algorithms might flip the chart by swapping row and column coordinates, this is nonetheless checked for.

[8] The reference data for the "Color Checker" chart can be downloaded from the X-Rite website at `http://xritephoto.com/ph_product_overview.aspx?ID=1192` in form of the PDF `http://xritephoto.com/documents/literature/en/ColorData-1p_EN.pdf`.



(a) Shown are crosses through every region center with both cross lines being parallel to the axes of the intermediate coordinate system. Clustering is clearly visible which allows to assign row and column numbers to each region.



(b) The final bilinear coordinate transform.

Figure 7

2.4.4 Final Coordinate Transform

In the last step a final coordinate transform is generated by 'fitting' a bilinear coordinate transform onto the locations of the region centers, simply calculated by averaging the region mask vertices, and the region coordinates, which were calculated and corrected in the previous steps.

The prototype implementation uses the `scipy.optimize.leastsq` function, which in turn uses the MINPACK's "lmdif" and "lmder" algorithms, to 'fit' the bilinear coordinate system onto the data. The definition of bilinear coordinate transform currently used by `caccrc.py`, which might differ from depiction shown early, is defined as follows:

$$location(row, col) = origin + row \cdot row_2 + col \cdot col_2 - row \cdot col \cdot (row_2 - row_1 + col_2 - col_1)$$

with $col_2 = (col_1 + row_2 - row_1)$.

An example for the final coordinate transform is depicted in figure 7b.

3 Color Analysis and Correction

3.1 Overview

In this section a method is demonstrated how to use the gathered data to correct the colors of a captured image or rather how to align the colors of one image to another.

Therefor first a model for how a camera transforms light into color is proposed in Section 3.2, followed by a model for color correction in Section 3.3.

3.1.1 Related Work

The camera model, presented in the next section, is based on the author's experience with colorimetry and spectrometry. The author considers the book of WYSZECKI and STILES the standard work on that and similar topics (see [ColorScience]).

The idea of using a scaling or a linear transform based color correction model and especially the use of the EMoR model of response is inspired by the author's use of the Hugin panorama creation software (see [Hugin]), which implements a similar model for color correction to seamlessly stitch sets of images together to form a panorama. This model is briefly described in PABLO D'ANGELOs paper "Radiometric alignment and vignetting calibration" (see [HuginPaper]).

E. REINHARD ET AL. have written a paper on the topic of aligning the colors of one image to another, which they call "color transfer". Their method is based on transforming the image from RGB space (implicitly assuming a certain color space) into the LMS space (the space modeling the reactions of the cones in the human retina). Then they apply a log transform and then further transform the colors into the $l\alpha\beta$ space proposed by RUDERMAN ET AL.. The the colors are then shifted and scaled based on the mean and standard deviation of the target and the reference images and then transformed back into the RGB scpae. All transforms, except for the log part, are, as far the author can tell, linear or affine.

A different method is presented in paper "Color Transfer in Correlated Color Space" of X. XIAO and L. MA (see [ColorTransfer2]). Their method works in RGB space by constructing an affine transform based on the Singular Value Decomposition of the covariance matrices of the target and reference images.

3.2 Camera Color Model

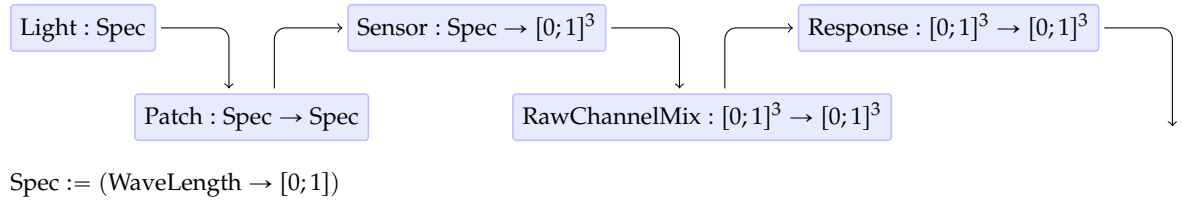


Figure 8: A general camera color model in form of chain of functions.

Analog Processing

First, it is assumed that the color chart is homogeneously illuminated by some light source with a spectral distribution modeled by the function $\langle \text{Light} \rangle$ mapping wavelength onto intensity:

$$\text{Light} : \text{Wavelength} \rightarrow [0; \infty[$$
$$\text{Light}(\lambda) \triangleq \text{“intensity of wavelength } \lambda\text{”}$$

The symbol Spec will be used as an abbreviation for such a function which maps Wavelength onto Intensity:

$$\text{Spec} := (\text{Wavelength} \rightarrow [0, \infty[)$$

The light is reflected by each patch differently depending on its reflectance. The function $\langle \text{Patch}_i \rangle$ maps the initial light spectrum onto the reflected spectrum:

$$\text{Patch}_i : \text{Spec} \rightarrow \text{Spec}$$
$$\text{Patch}_i(\lambda) \triangleq \text{“intensity of wavelength } \lambda \text{ of the reflected light”}$$

The light will pass through the lens and the sensor color filter, hit the sensor, be transformed into an electric charge, amplified and digitized into three raw values[9] for each image pixel. All effects, if they depend on wavelength or not or if they are linear or not, are modeled by the $\langle \text{Sensor} \rangle$ function:

$$\text{Sensor} : \text{Spec} \rightarrow [0; 1]^3 = \begin{pmatrix} [0; 1] \\ [0; 1] \\ [0; 1] \end{pmatrix} \qquad \text{Sensor} \triangleq \begin{pmatrix} \text{“raw signal of ‘red’ pixels”} \\ \text{“raw signal of ‘green’ pixels”} \\ \text{“raw signal of ‘blue’ pixels”} \end{pmatrix}$$

In the color correction model described later we will in fact assume that the $\langle \text{Sensor} \rangle$ function is linear which is not true for real sensors due to non-linear effects in the sensor such as saturation.

[9] Assuming a sensor with three different filter colors.

Digital Processing

Now, in the digital domain, the camera has to perform (at least) two additional tasks: It has to adjust the white point to match those of the light source (or some predefined white point) and to apply or correct a certain non-linear response (often called "gamma").

In the following it will be assumed, that the white point correction is a simple linear transform $\langle \text{RawChanelMix} \rangle$ through multiplication of each color triple with a 3x3 matrix:

$$\text{RawChannelMix} : [0;1]^3 \to [0;1]^3$$
$$\text{RawChannelMix} \begin{pmatrix} r_{raw} \\ g_{raw} \\ b_{raw} \end{pmatrix} := \begin{pmatrix} rr & gr & br \\ rg & gg & bg \\ rb & gb & bb \end{pmatrix} \cdot \begin{pmatrix} r_{raw} \\ g_{raw} \\ b_{raw} \end{pmatrix}$$

The $\langle \text{RawChannelMix} \rangle$ function also is 'responsible' for intensity differences due to different aperture, sensor sensitivity ("ISO") or exposure time settings.

The last step in the camera model is the application of a response curve (correction) via the $\langle \text{Response} \rangle$ function, which is assumed to be a monotone function:

$$\text{Response} : [0;1]^3 \to [0;1]^3$$

How the response function is models is described in Section 3.3.1.

3.3 Color Correction Model

The objective of the color correction step is to let the colors look as equal as possible across a set of cameras of an array by adjusting the colors of all cameras to match one *reference camera* of choice. The reference camera can be a virtual camera, which only exists as a model, or a real camera in the array.

It is assumed that the reason for color errors originate in differences of:

- ▻ the sensor characteristics (including its color filter array) modeled by the ⟨Sensor⟩ function,
- ▻ the used in-camera color transform including different aperture, sensor sensitivity ("ISO") or exposure time settings represented by the ⟨RawChannelMix⟩ function or the used matrix (assuming a linear transform) or
- ▻ the applied camera response curve represented by the ⟨Response⟩ function.

The perfect correction of these color errors would require the knowledge all model parameters and the model to perfectly characterize the camera, which is beyond the possibilities of this thesis' approach, except maybe for some virtual reference camera models which have defined behavior.

The response function and the mix of raw channels of the camera and the reference camera (represented by Response_{main}, $\text{RawChannelMix}_{main}$, Response_{ref} and $\text{RawChannelMix}_{ref}$) will simply be guessed from the color data.

An overview over the color correction model is depicted in Figure 9 on the next page.

If the reference camera is real camera of the array and the array consists of identical cameras set to the same settings, most of the parameters should be quite similar (indicated by the dashed lines - - -). Other parameters are assumed to or intended to be equal (indicated by the solid lines ——).

3.3.1 Camera Response Models

Since only one color sample per patch is extracted, only a handful samples are available to guess the response function from. Moreover, the only restriction for that function is that it has to be monotone and maps the range (normalized to $[0; 1]$) of allowed amounts of light falling onto the sensor (input irradiance) onto the possible image brightnesses (normalized to $[0; 1]$ as well). Beyond that, it could have any shape and thus an unlimited number of parameters. But with only a few color samples at hand, only a limited number of parameters can reasonably be guessed. This is aggravated by the fact, that the color correction parameters will be guessed at the same time, which increases the total number of parameters even further. So the (more or less obvious) idea is to use a model for camera response, which only has a small number of parameters.

