Solve RGB+ LEDs PWM from Chromaticity

CHINZEI, Kiyoyuki

August 10, 2020

Abstract

This article[1] is a private note to develop `kch-rgbw-lib`, a TypeScript library in github and npm. `Kch-rgbw-lib` provides classes and functions for multicolor LED, including color space conversions between HSV, RGB, XYZ and xyY and calculation of accurate color composition. This article gives a general solution of multi-number (more than R-G-B) LEDs to represent composite colors. It is not intended to carry new, comprehensive, or most efficient ideas. This document is granted under MIT License.

1 Define the Problem

1.1 Past works

Obtaining accurate colors by mixing R-G-B color sources has been utilized as color displays since 1950s. As various colors are available by LED, recent topics are solutions of color composite with additional color sources typically for OLED applications [1, 2]. Usually white light sources are used as an additional light source [1–3]. For display purposes additional colors other than R-G-B have been also used to expand the possible color ranges (gamut) [4]. Sharp once added yellow in Aquos flat-panel TV, but they initially researched 5-primary color display [5]. Amber [6], turquoise, and violet can be other colors to expand the gamut.

1.2 Given parameters and assumptions

We have $N \geq 3$ color sources (LEDs) with chromaticity (x_i, y_i) and maximum luminosity Y_i, where $i = 1 \ldots N$. Our problem is to find the optimum composite output ratio (PWM output) $\boldsymbol{\alpha} = [\alpha_1 \ldots \alpha_N]^T$ where $0 \leq \alpha_i \leq 1$ to represent a given color input with chromaticity (x, y) and luminosity Y.

Here, we set our goal of optimization as the following:

1. Under physical constraint $0 \leq Y \leq Y_1 + \ldots + Y_N$,
2. Obtain exact composite color to represent (x, y, Y).
3. If (x, y) is outside the gamut of color source, the nearest color in it is used. It can be achieved by projecting the input to the gamut contour.

[1] To cite, please refer to https://github.com/kchinzei/kch-rgbw-lib/docs/rgbw_solver.pdf

4. When possible, minimize energy consumption E,

$$E = \sum_{1}^{N} \alpha_i W_i \tag{1.2.1}$$

where W_i is the power of each LED at the maximum luminosity.

5. When possible, average turn-on time of LEDs to equalize LEDs lifetime,

6. When possible, set α_i to null when it's very small. It is preferable to avoid jitter at low PWM output.

It is a typical linear programming (LP) problem. When $N = 3$, e.g. R-G-B color sources only, it's a deterministic and not an optimization problem. And when $N = 4$, e.g. R-G-B-W LEDs, there is only one parameter to optimize, which makes the problem as simple as we don't need to use sophisticated LP solver. We first derive a general description of the problem and solve it for $N = 3$, $N = 4$ and $N > 4$ cases.

2 General solution

We use XYZ color space because a composite of color source in XYZ color space can be obtained as a simple sum of each term. In XYZ color space, our problem is to determine $\boldsymbol{\alpha} = [\alpha_1 \dots \alpha_N]$ which gives an equation between input color $[X, Y, Z]^T$ and color source $[X_i, Y_i, Z_i]^T$, $i = 1 \dots N$;

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \alpha_1 \begin{bmatrix} X_1 \\ Y_1 \\ Z_1 \end{bmatrix} + \dots + \alpha_N \begin{bmatrix} X_N \\ Y_N \\ Z_N \end{bmatrix} \tag{2.0.1}$$

Color space $[X, Y, Z]^T$ is expressed by using (x, y, Y):

$$X = \frac{x}{y} Y \tag{2.0.2}$$

$$Y = Y \tag{2.0.3}$$

$$Z = \frac{1 - x - y}{y} Y \tag{2.0.4}$$

Using matrix representation, Eq. (2.0.1) is written as

$$[\boldsymbol{X}] = [\boldsymbol{A}]\,[\boldsymbol{\alpha}] \tag{2.0.5}$$

where

$$[\boldsymbol{X}] = [X, Y, Z]^T \tag{2.0.6}$$

$$[\boldsymbol{A}] = \begin{bmatrix} \frac{x_1}{y_1} Y_1 & \dots & \frac{x_N}{y_N} Y_N \\ Y_1 & \dots & Y_N \\ \frac{1-x_1-y_1}{y_1} Y_1 & \dots & \frac{1-x_N-y_N}{y_N} Y_N \end{bmatrix} \tag{2.0.7}$$

$$[\boldsymbol{\alpha}] = [\alpha_1, \ \dots \ , \alpha_N]^T \tag{2.0.8}$$

Our goal is to solve Eq. (2.0.5) with respect to $\boldsymbol{\alpha} = [\alpha_1 \dots \alpha_N]^T$. To solve it, we can use the singular value decomposition (SVD) (Eq. (2.0.9)) [7].

