A while back (July 2010 going by Git commit history), I hacked together a program for automatically finding the GretagMacbeth ColorChecker in an image, and cheekily named it Macduff.

The algorithm I developed used adaptive thresholding against the RGB channel images, followed by contour finding with heuristics to try to filter down to ColorChecker squares, then using k-means clustering to cluster squares (in order to handle the case of images with an X-Rite ColorChecker Passport), then computing the average square colors and trying to find if any layout/orientation of square clusters would match ColorChecker reference values (within some Euclidean distance in RGB space). Because of the original use case I was developing this for (automatically calibrating images against an image of a ColorChecker on a copy stand), I could assume that the ColorChecker would take up a relatively large portion of the input image, and coded Macduff using this assumption.

I recently decided to briefly revisit this problem and see if any additional work had been done, and I thought a quick survey of what I turned up might be generally useful:

- Tajbakhsh, Touraj, and Rolf-Rainer Grigat. *Semiautomatic color checker detection in distorted images*. Proceedings of the Fifth IASTED International Conference on Signal Processing, Pattern Recognition and Applications. ACTA Press, 2008. Unfortunately I cannot find any online full-text of this article, and my library doesn’t have the volume. Based on the description in Ernst 2013, the algorithm proceeds as follows: “The user initially selects the four chart corners in the image and the system estimates the position of all color regions using projective geometry. They transform the image with a Sobel kernel, a morphological operator and thresholding into a binary image and find connected regions.”
- Kapusi, Daniel, et al. *Simultaneous geometric and colorimetric camera calibration*. 2010. This method requires color reference circles placed in the middle of black and white chessboard squares, which they then locate using OpenCV’s chessboard detection.
- K. Hirakawa, “ColorChecker Finder,” accessed from http://campus.udayton.edu/~ISSL/software. AKA CCFind.m. The earliest Internet Archive Wayback Machine snapshot for this page is in August 2013, however I also found this s-colorlab mailing list announcement from May 2012. Unfortunately this code is under a
restrictive license: “This code is copyrighted by PI Keigo Hirakawa. The softwares are for research use only. Use of software for commercial purposes without a prior agreement with the authors is strictly prohibited.”

According to the webpage, “CCFind.m does not detect squares explicitly. Instead, it learns the recurring shapes inside an image.”

- Liu, Mohan, et al. *A new quality assessment and improvement system for print media*. EURASIP Journal on Advances in Signal Processing 2012.1 (2012): 1-17. An automatic ColorChecker detection is described as part of a comprehensive system for automatic color correction. The algorithm first quantizes all colors to those in the color chart, then performs connected component analysis with heuristics to locate patch candidates, which are then fed to a Delaunay triangulation which is pruned to find the final candidate patches, which is then checked for the correct color orientation. This is the same system described in: Konya, Iuliu Vasile, and Baia Mare. *Adaptive Methods for Robust Document Image Understanding*. Diss. Universitäts-und Landesbibliothek Bonn, 2013.


**Data Sets**


- Gehler’s Dataset (approx. 8GB, 592MB downsampled)
  - Shi’s Re-processing of Gehler’s Raw Dataset (4.2GB total) Used by Hirakawa. 568 PNG images.
  - Reprocessed Gehler “We noticed that the renderings provided by Shi and Funt made the colours look washed out. There also seemed to be a strong Cyan tint to all of the images. Therefore, we processed the RAW files ourselves using DCRAW. We followed the same methodology as Shi and Funt. The only difference is we did allow DCRAW to apply a D65 Colour Correction matrix to all of the images. This evens out the sensor responses.”
Suggested citation:


This work is licensed under a Creative Commons Attribution 4.0 International License.