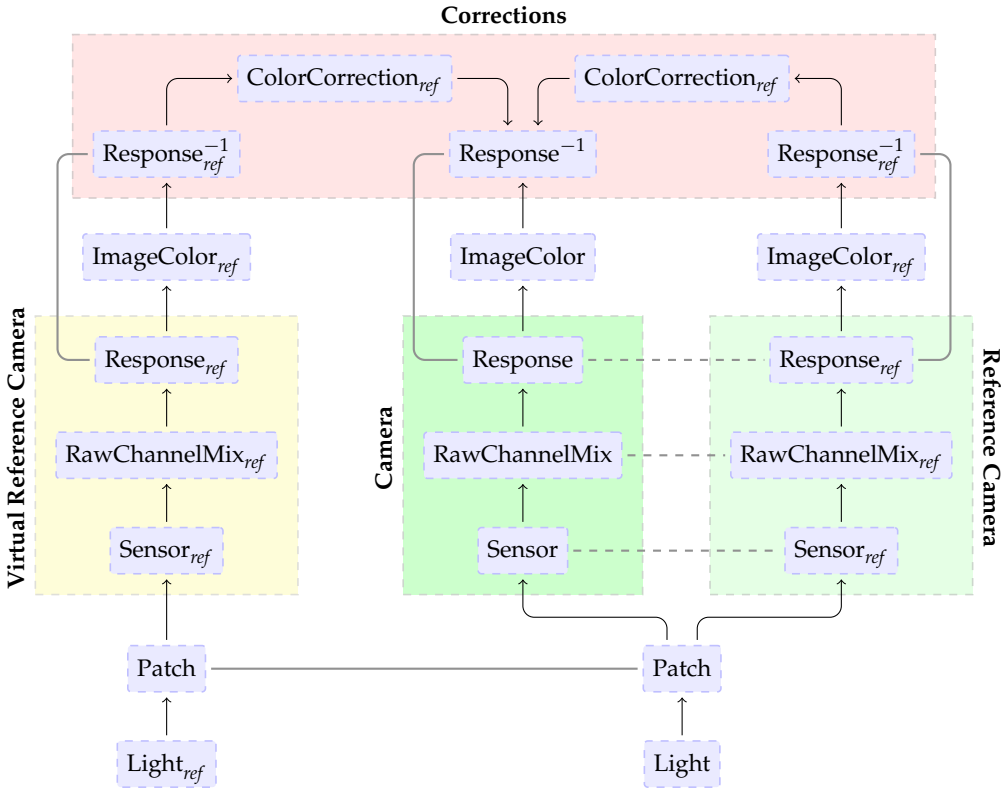


Figure 9: Shown are the functions involved in 'creating' and correcting image colors. Parameters, which ideally would be equal, but most likely won't, are joined by dashed lines and parameters, which assumed or intended to be equal, are joined by solid lines.

A possible very simple model would be a gamma function like

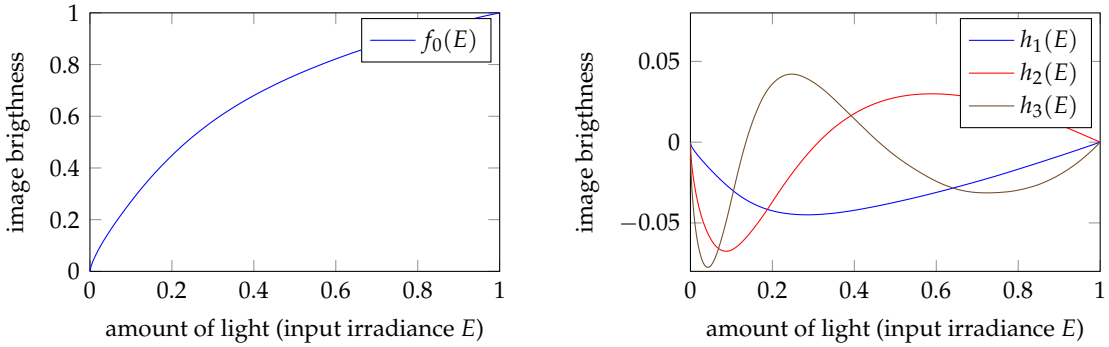
$$\text{brightness}(\mathit{irradiance}) := \mathit{irradiance}^{\gamma}$$

which has only one parameter and is often used as the response function in defined (standard) color spaces and models. But real cameras do not have a such simple response curve, but non the less, the gamma function could be used if artificial images or certain reference images with defined response are used as input.

Empirical Model of Response (EMoR)

Another approach is to approximate the response function of a real camera with some standard mathematical model, like a low degree polynomial. An even better solution has been proposed by GROSSBERG and NAYAR in [EMoR]. Their idea was to directly find the space of existing response functions of real analog films and digital sensors, not some standard function space like polynomials.

Therefor they collected response curves from various analogs films and digital sensors and build a database from that they called "Database of Response Functions" (DoRF) available at [EMoR-Web]. To create a model from that large set of response functions, they used a



(a) The first base vector which is the average of all camera response functions used to calculate the principal component analysis.

(b) The further base functions of the vector space of response functions. Please note the smaller vertical scaling.

Figure 10: Base of the "Empirical Model of Response" (EMoR) vector space of response functions. Data from [EMoR-Web].

standard method of signal processing to reduce the number of parameters needed to describe an arbitrary response function: Principal Component Analysis (PCA).

PCA requires a set of vectors of a finite-dimensional vector space. But the space of monotone functions has infinite dimension, so the functions have to be sampled first to "reduce" their dimensionality to some finite value. GROSSBERG and NAYAR chose to take one thousand samples per function and thus a one-thousand-dimensional vector space. From that the PCA generates a series of base functions (principal components) of decreasing 'importance', the first one being the average of all response functions. The first few base functions are shown in Figure 10.

GROSSBERG and NAYAR found out, that three to five base functions (not counting the average function, for which no parameter has to be calculated or stored) may suffice to approximate most response functions very closely. So we will use the EMoR as our standard model for response functions.

3.3.2 Color Transforms

Now after linearizing the response of the target image and the reference image using the EMoR model, a color transform is calculated, which should map, with minimal error, the colors of the target image onto the colors of reference image. Since this is as well an optimization problem, a model with few parameters is needed.

Currently only two simple linear models are used: First, a simple scaling of each channel:

$$\begin{pmatrix} r_{ref} \\ g_{ref} \\ b_{ref} \end{pmatrix} = \begin{pmatrix} r_{scale} \cdot r_{target} \\ g_{scale} \cdot g_{target} \\ b_{scale} \cdot b_{target} \end{pmatrix} = \begin{pmatrix} r_{scale} & 0 & 0 \\ 0 & g_{scale} & 0 \\ 0 & 0 & b_{scale} \end{pmatrix} \cdot \begin{pmatrix} r_{target} \\ g_{target} \\ b_{target} \end{pmatrix}$$

which is a model with only three parameters. And second a linear transform which could also be called channel (re-)mixing:

$$\begin{pmatrix} r_{ref} \\ g_{ref} \\ b_{ref} \end{pmatrix} = \begin{pmatrix} rr & gr & br \\ rg & gg & bg \\ rb & gb & bb \end{pmatrix} \cdot \begin{pmatrix} r_{target} \\ g_{target} \\ b_{target} \end{pmatrix}$$

which is a superset of the scaling model and has nine parameters.

A better model would be a transform based on the target and the reference color temperature and tint (green-pink-balance) which has only four parameters. But since color temperature based model eventually results in a linear transformation matrix as well (although it is not linear in its parameters) and since they are far more complicated and might even require knowledge about the camera sensor sensitivity spectra, they are not considered here.

3.3.3 Fitting Models to Data

Now, having a model for camera responses and for color transformations, the parameters of these models have to be adjusted to match the set of data points gathered from the color chart within the images. This is done by using a generic optimization algorithm, which needs some 'error measure' to minimize.

After some experimentation with different error measures, the author decided to use a simple geometric distance. First, given the colors points $c_1 := (r_1, g_1, b_1)$ and $c_2 := (r_2, g_2, b_2)$, the distance between them is defined by:

$$\Delta(c, c') := \sqrt{(r - r')^2 + (g - g')^2 + (b - b')^2}$$

Given two sets $D_1 = (c_1, c_2, \dots)$, $D_2 = (c'_1, c'_2, \dots)$ of color points, their total distance is defined by:

$$\Delta(D_1, D_2) := \sqrt{\sum_i (\Delta(c_i, c'_i))^2} = \sqrt{\sum_i \left((r_i - r'_i)^2 + (g_i - g'_i)^2 + (b_i - b'_i)^2\right)}$$

The prototype implementation uses a variant of the BFGS algorithm, namely the `fmin_l_bfgs_b` function from the `scipy.optimize` module.