$$\begin{bmatrix} \\ \boldsymbol{A} \\ \\ \end{bmatrix} = \begin{bmatrix} \\ \boldsymbol{U} \\ \\ \end{bmatrix} \begin{bmatrix} \omega_1 & & & \\ & \omega_2 & & 0 \dots 0 \\ & & \omega_3 & \end{bmatrix} \begin{bmatrix} \\ \boldsymbol{V}^T \\ \\ \end{bmatrix} \tag{2.0.9}$$

where $\boldsymbol{A}$ is $3 \times N$, $\boldsymbol{U}$ is 3×3, $[\omega_1 \ddots \omega_3, 0 \dots 0]$ is $3 \times N$, $\boldsymbol{V}^T$ is $N \times N$ matrixes [2] in this specific case, since $\boldsymbol{A}$ is a $3 \times N$ matrix and there are upto 3 ω's. When $N \geq 4$, $[\omega_1 \ddots \omega_3]$ is null-padded in $3 \times (N-3)$. The pseudo-inverse matrix $\boldsymbol{A}^{-1}$ is obtained by

$$\begin{bmatrix} \\ \boldsymbol{A} \\ \\ \end{bmatrix}^{-1} = \begin{bmatrix} \\ \boldsymbol{V}_{1-3} \\ \\ \end{bmatrix} \begin{bmatrix} 1/\omega_1 & & \\ & 1/\omega_2 & \\ & & 1/\omega_3 \end{bmatrix} \begin{bmatrix} \\ \boldsymbol{U}^T \\ \\ \end{bmatrix} \tag{2.0.10}$$

where $\boldsymbol{V}_{1-3}$ is the first 3 columns of $\boldsymbol{V}$, those correspond to $\omega_1 \dots \omega_3$ [3]. Using $\boldsymbol{A}^{-1}$, we obtain

$$[\boldsymbol{\alpha}] = [\boldsymbol{A}]^{-1} [\boldsymbol{X}] \tag{2.0.11}$$

By the way, what about the rest of columns in $\boldsymbol{V}$? They are null vectors of $\boldsymbol{A}$. A null vector $\boldsymbol{n}$ of $\boldsymbol{A}$ is such vector that satisfies $\boldsymbol{An} = [0]$. By denoting these columns as $\boldsymbol{n}_4 \dots \boldsymbol{n}_N$, Eq. (2.0.11) can be extended as

$$[\boldsymbol{\alpha}] = [\boldsymbol{A}]^{-1} [\boldsymbol{X}] + \beta_4 \boldsymbol{n}_4 + \dots + \beta_N \boldsymbol{n}_N \tag{2.0.12}$$

where $\beta_4 \dots \beta_N$ are arbitrary numbers. By choosing these using other constraints, we can obtain the optimum solution.

3 Solution of $N = 3$ case

When $N = 3$, Eq. (2.0.11) gives a deterministic solution. No optimization. However, the obtained α_i should be physically meaningful, i.e., $0 \leq \alpha_i \leq 1$. It can be out of range when

- Input color $[X]$ is out of the gamut defined by the color sources,
- Y of $[X]$ is greater (brighter) than the color sources.

When $\alpha_i < 0$, the nearest color in the gamut should be used as the input. We can also truncate such α_i to 0. In this case, the output color has certain error.

When $\alpha_i > 1$, all αs should be normalized by the largest α. This way the color will be correctly obtained, but it will be darker than expected.

[2] Many implementations of SVD return 'economy' $\boldsymbol{V}$ in $N \times 3$ instead of calculating full $N \times N$ size. Matlab, Octave without 'econ' option and svd-js javascript package in NPM with 'f' option (This is my contribution!) give full $\boldsymbol{V}$.

