Project Color Calibration Writeup

Authors

- Caio Cesar Hideo Nakai
- Gabriel Choptian

Instructions

- Describe any interesting decisions you made to write your algorithm.
- Show and discuss the results of your algorithm.
- Feel free to include code snippets, images, and equations.
- Use as many pages as you need, but err on the short side If you feel you only need to write a short amount to meet the brief, th
- **Please make this document anonymous.**

Goal

O objetivo deste trabalho é ajustar a distribuição de intensidade dos *pixels* em uma imagem, de modo que um histograma de um *patch* obtido coincida com o histograma de um *patch* extraído de outra imagem tirada com a câmera posicionada de forma diferente e possivelmente com balanço de branco, brilho e contraste diferentes.

Process

A ideia básica para realizar o ajuste da distribuição de intensidade dos *pixels*, é calcular a diferença de histogramas entre o *patch* da imagem de entrada e o *patch* da imagem que deseja-se obter. Assim, quanto menor a diferença mais parecido serão os histogramas. Para o cálculo da diferença foi utilizada a seguinte fórmula matemática para o cálculo da diferença de histogramas:

$$D(h1, h2) = \sum_{0 \leq k \leq 255} (h1(k) - h2(k))^2$$

Para minimizar a diferença entre os histogramas dos *patches*, foi utilizado o algoritmo de otimização *Simulated Annealing* (AS). O AS aplica as transformações de histograma baseadas em tabelas de conversão para minimizar a diferença entre os histogramas das imagens nas áreas de sobreposição (os *patches*). A tabela de conversão é uma função que

atribui um novo valor para cada nível de cinza de uma imagem. Portanto o problema é encontrar um polinômio para que a diferença entre os histogramas seja a menor possível. Neste trabalho foi utilizado um polinômio de segundo grau, pois utilizando três pontos a probabilidade do AS encontrar resultados melhores é maior, além de convergir mais rapidamente ao resultado esperado.

Problems

O primeiro problema foi a normalização do histograma antes de realizar o cálculo da diferença. O segundo problema foi ajustar os parâmetros do *Simulated Annealing*, pois a temperatura inicial escolhida tornava a constante de aceitação muito alta, fazendo com que o algoritmo aceitasse todas as soluções mesmo quando a temperatura já estivesse ”baixa”. Outro problema era com os resultados (eram mt ruins), porém após debugar função por função descobrimos que o problema ainda estava no *Simulated Annealing*, além disso também alteramos a função de gerar vizinhos aleatórios, isto é, mudamos a função que gerava novos estados para serem testados, alterando somente um único ponto (o do meio) da função polinomial de segundo grau. Finalmente, após alterar a forma como eram feitas as iterações do *loop* principal conseguimos bons resultados.

Interesting Implementation Detail

O código a seguir é a implementação final do *Simulated Annealing*

```
1  function L = simulated_annealing(s0, t0, epsilon, im1,
       im2)
2    i = 0;
3    s = s0;
4    t = t0;
5    k = 0;
6    while(t>epsilon)
7      disp(t);
8      if(k>=150)
9        t = updateTemperature(t0, i);
10       k=0;
11     endif
12     sn = vizinho(s);
13     ls = xau(s,im1);
14     lsn = xau(sn,im1);
15     es = D(ls, im2);
16     esn = D(lsn, im2);
17     dif = esn - es;
18     p = 0;
19
```

```
20      if(dif < 0)
21        p = 1;
22      else
23        p = exp(-(dif)/t);
24      endif
25      q = rand();
26      if(q < p)
27        sn
28        s = sn;
29      endif
30      i = i+1;
31      k = k+1;
32    endwhile
33    L = s;
34  endfunction
```

A Result

A Figura 1 a seguir representa o resultado final obtido após a execução do algoritmo implementado.

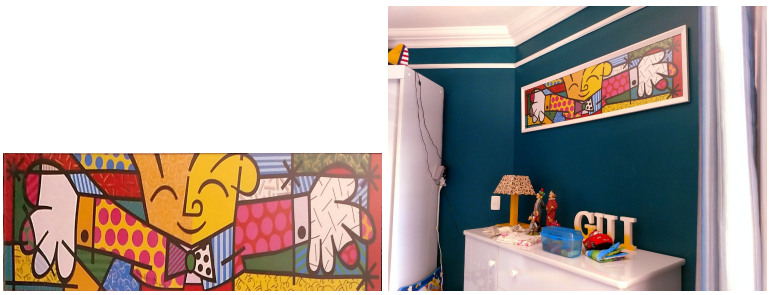

Figure 1: *Esquerda:* Patch. *Direita:* Imagem Final.