Linking Parameters

To improve robustness, certain parameters can be linked together by defining them to be equal. This is possible when it can be assured, that two or more pictures are shot using the same camera response function or color transform, for example because the same camera with identical settings where used. This reduces the number of parameters to be guess by the optimizer and thus improves robustness while potentially increasing the error due to the model becoming 'simpler'.

4 Practical Experiments and Results

4.1 Color Correction Experiments

4.1.1 Basic Setup

In this section the author wants to present the results of applying the `caccrc.py` software, which implements the algorithms described earlier, to a series of photographs made with two different cameras of him.

To verify if color correction works as expected, multiple shots were taken of a test scene illuminated using daylight. All used shots are shown in Figure 11 and Figure 12.

The cameras were a "Canon EOS 400D", a small single lens reflex (SLR) camera, and a "Canon PowerShot A650 IS", an older compact camera.

Between the shots, white balance, color correction (only EOS 400D) and exposure compensation settings were varied to simulate different cameras. Since the environment was not controlled, the illumination might have varied a little as well, but that should impact the results.

Additionally software was applied to a mixed set of shots from both cameras to see if the calculation of the response function works out. The mixed set contains all shots except the pure '+1 EV' and '−1 EV' shots for reasons of clarity.

Both Cameras were set to aperture priority mode (meaning fixed aperture, variable exposure time) and the settings where varied according to Table 1 on the next page.

Canon EOS 400D	
Shot	Settings (WB, CC, ExpComp)
IMG_5745	–
IMG_5746	“Tungsten”
IMG_5747	“Fluorescent”
IMG_5748	“Sunny”
IMG_5749	“Cloudy”
IMG_5750	“Shade”
IMG_5751	−1 EV
IMG_5752	+1 EV
IMG_5753	“Tungsten”, −1 EV
IMG_5754	“Sunny”, +1 EV
IMG_5755	CC: +3 Blue, +5 Green
IMG_5756	CC: +3 Amber, +5 Magenta

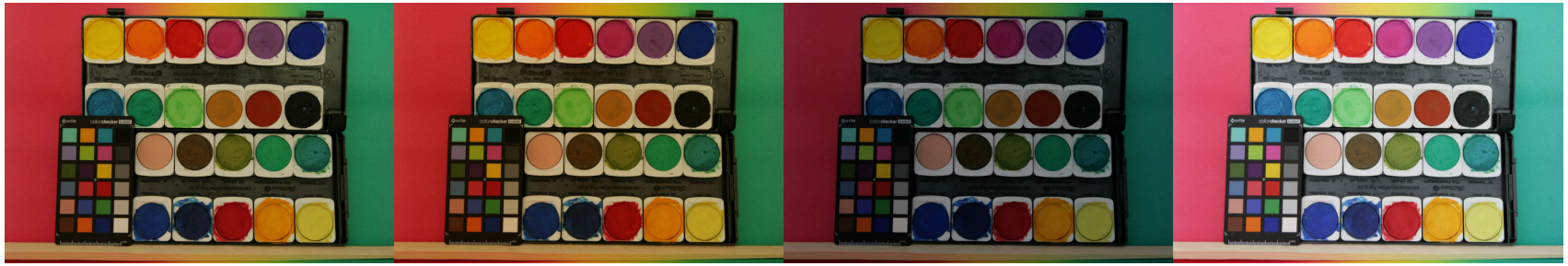
(a) Settings used for the shots of the EOS 400D series.

Canon PowerShot A650 IS	
Shot	Settings (WB, ExpComp)
IMG_0396	–
IMG_0397	“Tungsten”
IMG_0398	“Fluorescent”
IMG_0399	“Sunny”
IMG_0400	“Cloudy”
IMG_0401	−1 EV
IMG_0402	+1 EV
IMG_0403	“Tungsten”, −1 EV
IMG_0404	“Cloudy”, +1 EV

(b) Settings used for the PowerShot A650 series.

Table 1: Camera settings used while shooting the different images. Common settings, unless stated otherwise, where: aperture priority mode with aperture set to 5.0, automatic color balance and exposure time.

(a) IMG_5745 (Auto) (b) IMG_5746 (“Tungsten”) (c) IMG_5747 (“Fluorescent”) (d) IMG_5748 (“Sunny”)

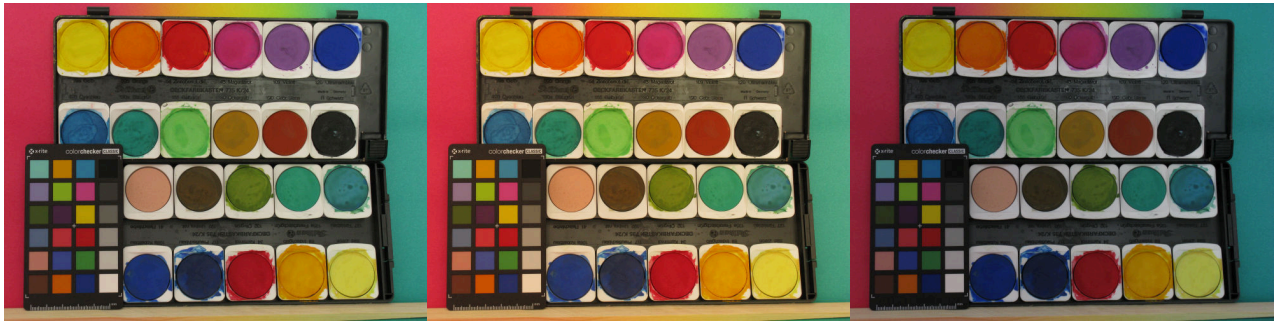


(e) IMG_5749 (“Cloudy”) (f) IMG_5750 (“Shade”) (g) IMG_5751 (–1 EV) (h) IMG_5752 (+1 EV)

(i) IMG_5753 (“Tungsten”, –1 EV) (j) IMG_5754 (“Sunny”, +1 EV) (k) IMG_5755 (CC: +3 Blue, +5 Green) (l) IMG_5756 (CC: +3 Amber, +5 Magenta)

Figure 11: Shown are all images from the EOS 400D set with the used settings in parentheses.

(a) IMG_0396 (Auto) (b) IMG_0397 (“Tungsten”) (c) IMG_0398 (“Fluorescent”)



(d) IMG_0399 (“Sunny”) (e) IMG_0400 (“Cloudy”) (f) IMG_0401 (−1 EV)

(g) IMG_0402 (+1 EV) (h) IMG_0403 (“Tungsten”, −1 EV) (i) IMG_0404 (“Cloudy”, +1 EV)

Figure 12: Shown are all images from the PowerShot A650 set with the used settings in parentheses.

4.1.2 Single Camera Results

In the first Scenario an array of cameras of identical type but with slightly different color reproduction behavior, due to normal variations of the material or due to errors in the setup (for example different aperture settings), is simulated.

Canon EOS 400D Set

The results of applying the color correction software to the set of twelve shots made with the Canon EOS 400D and the effect of using different color correction models is shown in Figure 13 on the following page.

The attempt to match the reference color worked quite successful for the shots without exposure compensation, which might work even be better, if the optimizer would not have to match the colors of the shots with exposure compensation. But at least the artificially wrong exposure is clearly visible in the color transformations matrices and can thus be detected and manually corrected.

The color transformation matrices used in the linear+EMoR setup is shown in Figure 14 on page 31 and the calculated camera response and the used EMoR parameters are shown in Figure 15 on page 31.

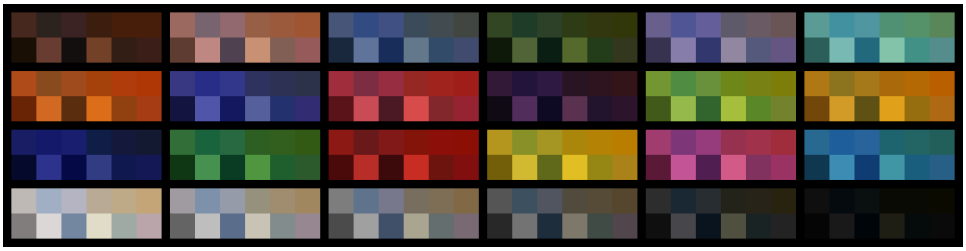
Canon PowerShot A650 Set

The results for the PowerShot A650 set are shown in Figure 16 on page 32.

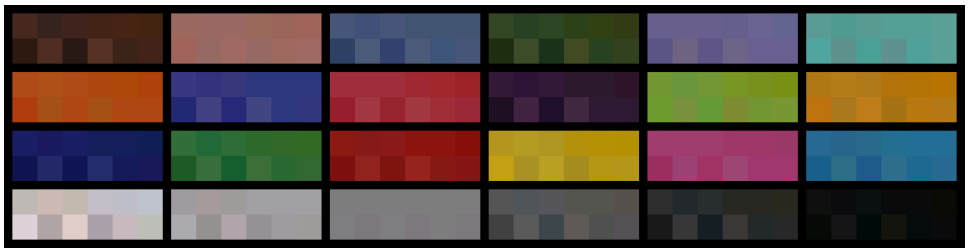
For the PowerShot set similar or even better results were achieved. For some reason the exposure compensation shots do align better than the respective shots of the EOS 400D set.