[3] Many implementations of SVD do not sort $\boldsymbol{U}, \boldsymbol{V}$ by ωs. Matlab and Octave do sort. Algorithm in 'Numerical Recipes in C' and svd-js in NPM do not.

4 Solution of $N = 4$ case

When $N = 4$, Eq. (2.0.12) is simple.

$$[\boldsymbol{\alpha}] = [\boldsymbol{A}]^{-1}[\boldsymbol{X}] + \beta_4 \boldsymbol{n}_4 \tag{4.0.1}$$

We will determine parameter β_4 using the assumptions in section 1.2. Since it is always $0 \leq \alpha_i \leq 1$, by solving it for β (hereafter omitting '$_4$'), we obtain the following conditions.

$$\beta \geq \begin{cases} -\frac{b_i}{n_i} & \text{if } n_i > 0 \\ \frac{1-b_i}{n_i} & \text{if } n_i < 0 \end{cases} \tag{4.0.2}$$

$$\beta \leq \begin{cases} \frac{1-b_i}{n_i} & \text{if } n_i > 0 \\ -\frac{b_i}{n_i} & \text{if } n_i < 0 \end{cases} \tag{4.0.3}$$

$$E = \sum_{i=1}^{4} (\beta n_i + b_i) W_i \rightarrow min \tag{4.0.4}$$

where $[b_1, \ldots, b_4]^T = [\boldsymbol{A}]^{-1}[\boldsymbol{X}]$, n_i are the elements of $\boldsymbol{n}$. Eq. (4.0.4) is from Eq. (1.2.1). Finding the largest and smallest values of the right hand side of Eqs. (4.0.2) and (4.0.3), denoted as β_{min} and β_{max}, Eqs. (4.0.2) and (4.0.3) are rewritten as

$$\beta_{min} \leq \beta \leq \beta_{max} \tag{4.0.5}$$

Since Eq. (4.0.4) is rewritten as $E = s_1\beta + s_2$, here s_1 and s_2 are constants determined by calculating the sums in Eq. (4.0.4), optimized β is determined as

$$\beta = \begin{cases} \beta_{min} & \text{if } s_1 = \sum_1^4 n_i W_i > 0 \\ \beta_{max} & \text{else} \end{cases} \tag{4.0.6}$$

4.1 When β is not determined

When $\beta_{min} > \beta_{max}$, there is no feasible answer. Again, there are two cases, when the input color $[\boldsymbol{X}]$ is out of the gamut, or when Y in Eq. (2.0.6) is greater than the color sources. Since $\alpha_i \geq 0$ cannot compromise, we find β that satisfies

$$\beta \geq -\frac{b_i}{n_i} \quad \text{if } n_i > 0 \tag{4.1.7}$$

$$\beta \leq -\frac{b_i}{n_i} \quad \text{if } n_i < 0 \tag{4.1.8}$$

Then we normalize the largest α_i to be 1.

4.2 When α_i is small

To implement the assumption 6 in section 1.2, we can introduce allowance of small α, α_ε, and exchange Eqs. (4.0.2) and (4.0.3) as

$$\beta \geq \frac{\alpha_\varepsilon - b_i}{n_i} \tag{4.2.9}$$

But this also needs to assert if $\beta_{min} \leq \beta_{max}$.

5 Solution of $N > 4$ case

We can optimize Eq. (1.2.1) under constraints of $0 \leq \alpha_i \leq 1$ for Eq. (2.0.12) using a linear programming solution [8]. To use linear programming we need to rewrite our problem into the *normal form*. Normal form of a linear programming problem is

- Objective functions are to be minimized,
- All constraints are equality formulas,
- All variables ≥ 0.

Since our constraints $0 \leq \alpha_i \leq 1$ are unequal and β_i can be negative, we introduce *slack variables* γ_i, δ_i and ϵ_i, ζ_i to replace β.

- Introduce $\gamma_i \geq 0$ such that $b_i + \beta_4 n_{4i} + \ldots + \beta_N n_{Ni} - \gamma_i = 0$,
- Introduce $\delta_i \geq 0$ such that $b_i + \beta_4 n_{4i} + \ldots + \beta_n n_{Ni} + \delta_i = 1$,
- Replace $\beta_i = \epsilon_i - \zeta_i$, such that $\epsilon_i \geq 0$ and $\zeta_i \geq 0$.

