Color calibration using differential evolution

Alan Samuel Aguirre Salazar
Faculty of Engineering.
Universidad Panamericana
Jalisco, México
Emails: ASASauqui@protonmail.com / 0227221@up.edu.mx

Josué Olmos Hernández
Faculty of Engineering.
Universidad Panamericana
Jalisco, México
Email: 0226153@up.edu.mx

***Abstract*—Color calibration is for practical use in some problems, such as algorithms for meat spoilage detection based on color, chemical measurements based on reagents, etc.; color is a physical aspect that provides information, therefore, its correct calibration is relevant. In this paper, the use of the differential evolution algorithm and various image processing techniques, such as inverse binary thresholding and perspective-dependent image warping, for calibrating colors in an image based on a ColorChecker. Transformation and perspective deformation were used for the robustness of the algorithm for the detection of the ColorChecker matrix, as well as the search for polygons according to the contours obtained based on an external search and their polygonal approximation through the algorithm of Douglas-Peucker.**

Keywords—Color calibration, differential evolution, perspective transform, image processing.

I. Introduction

The objective of this article is to develop an algorithm that allows the correct calibration of color images. The colors within the image contain information about the image that can be used for various analyses, either for the collection of physical or chemical characteristics of the image. In other related works, different methodologies are used to achieve this calibration, such as the use of Bayesian neural network models, where a neural network is trained with white balance data [1]; the application of Thin-Plate Spline interpolation algorithms, where the sRGB coordinates of the ColorChecker were deformed through TPS interpolation, modified for three-dimensional space [2]; the use of polynomial fit based on pieces, where functions are selected for different intervals [3]; the use of statistical-based methods in an unsupervised learning model where color constancy will be evaluated [4]; or there are even some more focused on merely detecting the ColorChecker in an optimal way, either with the use of deep convolutional neural networks, where a neural network is trained with synthetic images of the ColorChecker 3D models and different background images [5] .

The proposal within this article suggests the use of image processing to search for important elements that share characteristics with the matrix being searched for; deformation of the ColorChecker matrix image based on perspective to have a better manipulation of the information; and the use of the differential evolution algorithm for the optimization of an objective function that seeks to reduce the sum of the difference between the control ColorChecker matrix and the calibrated ColorChecker matrix, this by using a 3x3 matrix that will be responsible to be able to calibrate the image.

II. Color matrix extraction

A. Load images

Firstly, you need to load the two images to be used, the image that contains the control matrix and another image which will contain the same matrix, but in a different context (different lighting and position). Once these images are loaded, you can proceed to extract the color matrix.

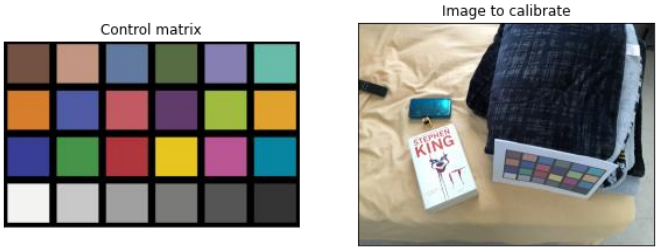

Fig. 1. Control matrix and the image to calibrate

B. Cut matrix

This is a method that receives the image where the color matrix is located, which you want to cut in order to extract the information. This method must be applied for both images.

Firstly, the area of the image (height x width) is obtained, this for later steps. In order to do a good image processing, the image must be transformed to a gray scale, and after that apply an inverse binary thresholding, it has to be inverse so that the matrix is taken with values of 1, otherwise it will take parts that are not wanted.

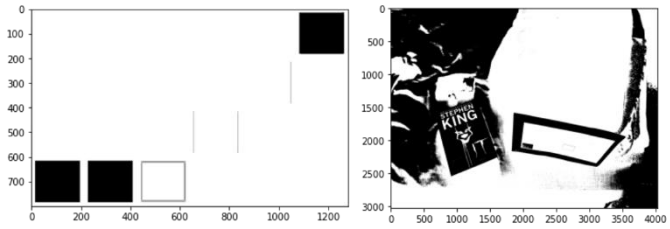

Fig. 2. Inversely thresholded images.