The color transformation matrices used in the linear+EMoR setup is shon in Figure 17 on page 33 and the calculated camera response and the used EMoR parameters are shown in Figure 18 on page 33.

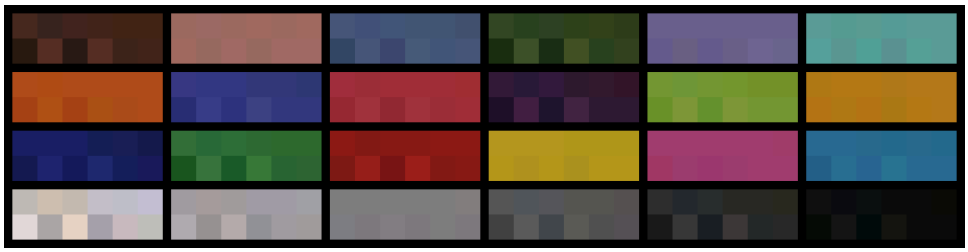
Color Comparison for the EOS 400D Set



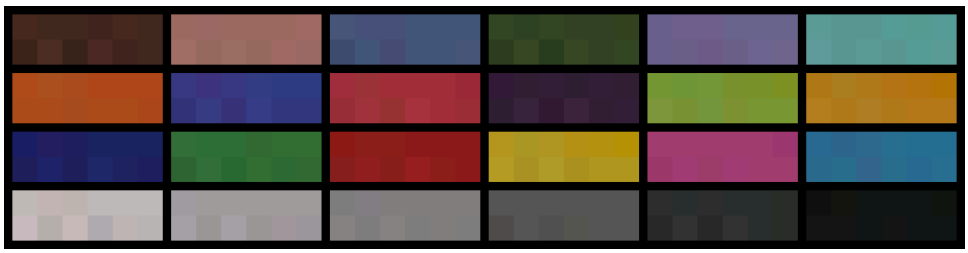
(a) Colors without correction.



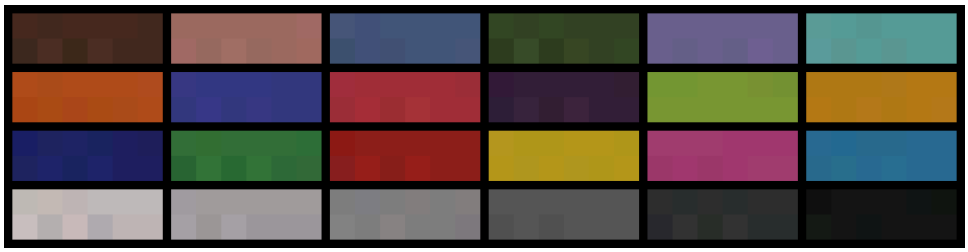
(b) . . . with scale transform.



(c) . . . with linear transform.



(d) . . . with scale transform and EMoR.



(e) . . . with linear transform and EMoR.

Figure 13: Shown are the color samples extracted from the patches of the color chart as depicted in the different Canon EOS 400D shots with different settings for color correction. The top left field of each patch of the chart is extracted from the reference image stays unchanged after color correction.

Color Transform Parameters for the EOS 400D Set

— IMG_5745 —
(WB: "Auto")

—

— IMG_5746 —
(WB: "Tungsten")

$$\begin{pmatrix} +2.58 & -0.08 & -0.04 \\ +0.11 & +1.83 & -0.07 \\ -0.14 & +0.05 & +1.48 \end{pmatrix}$$

— IMG_5747 —
(WB: "Fluorescent")

$$\begin{pmatrix} +1.89 & -0.02 & -0.03 \\ +0.03 & +1.69 & -0.03 \\ -0.08 & +0.00 & +1.45 \end{pmatrix}$$

— IMG_5748 —
(WB: "Sunny")

$$\begin{pmatrix} +1.83 & +0.02 & -0.02 \\ -0.07 & +1.96 & +0.05 \\ +0.07 & -0.12 & +2.45 \end{pmatrix}$$

— IMG_5749 —
(WB: "Cloudy")

$$\begin{pmatrix} +1.66 & +0.03 & -0.02 \\ -0.08 & +1.98 & +0.10 \\ +0.11 & -0.16 & +2.76 \end{pmatrix}$$

— IMG_5750 —
(WB: "Shade")

$$\begin{pmatrix} +1.63 & +0.04 & -0.01 \\ -0.13 & +2.12 & +0.17 \\ +0.20 & -0.20 & +3.26 \end{pmatrix}$$

— IMG_5751 —
(–1 EV)

$$\begin{pmatrix} +3.85 & -0.05 & -0.06 \\ -0.25 & +3.54 & -0.18 \\ +0.05 & +0.03 & +3.85 \end{pmatrix}$$

— IMG_5752 —
(+1 EV)

$$\begin{pmatrix} +1.09 & -0.02 & -0.03 \\ +0.02 & +1.13 & +0.01 \\ -0.04 & -0.03 & +1.08 \end{pmatrix}$$

— IMG_5753 —
(WB: "Tungsten", –1 EV)

$$\begin{pmatrix} +4.76 & -0.14 & -0.03 \\ +0.08 & +3.05 & -0.25 \\ -0.23 & +0.10 & +2.47 \end{pmatrix}$$

— IMG_5754 —
(WB: "Sunny", +1 EV)

$$\begin{pmatrix} +0.94 & -0.01 & -0.02 \\ +0.01 & +1.07 & +0.04 \\ -0.01 & -0.07 & +1.23 \end{pmatrix}$$

— IMG_5755 —
(CC: +3 Blue, +5 Green)

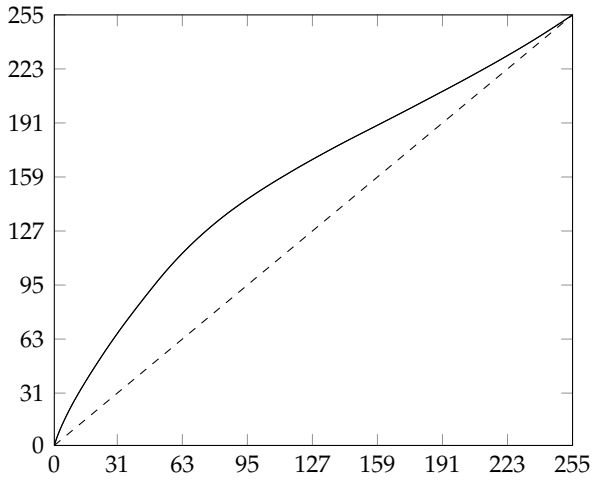
$$\begin{pmatrix} +2.31 & -0.02 & -0.03 \\ +0.02 & +1.97 & -0.00 \\ -0.04 & -0.05 & +2.04 \end{pmatrix}$$

— IMG_5756 —
(CC: +3 Amber, +5 Magenta)

$$\begin{pmatrix} +1.83 & +0.01 & -0.03 \\ -0.07 & +2.03 & -0.01 \\ -0.01 & -0.04 & +1.91 \end{pmatrix}$$

Figure 14: The parameters of the linear color transform used to correct the shots of the EOS 400D series to look like the reference shot.

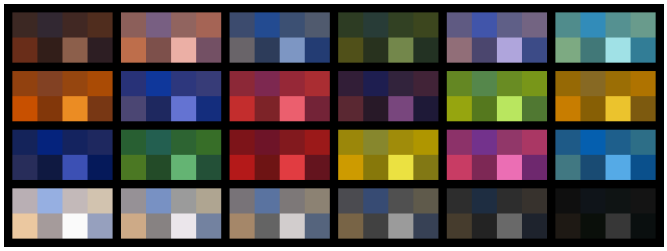
Camera Response for the EOS 400D Set



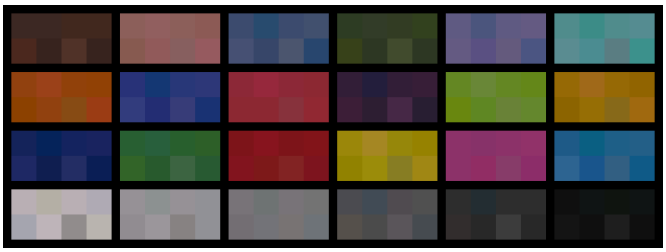
$p_1 = +2.157$
$p_2 = -0.485$
$p_3 = +0.453$
$p_4 = -0.177$
$p_5 = +0.084$

Figure 15: The calculated response curves for the EOS 400D series and the factors for the base functions h_1 to h_5.