After this modification, we have $2N$ equations with $2N$ variables (for γ and δ) and $2(N-3)$ variables (for ϵ and ζ). Using these replacements, the objective function (Eq. (1.2.1)) and Eq. (2.0.12) are rewritten as

$$\sum_{i=1}^{N}(n_{4i}\epsilon_4 - n_{4i}\zeta_4 + \ldots + n_{Ni}\epsilon_N - n_{Ni}\zeta_N + b_i)W_i \quad \rightarrow \quad min \tag{5.0.1}$$

$$n_{4i}\epsilon_4 - n_{4i}\zeta_4 + \ldots + n_{Ni}\epsilon_N - n_{Ni}\zeta_N - \gamma_i + b_i \quad = \quad 0 \tag{5.0.2}$$

$$n_{4i}\epsilon_4 - n_{4i}\zeta_4 + \ldots + n_{Ni}\epsilon_N - n_{Ni}\zeta_N + \delta_i + b_i \quad = \quad 1 \tag{5.0.3}$$

Eq. (5.0.1) is the objective function. Solvers applicable to Eqs. (5.0.1) - (5.0.3) can be found in many numerical packages. `Kch-rgbw-lib` uses linear-program-parser and linear-program-solver in npm.

5.1 When solution infeasible

Linear programming solver may find it's 'infeasible' - no solution to satisfy all constraints in Eqs. (5.0.2) and (5.0.3). Another possibility is that the solution would not converge. Yet again, there are two cases, when the input color $[X]$ is out of the gamut, or when Y in Eq. (2.0.6) is greater (brighter) than the color sources.

When the object brightness is too large, we can find a solution by omitting Eq. (5.0.3). Physically doable solution is to normalize the maximum Y_i to 1.0.

We may also see infeasible case when one or more value in solution is very close to zero and Eq. (5.0.2) fails due to numerical error. If it is the case, you can set a small negative value to the right side of Eq. (5.0.2) instead of 0.

6 More to do

The following sections discuss items not implemented in `kch-rgbw-lib`. We leave them as 'homework' fun.

6.1 Not to compromise brightness Y

When Y in Eq. (2.0.6) is greater (brighter) than the color source's capacity, there is no solution. Here we suggested a (compromised) solution to normalize Y_i. But we can also take another strategy to maintain the goal brightness by compromising the goal color.

This strategy requires computation of gradient of Y in $\boldsymbol{\alpha}$ space.

6.2 LED's lifetime averaging

In section 1.2 'Given parameters and assumptions' we introduced an optional condition 5, 'When possible, average turn-on time of LEDs to equalize LEDs lifetime'. It is equivalent to minimize the variance of $\alpha_1 \dots \alpha_N$. It can be an optimization of a quadratic function.

In $N = 4$ case, we have only one variable β therefore we can optimize only one condition. This means we need to chose between the energy minimization and this condition.

References

[1] Chi Can and Ian Underwood. A compact and efficient method of RGB to RGBW data conversion for oled microdisplays. In *EURODISPLAY 2011*, pages 59–62, 2011.

[2] Chul Lee and Vishal Monga. Power-constrained RGB-to-RGBW conversion for emissive displays. In *IEEE International Conference on Acoustic, Speech and Signal Processing*, pages 1214–1218, 2014.

[3] Brian Tompson, Stephen Allen, and Microchip Technology Inc. High resolution RGB LED color mixing application note. Technical Report AN1562, Microchip Technology Inc., 2014.

[4] Wikipedia contributors. Multi-primary color display, 2020. [Online; accessed 14-June-2020].

[5] 冨沢 一成. 多原色ディスプレイの意義. In **フラットパネルディスプレイの人間工学シンポジウム** *2011*. JEITA, March 2011.

[6] Swathi Sridhar, Ashutosh Tiwari, Namrata Dalvi, and Microchip Technology Inc. RGBA color mixing with bluetooth ® low energy communication. Technical Report AN2026, Microchip Technology Inc., 2016.

[7] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Singular Value Decomposition*, chapter 2.6, pages 59–70. Cambridge University Press, Cambridge, MA, USA, second edition, 1992.

[8] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Linear Programming and the Simplex Method*, chapter 2.6, pages 430–438. Cambridge University Press, Cambridge, MA, USA, second edition, 1992.

MIT License