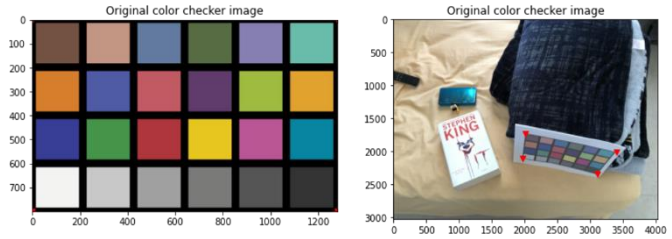

Fig. 3. ColorChecker detected.

Once this thresholding has been applied, an algorithm must be applied to search for the contours present in the image, but, since it is only desired to find the matrix in its entirety, this search for contours has to be for those external contours that do not belong to another contour, that is, look for contours that do not depend on others. With this, the matrix can be isolated for later location.

Once all the possible contours have been obtained, the following must be applied for each of them:

1. Get the area of the contour.
2. Calculate the perimeter of the contour by calculating the length of the curve:

   $$\varepsilon = 0.018 * \text{Arc Length of the contour}$$

   Where ε is the precision to compute a figure approximation.
3. Approximate a curve polygon or polygon with another curve/polygon with fewer vertices, so that the distance between them is less than or equal to the specified precision. This is done by using the Douglas-Peucker algorithm [6]. With this you can find the number of vertices of the polygon.
4. Once the number of vertices of the contour is obtained, those contours that are not suitable to be a matrix can be cleaned, those that are must have 4 vertices to comply with the shape of a rectangle [7]. In addition, it must be verified that the area of said contour is greater than 1% of the total area of the image, this to eliminate tiny garbage contours that are usually generated in the search step for external contours. Now, the normal thing is that the contour of the largest area of the image that meets these conditions must be the color matrix, since the other contours do not have that shape or considerable size.

Having found the best candidate (which should be the color matrix), we can proceed to the image transformation part of the matrix.

III. Transformation of the Matrix Image

A deformation process will be applied to the matrix cut or extracted from the image so that, regardless of the angle at which said matrix is, it remains in a flat context for better manipulation for data extraction.

Firstly, the coordinates of the vertices of said matrix must be arranged to your liking, in this case the order must be clockwise, starting from the upper left vertex, that is: Top-Left, Top-Right, Bottom-Right, Bottom-Left [8].

Once these vertices are arranged in the desired order, the dimensions where the image of the deformed matrix will reside must be created. In this case, the distances between two points must be obtained, obtaining the width A and B, and the height A and B, this is because a perfect rectangle cannot always result, normally it will have the shape of a trapezoid.

$$width_A = \sqrt{(Bottom\ Right_x - Bottom\ Left_x)^2 + (Bottom\ Right_y - Bottom\ Left_y)^2}$$

$$width_B = \sqrt{(Top\ Right_x - Top\ Left_x)^2 + (Top\ Right_y - Top\ Left_y)^2}$$

$$height_A = \sqrt{(Top\ Right_x - Bottom\ Right_x)^2 + (Top\ Right_y - Bottom\ Right_y)^2}$$

$$height_B = \sqrt{(Top\ Left_x - Bottom\ Left_x)^2 + (Top\ Left_y - Bottom\ Left_y)^2}$$

As the final image is intended to be a rectangle that maintains the proportions of the matrix, the maximum value must be taken for each width and each height.

$$maxWidth = max(width_A, width_B)$$

$$maxHeight = max(height_A, height_B)$$

Once the maximum height and width values are obtained, an array can be made with the dimensions of the new image, following the order of clockwise and starting from the top left. Here you must put the values in which each vertex will be.

$$NewImage = < [0,0],$$

$$[maxWidth - 1,0],$$

$$[maxWidth - 1, maxHeight - 1],$$

$$[0, maxHeight - 1] >$$

Once the maximum height and width values are obtained, a perspective transformation can be performed with the original points and the destination points. Having this, a perspective transformation algorithm is performed in order to provide you with more information about the required information of the given image.

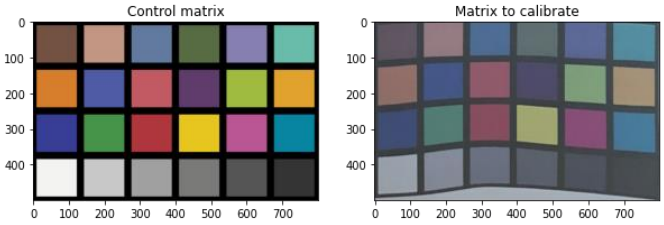

Fig. 4. Control matrix and matrix to be calibrated with the perspective transformation.