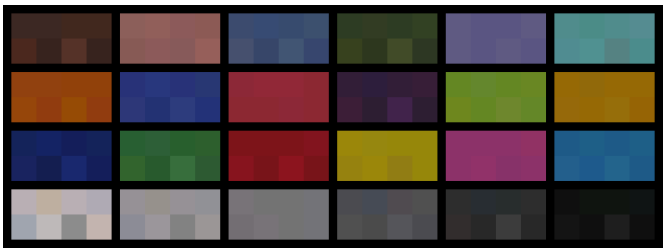
Color Comparison for the PowerShot A650 Set



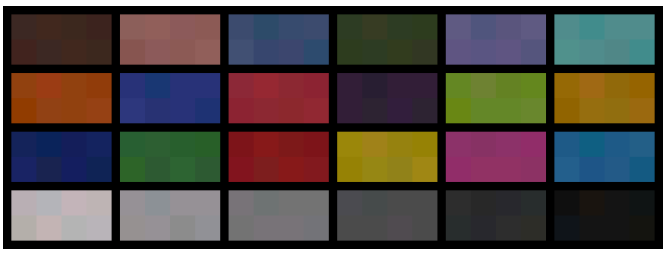
(a) Colors without correction.



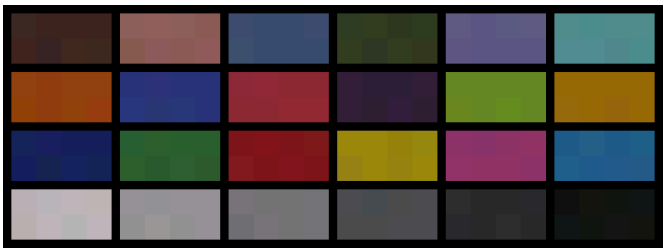
(b) . . . with scale transform.



(c) . . . with linear transform.



(d) . . . with scale transform and EMoR.



(e) . . . with linear transform and EMoR.

Figure 16: Shown are the color samples extracted from the patches of the color chart as depicted in the different Canon PowerShot A650 shots with different settings for color correction. The top left field of each patch of the chart is extracted from the reference image stays unchanged after color correction.

Color Transform Parameters for the PowerShot A650 Set

— IMG_0396 —
(WB: "Auto")
—

— IMG_0397 —
(WB: "Tungsten")
$$\begin{pmatrix} +2.18 & -0.01 & -0.02 \\ -0.07 & +1.80 & -0.16 \\ +0.18 & -0.01 & +1.17 \end{pmatrix}$$

— IMG_0398 —
(WB: "Flourescent")
$$\begin{pmatrix} +1.64 & +0.01 & +0.00 \\ +0.01 & +1.63 & -0.00 \\ +0.01 & +0.01 & +1.67 \end{pmatrix}$$

— IMG_0399 —
(WB: "Sunny")
$$\begin{pmatrix} +1.31 & +0.03 & +0.02 \\ +0.03 & +1.44 & +0.05 \\ -0.04 & -0.01 & +1.76 \end{pmatrix}$$

— IMG_0400 —
(WB: "Cloudy")
$$\begin{pmatrix} +0.96 & +0.04 & +0.02 \\ +0.02 & +1.25 & +0.08 \\ -0.07 & -0.02 & +1.87 \end{pmatrix}$$

— IMG_0401 —
(–1 EV)
$$\begin{pmatrix} +2.17 & -0.04 & +0.00 \\ +0.02 & +2.20 & -0.07 \\ +0.03 & +0.10 & +2.39 \end{pmatrix}$$

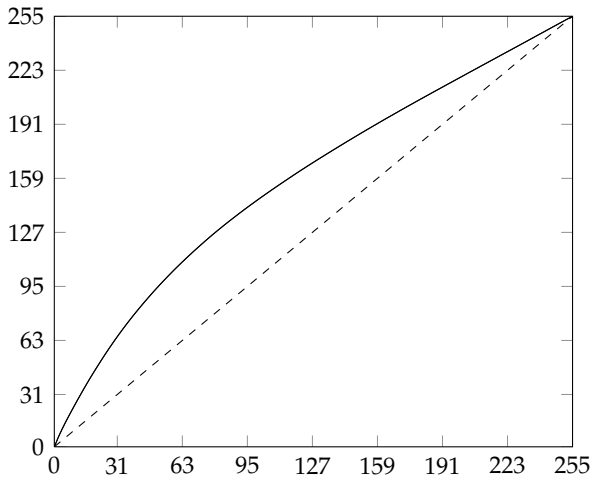
— IMG_0402 —
(+1 EV)
$$\begin{pmatrix} +0.59 & -0.00 & +0.01 \\ +0.03 & +0.60 & -0.04 \\ -0.01 & +0.02 & +0.65 \end{pmatrix}$$

— IMG_0403 —
(WB: "Tungsten", –1 EV)
$$\begin{pmatrix} +2.42 & -0.01 & -0.02 \\ -0.06 & +2.19 & -0.11 \\ +0.19 & +0.02 & +1.71 \end{pmatrix}$$

— IMG_0404 —
(WB: "Cloudy, +1 EV")
$$\begin{pmatrix} +0.72 & +0.03 & +0.01 \\ +0.01 & +0.84 & +0.01 \\ -0.04 & -0.02 & +1.14 \end{pmatrix}$$

Figure 17: The parameters of the linear color transform used to correct the shots of the PowerShot A650 series to look like the reference shot.

Camera Response for the PowerShot A650 Set



$p_1 = +2.309$
$p_2 = -0.340$
$p_3 = +0.257$
$p_4 = +0.021$
$p_5 = +0.030$

Figure 18: The calculated response curves for the PowerShot A650 series and the factors for the base functions h_1 to h_5.

4.1.3 Multiple Camera Results

In a second scenario a camera array with cameras of two different types was simulated. So the *mixed set*, used as the basis of the following results, consists of all but four shots of the EOS 400D set and the PowerShot A650 set. The missing shots are the ones which only feature a simple exposure compensation and are removed for reasons of clarity only.

The results of applying the color correction software onto this set, again with different models, whereby the two subsets are always assigned two different camera responses of course, are shown in Figure 20 on the following page.

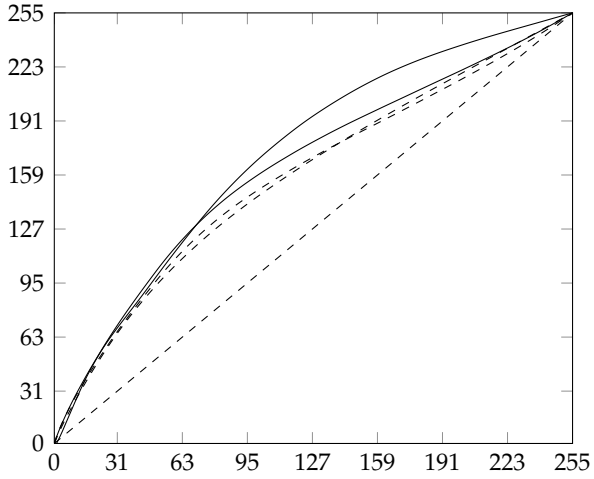
The results are quite good but not as good as in the single camera scenario. Especially the response curve differ quite heavily from the ones calculated in the single camera scenarios. It seems, that either there is not enough information available to correctly fit the model or that the model is not good enough or both.

Like before the color transform matrices of the linear+EMoR setup are shown in Figure 21 on page 36 and the calculated camera responses along with the used EMoR parameters are shown in Figure 19.

Shots of a 'Real' Scene

As a final example, a set of four image from the pool of images used while working on this topic has been fed into the `caccrc.py` program and the results are shown in Figure 22.

Camera Response for the Mixed Set

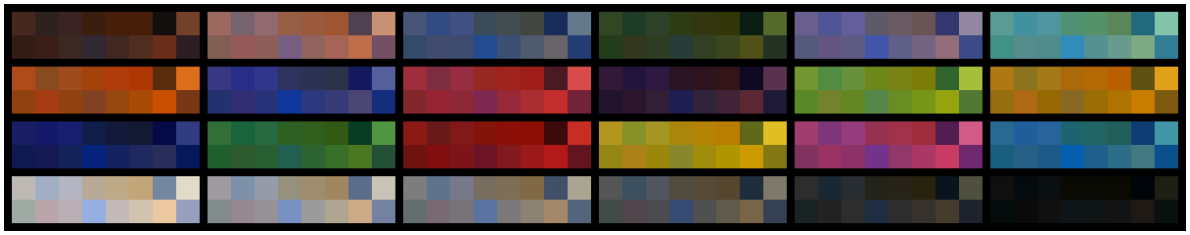


$p_1 = +0.500$
$p_2 = +0.643$
$p_3 = -0.197$
$p_4 = +0.091$
$p_5 = +0.166$

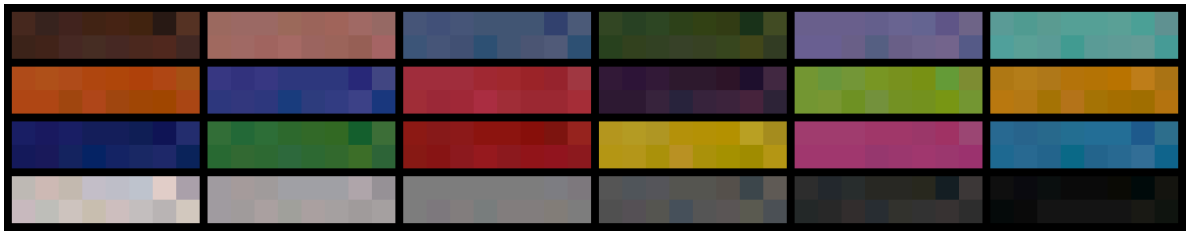
$p_1 = +1.378$
$p_2 = -0.267$
$p_3 = +0.389$
$p_4 = -0.157$
$p_5 = +0.093$

Figure 19: The calculated response curves for the mixed set and the factors for the base functions h_1 to h_5. The dashed lines show the responses calculates in the single camera scenarios.

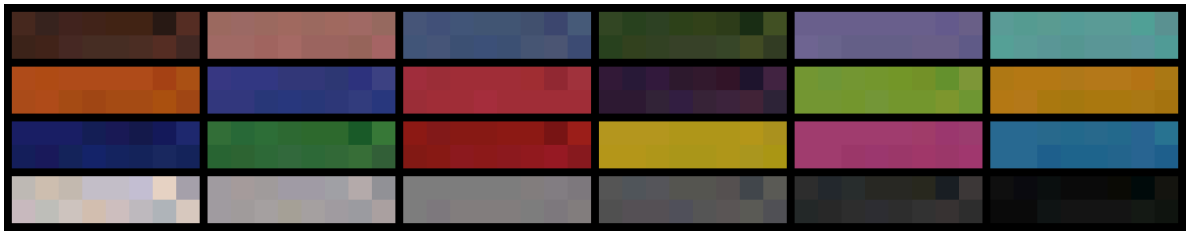
Color Comparison for the Mixed Set



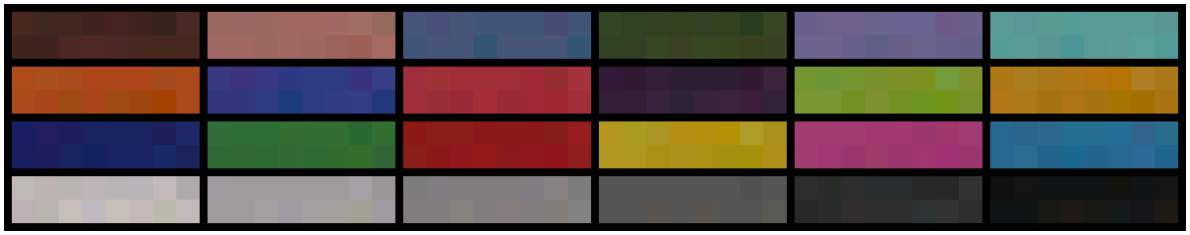
(a) Color without correction.



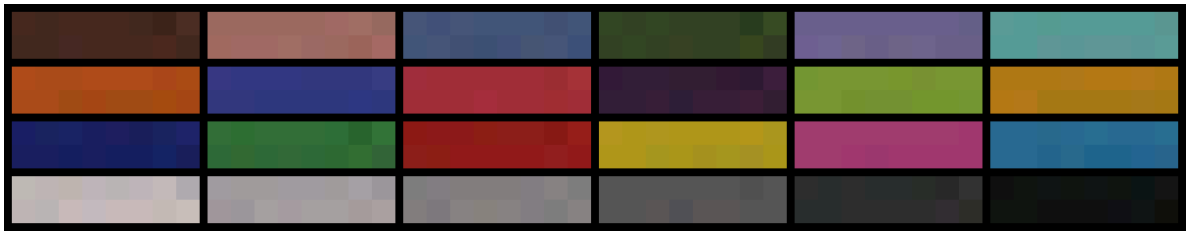
(b) . . . with scale transform



(c) . . . with linear transform.



(d) . . . with scale transform and EMoR.



(e) . . . with linear transform and EMoR.

Figure 20: Shown are the color samples extracted from the patches of the color chart as depicted in the different shots of the mixed set with different settings for color correction. The top left field of each patch of the chart is extracted from the reference image stays unchanged after color correction.

Color Transform Parameters for the Mixed Set

— IMG_5745 —
(WB: “Auto”)
—

— IMG_5746 —
(WB: “Tungsten”)
$$\begin{pmatrix} +2.05 & -0.06 & -0.03 \\ +0.09 & +1.46 & -0.06 \\ -0.11 & +0.05 & +1.20 \end{pmatrix}$$

— IMG_5747 —
(WB: “Fluorescent”)
$$\begin{pmatrix} +1.53 & -0.01 & -0.02 \\ +0.02 & +1.36 & -0.03 \\ -0.06 & +0.01 & +1.18 \end{pmatrix}$$

— IMG_5748 —
(WB: “Sunny”)
$$\begin{pmatrix} +1.48 & +0.02 & -0.01 \\ -0.06 & +1.56 & +0.05 \\ +0.06 & -0.08 & +1.94 \end{pmatrix}$$

— IMG_5749 —
(WB: “Cloudy”)
$$\begin{pmatrix} +1.35 & +0.03 & -0.01 \\ -0.07 & +1.57 & +0.09 \\ +0.09 & -0.11 & +2.17 \end{pmatrix}$$

— IMG_5750 —
(WB: “Shade”)
$$\begin{pmatrix} +1.33 & +0.04 & +0.00 \\ -0.11 & +1.68 & +0.14 \\ +0.16 & -0.13 & +2.57 \end{pmatrix}$$

— IMG_5753 —
(WB: “Tungsten”, –1 EV)
$$\begin{pmatrix} +3.70 & -0.08 & -0.00 \\ +0.09 & +2.38 & -0.20 \\ -0.15 & +0.11 & +1.96 \end{pmatrix}$$

— IMG_5754 —
(WB: “Sunny”, +1 EV)
$$\begin{pmatrix} +0.78 & -0.01 & -0.02 \\ +0.00 & +0.88 & +0.03 \\ -0.01 & -0.06 & +1.00 \end{pmatrix}$$

— IMG_5755 —
(CC: +3 Blue, +5 Green)
$$\begin{pmatrix} +1.84 & -0.01 & -0.02 \\ +0.02 & +1.57 & +0.00 \\ -0.02 & -0.03 & +1.63 \end{pmatrix}$$

— IMG_5756 —
(CC: +3 Amber, +5 Magenta)
$$\begin{pmatrix} +1.48 & +0.02 & -0.02 \\ -0.05 & +1.61 & -0.01 \\ -0.01 & -0.02 & +1.53 \end{pmatrix}$$

— IMG_0396 —
(WB: “Auto”)
$$\begin{pmatrix} +1.51 & +0.04 & +0.01 \\ -0.02 & +1.41 & +0.11 \\ +0.02 & -0.01 & +1.26 \end{pmatrix}$$

— IMG_0397 —
(WB: “Tungsten”)
$$\begin{pmatrix} +1.85 & +0.04 & -0.00 \\ -0.08 & +1.45 & -0.02 \\ +0.17 & -0.02 & +0.86 \end{pmatrix}$$

— IMG_0398 —
(WB: “Flourescent”)
$$\begin{pmatrix} +1.42 & +0.04 & +0.02 \\ -0.02 & +1.33 & +0.11 \\ +0.02 & -0.02 & +1.19 \end{pmatrix}$$

— IMG_0399 —
(WB: “Sunny”)
$$\begin{pmatrix} +1.14 & +0.05 & +0.03 \\ -0.00 & +1.17 & +0.13 \\ -0.03 & -0.04 & +1.25 \end{pmatrix}$$

— IMG_0400 —
(WB: “Cloudy”)
$$\begin{pmatrix} +0.84 & +0.05 & +0.02 \\ -0.01 & +1.02 & +0.15 \\ -0.06 & -0.06 & +1.32 \end{pmatrix}$$

— IMG_0403 —
(WB: “Tungsten”, –1 EV)
$$\begin{pmatrix} +2.04 & +0.06 & +0.01 \\ -0.07 & +1.73 & +0.05 \\ +0.20 & +0.01 & +1.22 \end{pmatrix}$$

— IMG_0404 —
(WB: “Cloudy, +1 EV”)
$$\begin{pmatrix} +0.62 & +0.03 & +0.01 \\ +0.00 & +0.71 & +0.06 \\ -0.04 & -0.04 & +0.82 \end{pmatrix}$$

Figure 21: The parameters of the linear color transform used to correct the shots of the mixed set to look like the reference shot.

(a) The reference image (“eos5_4”).

(b) The original shots (“eos5_1”, “eos5_3”, “iphone4”)...

(c) ...and the color corrected version.

Figure 22: Shown are several images (the originals provided by Prof. Eisert) and their color corrected version of them using the image in Figure 22a as reference. For each shot a separate response curve was calculated.

4.2 Optimal Processing Parameter Guessing

The Algorithms described in Section 2 "Chart Discovery Algorithm" depend on various parameters, which have to be determined somehow. The obvious method would be to experiment with the values until the results are satisfactory, but this requires a way to measure the quality of the result.

The result of the chart discovery process is primarily a coordinate system for the chart as it is depicted in that shot. Thus first the quality of that coordinate system should be measured to begin with. For that a reference coordinate system is needed, which the author created by hand-drawing 'reference pictures' for some of the shots[10]. In them the region (connected set of pixel) for each patch are marked. All images with their reference image are shown in Figure 23 on the next page.