To finish, simply resize the image to 800x500 pixels, this to have a constant image size for data extraction, otherwise it may vary in size and affect the last process.

IV. Taking of Colors

A matrix 'm' of the size of the color matrix is created, which has 4 rows and 6 columns, therefore, it will be 4x6 in size.

$$m = \begin{pmatrix} R_1 & G_1 & B_1 \\ R_2 & G_2 & B_2 \\ | & | & | \\ R_6 & G_6 & B_6 \end{pmatrix}$$

You must determine in which position (x, y) you want to start to extract the colors in the image of the flat matrix, in this case it was taken: x = 50, y = 50. And each extraction square will have a height and width of 10 pixels.

All you have to do is increase the 'x' and 'y' by 135 pixels and 124 pixels respectively. The color matrix is made up of 4 rows and 6 columns, from there they must be increased.

The region of the square from xi to xi + width, and from yi to yi + height is extracted.

$$Color\ region = img[y_i : y_i + height, x_i : x_i + width]$$

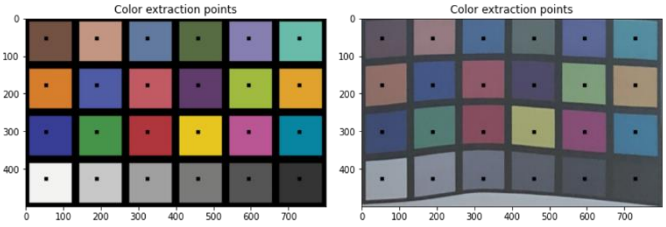

Fig. 5. Extracted color regions.

And, subsequently, the median of the values of said region of the image is taken for the RGB regions.

$$Median\ color = \\ < median(color\ region_R), median(color\ region_G), median(color\ region_B) >$$

Said average value of the region must be the color that represents the color boxed in the matrix image, an average could also be made, but in this case the median was better chosen. With this average color obtained, it must be added to the matrix 'm' in position 'i'.

$$m_i = Median\ color$$

In this way you can extract the information from the matrix of an image. Once the two matrices have been made (one for each given image), both the control matrix 'C' and the matrix to be calibrated 'Mo' can be used for the image calibration process.

V. Differential Evolution

The differential evolution algorithm is a stochastic search algorithm, which works through a population of solutions called vectors [9], of the type:

$$x = < x_1, x_2, \ldots, x_n >$$

This algorithm is useful to explore different search areas and gradually move the population to better regions in the search space [10], it is based on 3 important steps: mutation, crossover and selection of survivors.

In the mutation, for each individua x_i, another called v_i is generated.

$$v_i = x^{r1} + F(x^{r2} - x^{r3})$$

Where F is a random number between 0 and 2.

Later, in the crossover, what is sought is to combine the original vector x_i with the previously created v_i for the emergence of another vector called u_i. This combination is made interleaved from the index 0 to k, and the decision to take for u_{ik} the x_{ij} or the v_{ij}, is made by means of a probability 'Cr', which in this case is 0.5 [11].

$$u_k^i = \begin{cases} v_k^i, If\ rand(0,1) \leq Cr \\ x_k^i, otherwise \end{cases}$$

And finally, in the selection of survivors, the best individual between x_i and u_i is selected for a tournament, and the one who survives will be part of the next generation.

$$x_i = best(x_i, u_i)$$

A. Objective function

What is sought to do to solve this problem is to minimize the sum of the difference between the image of the control ColorChecker matrix (I_1), and the product between the image of the ColorChecker matrix that is being calibrated (I_2) with the proposed matrix 'm', this will be taken as the objective function that will serve in the differential evolution algorithm, only the difference will be absolute.

$$f(I_1, I_2, m) = \sum |I_1 - (I_2 \cdot m)|$$

B. Parameters

In this proposal, the representation of the population will be a vector of size 9, since what is sought is to optimize a 3x3 matrix 'm', which is what will be used to calibrate the image.

The bounds that will be used to limit the values proposed by the differential evolution algorithm range from -2 to 2, these values were taken since, when running this algorithm previously, it was possible to observe that they varied between

these values, so they were selected. and they were put to improve the work of the algorithm.

The population size used was 30, with 1000 generations and a crossover probability of 0.5.