These reference images are similar to the images of "flat" pixels used in the chart discovery process, but here the patch belonging to each regions is already encoded. So its possible to jump right to the coordinate system creation by calculating the centers of mass and use them as the patch locations in the picture.

Then the positions of the patches according to the reference image can be compared to the positions of the patches according to the coordinate system obtained from the chart discovery process applied to the same shot. The error is calculated in image space with normalized image size and the mean Euclidean distance is used as the error measure.

Unfortunately, since the process currently is not robust, the chart discovery may fail completely, for example because not enough regions (complete row or columns missing) or erroneous regions, not associated with a patch but some other object, were found, so the failures of that process are counted as well and used as a second measure.

Now one parameter is varied at a time leaving the others at their default setting to see, if this changes the quality of the result. The optimal setting should primarily minimize the failure rate. After that has been achieved, the improvement of the accuracy of the grid is the second objective.

The results are shown in Figures 24, 25 and 26 on the following pages.

[10] Some of the shots were provided by the Prof. Eisert

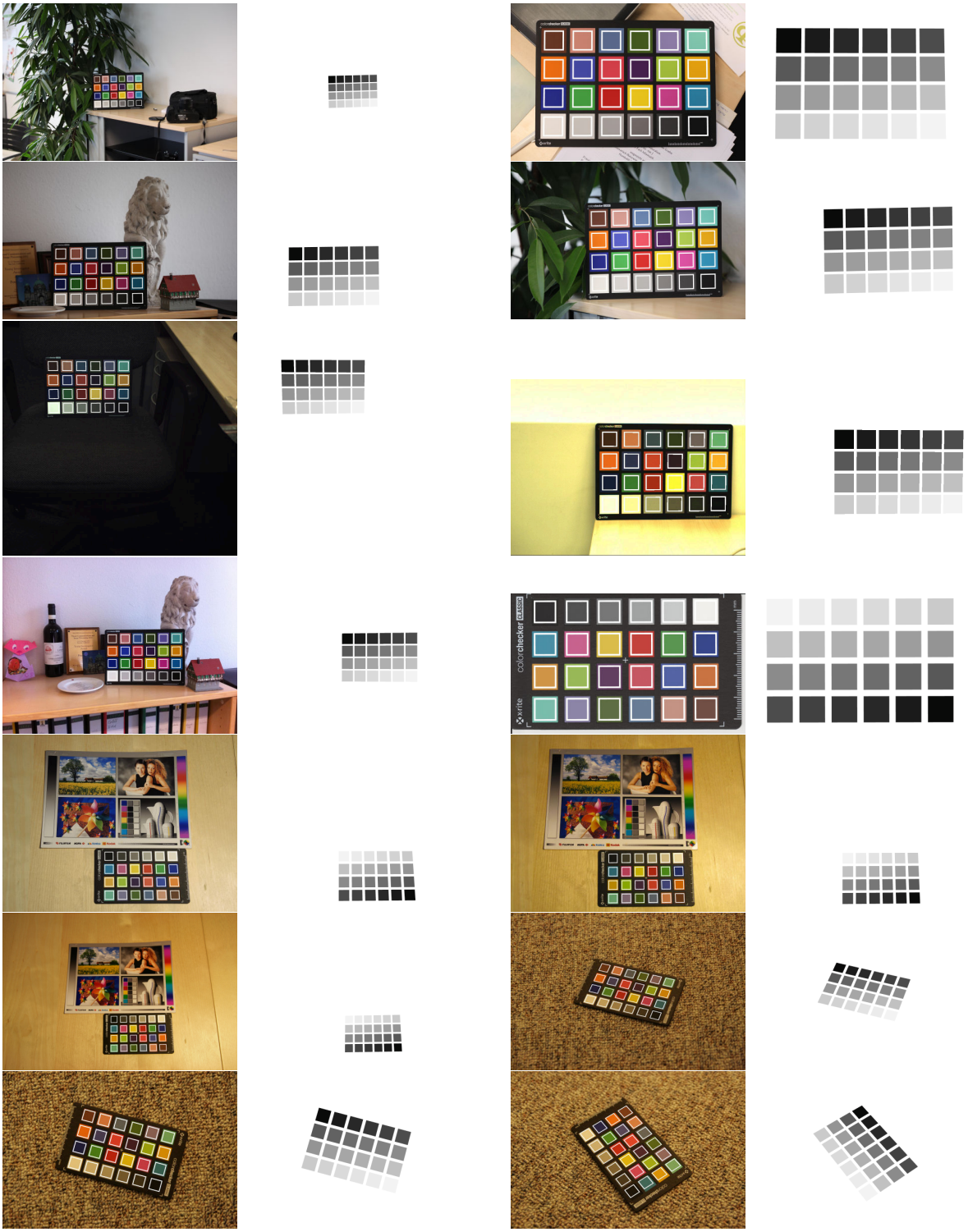
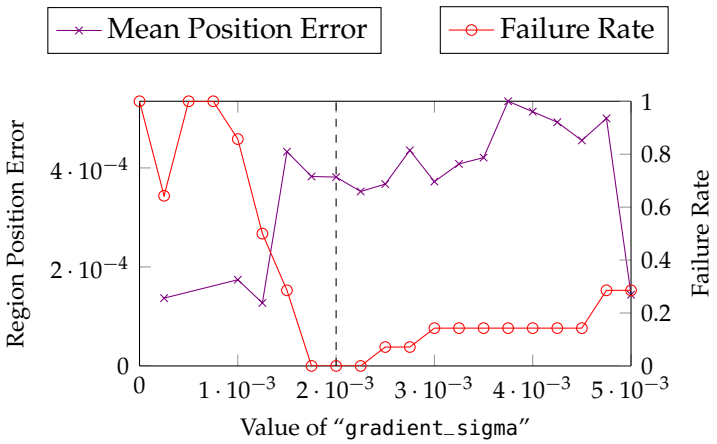
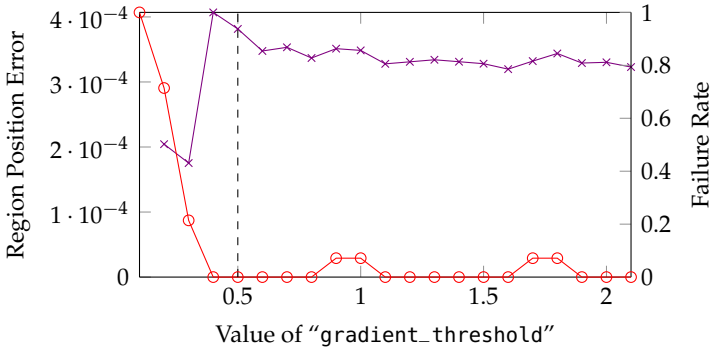


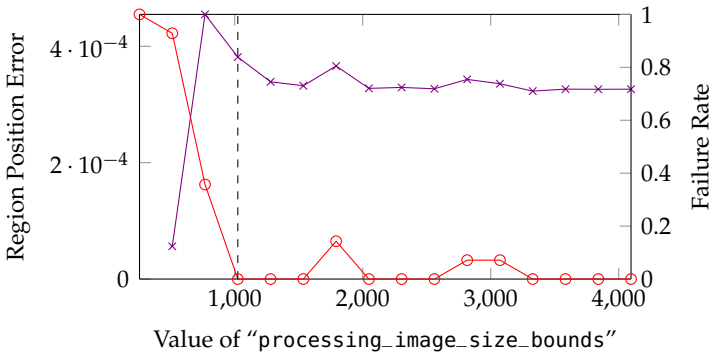
Figure 23: Shown are all images used in the evaluation with their respective reference image next to them. The inner of the white quadrilaterals shows the computed area used for color extraction.



(a) The "gradient_sigma" parameter controls amount of high frequency filtering and is used as a factor in calculating the standard deviation of the Gaussian kernel used for filtering. More filtering reduces the influence of noise but weakens edges and degrades shapes.

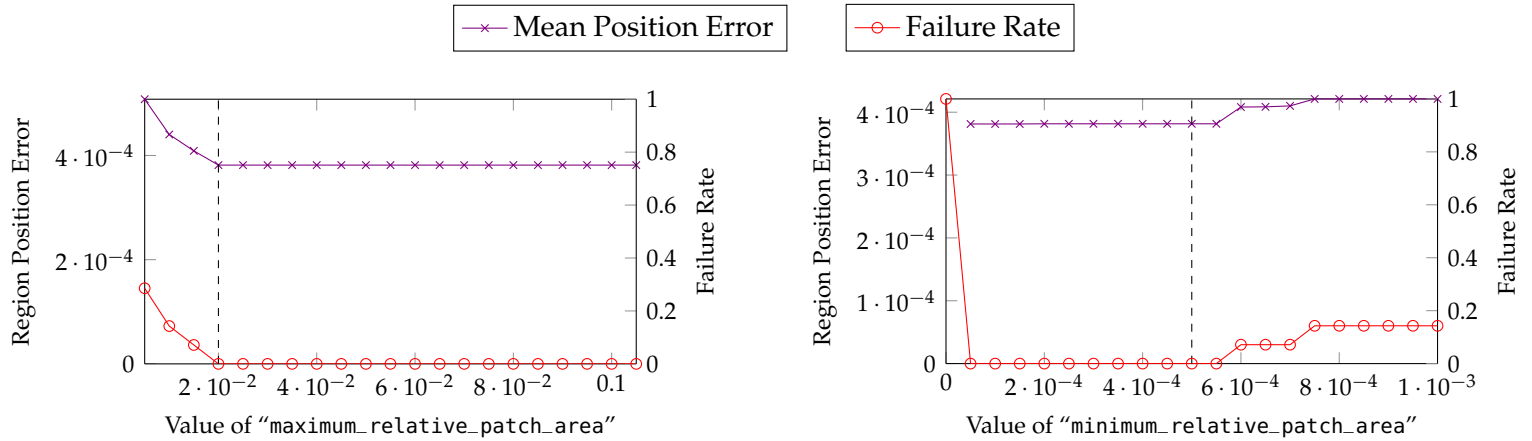


(b) The "gradient_threshold" parameter controls which pixels are considered flat and thus part of a region. Larger values cause more and larger regions with regions fusing eventually.

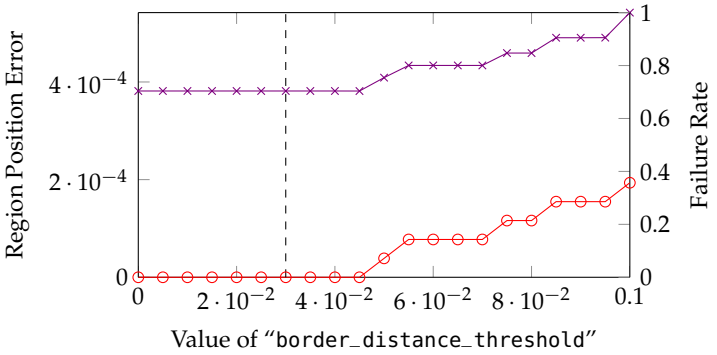


(c) For performance reasons not the full resolution image is processed, but a down-scaled version. The "processing_image_size_bounds" parameter sets this smaller size. Smaller values improve performance but reduce the precision of the calculated coordinate systems and cause more errors.

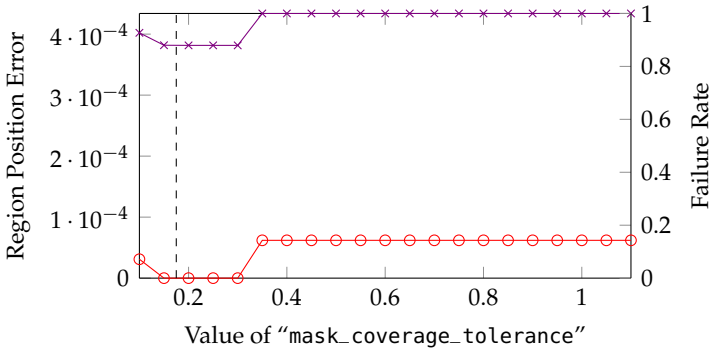
Figure 24: Shown are the failure rate and the mean position error relative to variations of certain parameters. The default value is indicated by a dashed line.



(a) The "maximum_relative_patch_area" and "minimum_relative_patch_area" parameters control, how large a region must be or may be to be considered. To large and to small regions are ignored.

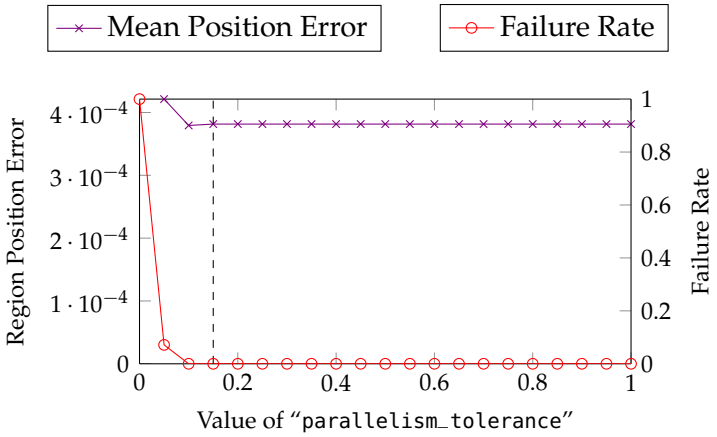


(b) The "border_distance_threshold" parameter controls how far a region must be from the border of the image. Regions near the border are ignored.

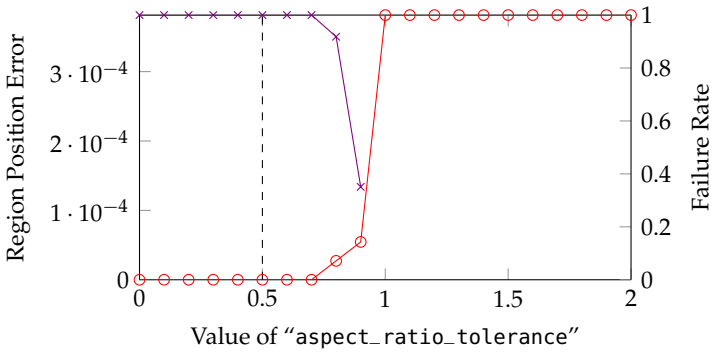


(c) The "mask_coverage_tolerance" parameters controls, to which fraction the region may deviate from the rasterized region mask. If their shapes, counted pixel by pixel, differ to much, the region is discarded.

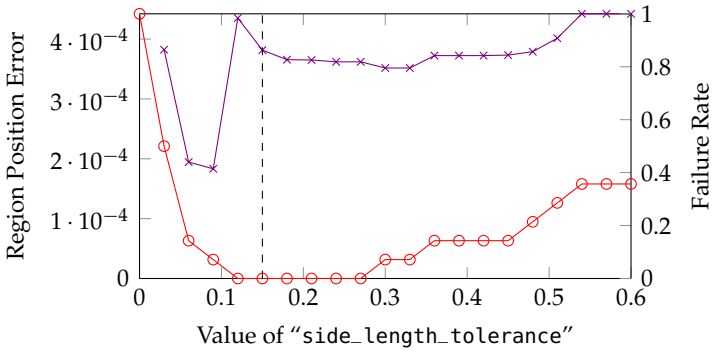
Figure 25: Shown are the failure rate and the mean position error relative to variations of certain parameters. The default value is indicated by a dashed line.



(a) The "parallelism_tolerance" parameter controls how 'parallel' the opposite edges of the region masks must be for the region to be considered.



(b) The "aspect_ratio_tolerance" parameter controls who far the aspect ratio of the region mask may deviate from 1 (square).



(c) The "side_length_tolerance" parameter how far the side lengths of all preselected region masks may deviate for the region to be selected.

Figure 26: Shown are the failure rate and the mean position error relative to variations of certain parameters. The default value is indicated by a dashed line.

5 Conclusion

In the thesis the author presented an experimental method of finding a color chart within an image using a bottom-up constructive method based on finding regions with homogeneous texture and combining them into a chart.

Further, a model for camera color processing and, based on that, a model for color correction has been proposed, using the EMoR model for camera responses.

Although the algorithms, in their current implementation, have some robustness issues, their is plenty room for improvement and some possible way to do that were already indicated in the text.

Finally, a reference implementation of the presented algorithms was applied to a series of images to test and show the capabilities of the proposed approach, as well as test the influence of different processing parameters on the quality of the result.

References

[CheckMyChart] A. ERNST, A. PAPST, T. RUF, J.-U. GARBAS:
"Check My Chart: A Robust Color Chart Tracker for Colorimetric Camera Calibration".
MIRAGE '13, June 06–07 2013, Berlin, Germany.

[ColorScience] G. WYSZECKI, *W. S. Stiles*:
"Color Science", Second Edition.
Wiley, 2000.

[Hugin] *"Hugin - Panorama photo sticher"*.
`http://hugin.sourceforge.net`.

[HuginPaper] PABLO D'ANGELO:
"Radiometric alignment and vignetting calibration".

[ColorTransfer] E. REINHARD, M. ASHIKHMIN, B. GOOCH, P. SHIRLEY:
"Color Transfer between Images".
IEEE Computer Graphics and Applications, Sep./Oct. 2001.

[ColorTransfer2] X. XIAO, L. MA:
"Color Transfer in Correlated Color Space".
VRCIA 2006, Hong Kong, 14–17 June 2006.

[EMoR] MICHAEL D. GROSSBERG, SHREE K. NAYAR:
"What is the Space of Camera Response Functions?".
Proceedings of the 2003 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'03).

[EMoR-Web] *"Camera Response Functions: Database (DoRF) and Model (EMoR)"*.
`http://www.cs.columbia.edu/CAVE/software/softlib/dorf.php`