C. Process

The differential evolution algorithm is called, and based on the array 'P' obtained, the function is called to calibrate the image based on the array 'P' transformed into the 3x3 matrix 'm'.

$$Calibrated\ Image\ =\ Image\ to\ calibrate\ \cdot m$$

Once this is done, the image is calibrated with respect to the matrix 'm' and the process is finished.

VI. . Results

The proposed algorithm was run 100 times, where the following information could be collected.

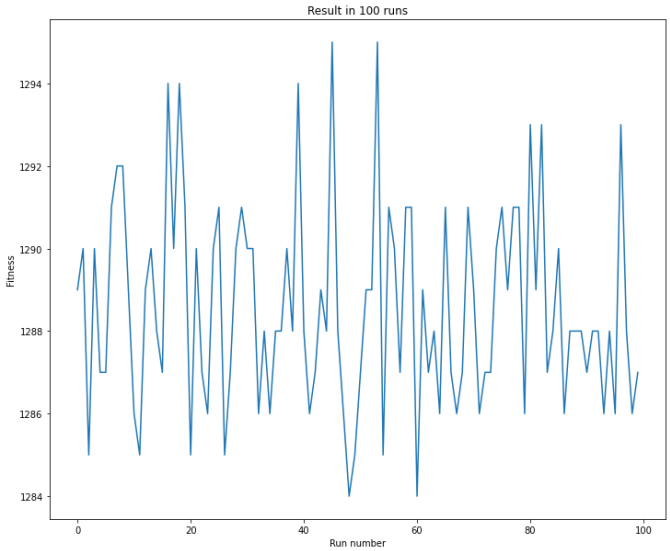

Fig. 6. Result of 100 runs of the algorithm.

Of those 100 given runs, the best result was run number 48, which obtained a fitness of 1284.0, corresponding to the matrix:

$$\begin{pmatrix} 1.4785 & -0.1431 & -0.10365 \\ 0.0565 & 1.4331 & -1.1011 \\ -0.2031 & -0.1527 & 2.0 \end{pmatrix}$$

While the worst result of those 100 runs was attempt number 45, which obtained a fitness of 1295.0 whose matrix was:

$$\begin{pmatrix} 1.5273 & -0.1431 & -0.10365 \\ 0.0565 & 1.4331 & -1.1011 \\ -0.2031 & -0.1527 & 2.0 \end{pmatrix}$$

As can be seen, the results of the extremes are almost identical, both in the fitness and in the matrix, it only varies in the value [0,0] of the matrix; this small difference caused an 11-point variation in fitness, which is very small.

The mean and median of these 100 results are equivalent, the mean was 1288.57 and the median was 1288.0, only 0.57 difference, so this was the most obtained result. While the standard deviation indicates that there was not much variation between results, giving a deviation of 2.467.

The calibrated images based on these calibration matrices compared with the original image are the following:

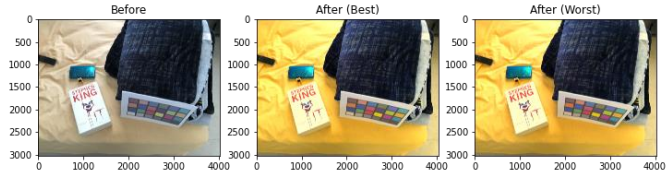

Fig. 7. Comparison of calibrated images.

It can be seen that both the worst and the best solutions obtained very good results. The control matrix, being a digital image, seeks to make the colors more vivid by being 100% faithful to the original. Although a calibration that is identical to the control matrix was not achieved, very positive changes were made.

The result matrices, in comparison with the control matrix and with the matrix that was sought to be calibrated at the beginning, are seen as follows:

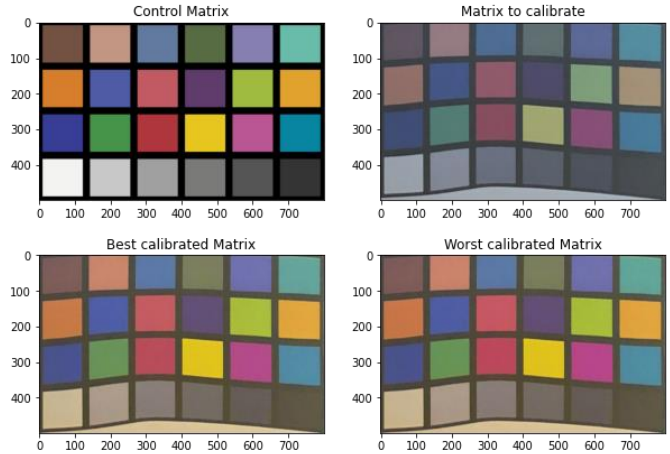

Fig. 8. Comparison of calibrated matrix.

As previously mentioned, both the worst and the best solutions have very good results, so there is not much change.

VII. Conclusion

This methodology is not perfect, but it does give decent results for color calibration. The fitness does not reach zero (perfect calibration), this is probably due to light factors that affect some original colors, or other factors related to the moment of taking the image. Now, the control matrix used is digital, therefore, reaching those tones in real life is practically impossible, since they should be 100% faithful; this also affects the performance

of the algorithm, but a digital image was used to further highlight the colors and make the changes noticeable, since more striking colors would need to be achieved.

Some clear improvements that can be made are when detecting the ColorChecker. A possible quick improvement could be to implement Haar Cascade to detect the pattern of the matrix, since the way it is right now could lead to it taking other elements that are also rectangles but not the ColorChecker, In this way, the effectiveness rate is more ensured when detecting such a matrix.

It could also be interesting to apply it with Particle Swarm Optimization, to visualize what search space it explores and to know if there are differences that can make the difference between applying one algorithm or another.

REFERENCES

1. V. Kamath, C. P. Kurian and U. S. Padiyar, "Development of Bayesian Neural Network Model to Predict the Correlated Color Temperature Using Digital Camera and Macbeth ColorChecker Chart," in *IEEE Access*, vol. 10, pp. 55499-55507, 2022, doi: 10.1109/ACCESS.2022.3177195.
2. Menesatti P, Angelini C, Pallottino F, Antonucci F, Aguzzi J, Costa C. RGB Color Calibration for Quantitative Image Analysis: The "3D Thin-Plate Spline" Warping Approach. *Sensors*. 2012; 12(6):7063-7079. https://doi.org/10.3390/s120607063.
3. C. Nan-qing, L. Xu-yang, Y. Hong-wei, R. Zhi-guang and M. Zi-xuan, "Remote Sensing Image Color Correction Method Based on Automatic Piecewise Polynomial Method," *2020 IEEE 5th Optoelectronics Global Conference (OGC)*, 2020, pp. 161-165, doi: 10.1109/OGC50007.2020.9260437.
4. Nikola Banić. "Unsupervised Learning for Color Constancy". 2017. https://doi.org/10.48550/arXiv.1712.00436.
5. Marrero Fernández, P. D., Guerrero Peña, F. A., Ing Ren, T., & Leandro, J. J. (2019). Fast and robust multiple ColorChecker detection using deep convolutional neural networks. Image and Vision Computing, 81, 15-24. https://doi.org/10.1016/j.imavis.2018.11.001.
6. Berna Ari, Nebras Sobahi, Ömer F. Alçin, Abdulkadir Sengur, U.Rajendra Acharya. Accurate detection of autism using Douglas-Peucker algorithm, sparse coding based feature mapping and convolutional neural network techniques with EEG signals. Computers in Biology and Medicine, Volume 143, 2022, 105311, ISSN 0010-4825. https://doi.org/10.1016/j.compbiomed.2022.105311.
7. GeeksforGeeks. Python. Detect Polygons in an Image using OpenCV. 2019. https://www.geeksforgeeks.org/python-detect-polygons-in-an-image-using-opencv/.
8. OpenCV: Geometric Transformations of Images. 2019. https://docs.opencv.org/4.x/da/d6e/tutorial_py_geometric_transformations.html.
9. Ameca, M. Y. (2016, 28 november). Evolución Diferencial para Resolver Problemas de Optimización Dinámica con Restricciones. Universidad Veracruzana. https://www.uv.mx/.
10. Lynn N, Ali MZ, Suganthan PN. Poluation topologies for particle swarm optimization and differential evolution. Swarm Evol Comput 2018. https://doi.org/10.1016/j.swevo.2017.11.002.
11. Novoa-Hernández, C., Cruz, C. & Pelta, C. D. (2014, december). A comparative study on self-adaptive differential evolution in dynamic environments. Scielo. http://scielo.sld.cu/scielo.php?script=sci_arttext&pid=S2227-18992014000400005.