

## Imagens Híbridas

### Instruções

O código foi desenvolvido utilizando a linguagem de programação GNU Octave, onde no diretório do arquivo disponibilizado "*Color\_Calibration\_Histogram/code*" encontram -se todo o desenvolvimento da atividade solicitada, na qual em cada trecho do código existem comentários sobre seu funcionamento. Também neste mesmo arquivo foram inseridos as imagens utilizadas para a execução do algoritmo sendo elas: "*img1.png*", "*img1\_patch.png*", "*img2.png*" e "*img2\_patch.png*", podendo todas serem encontradas no diretório "*Color\_Calibration\_Histogram/data*".

Para a execução desta atividade é necessário a instalação dos pacotes "*Image*" e "*optim*" do GNU Octave, uma vez realizado a instalação do pacote, basta carregar o projeto em uma plataforma com suporte ao octave, executar o projeto definido como "*Code.m*" e definir o diretório padrão como sendo "*Color\_Calibration\_Histogram*". Após a execução do código é necessário aguardar um determinado tempo, sendo este tempo proporcional ao valor numérico da variável "*temp*" que representa o número de vezes que o código irá se repetir, em média esta variável com valor de 5 levou de 8 a 10 minutos. Por padrão a variável "*temp*" será iniciada com o valor de 15.

### Detalhes de Implementação

Basicamente todo o código principal encontra -se dentro do arquivo definido como "*Code.m*", além deste arquivo, neste diretório encontram -se outros dois arquivos sendo eles: "*D.m*" e "*Sistema.m*". O primeiro arquivo é referente à função que calcula a distância entre pontos por meio da distância euclidiana, que recebe como argumento dois histogramas referentes as imagens e retorna a soma total da distância entre os pontos. A seguir é apresentado seguinte trecho de código referente ao arquivo "*Sistema.m*":

```

1 function dis = D(u,v)
2   'inicializacao de variaveis
3   dis = 0;
4   [x,y] = size(u);
5
6   for i= (1:x)
7     'acumulo da diferenca ao quadrado de dois pontos
8     dis = dis + (u(x) - v(x))^2;
9   endfor
10 endfunction

```

O arquivo definido como "*Sistema.m*" armazena a função que representa um polinômio de segundo grau, apresentada como:  $ax^2+bx+c = y$ , sendo assim, esta função recebe como argumentos os valores  $x1, y1, x2, y2, x3, y3$ , representando três pontos que em combinação

irão tentar encontrar a melhor curva para o mapeamento de ajuste de cores. Esta função retorna como valor um vetor contendo três valores, representando as três soluções do sistema para os três pontos. A seguir é apresentado o trecho de código referente ao arquivo "*Sistema.m*":

```

1 function res = Sistema(x1,y1,x2,y2,x3,y3)
2     A = [x1^2, x1, 1; x2^2, x2 , 1; x3^2, x3, 1];
3     B = [y1;y2;y3];
4
5     res = A\B;
6 endfunction

```

Para o arquivo definido como "*Code.m*", o processo de inicialização de variáveis é apresentado através do seguinte trecho de código:

```

1 pkg load image;      #load packages
2 pkg load optim;     #load packages
3
4 image01 = imread("data/img1_patch.png");    #load images
5 image02 = imread("data/img2_patch.png");    #load images
6
7 figure('name','Imagen Inicial','numbertitle','off'),
8     imshow(image01);                      #apresentation of original
9     images
10 figure('name','Imagen Para Comparacao','numbertitle','off'
11     ), imshow(image02);                  #apresentation of original
12     images
13
14 image01_out_R = zeros(1,256);    #output channel red
15 image01_out_G = zeros(1,256);    #output channel green
16 iamge01_out_B = zeros(1,256);    #output channel blue
17 [size_X,size_Y,size_Z] = size(image01);        #dimensions
18     of image
19 x1 = 1; y1 = 1;                      #initial
20     point to P1
21 x2 = 50; y2 = 45;                    #initial
22     point to P2
23 x3 = 120; y3 = 100;                  #initial
24     point to P3
25 new_vetor = zeros(1,256,"uint8");    #map color
26     1 to 256
27 temp=5;                            #initial
28     temperature
29 image_final = image01;              #create
30     var to final iterate image
31 best_image = zeros(size_X,size_Y,size_Z);    #create
32     var to best image

```

```

21 diff_red = 0;                                #diff
      perl channel (Red)
22 diff_green = 0;                             #diff
      perl channel (Green)
23 diff_blue = 0;                            #diff
      perl channel (Blue)
24 best_difR = inf(1);                         #var best
      difference init with +inifinite Red
25 best_difG = inf(1);                         #var best
      difference init with +inifinite Green
26 best_difB = inf(1);                         #var best
      difference init with +inifinite Blue
27 new_dif = 0;                                #diff in
      loop

```

Para o processo de reconhecimento simulado, tal algoritmo só é finalizado quando a temperatura definida inicialmente atingir o valor zero, uma vez atingido tal valor, o algoritmo deverá selecionar a imagem que apresentou melhores resultados dentre todas as outras imagens geradas, utilizando a função de cálculo da distância para saber a proximidade entre a imagem que foi gerada em relação a imagem utilizada como comparação, armazenando assim os melhores resultados de cada decremento da temperatura, sendo a imagem utilizada para comparação armazenada pela variável "*image02*".

Para cada redução unitária da variável "*temp*", é percorrido dentre os três canais RGB (Red, Green e Blue) das imagens originais, buscando a diferença entre seus histogramas, uma vez tendo a diferença entre os histogramas das imagens originais separadas por canal é selecionado três pontos de forma aleatória, na qual esses pontos são utilizados para minimizar a diferença entre as cores da imagem original utilizada como comparação, sendo assim tal exemplificação é apresentada através do trecho de código a seguir:

```

1 while (temp > 0)
2   printf("Temperatura: %d ... \n", temp);
3   i=1;                                #
      iterate in channels
4   while (i <= 3)                      #
      repeat to all channels RGB
5     image01_histogram=(hist((image01(:,:,i)),256));    #
      histogram per channel 01
6     image02_histogram=(hist((image02(:,:,i)),256));    #
      histogram per channel 02
7     dif = D(image01_histogram,image02_histogram);      #
      difference between histograms
8     new_dif = 0;                                #
      difference on loop between images
9     saida = zeros(size_X,size_Y);                #output
      at final iteration

```

```
10
11     while (new_dif <= dif)
12         #print iteration
13         printf("Iniciando Iteracao... [%d,%d] [%d,%d] [%d,%d]\n",x1,y1,x2,y2,x3,y3);
14
15         #solution results for all three points
16         solution = Sistema(x1,y1,x2,y2,x3,y3);    #return
17             three points 2function ax^2+bx+c=y
18
19         #create color map
20         for aux= (0:255)
21             new_vetor(aux+1) = (solution(1)*(aux^2)) + (
22                 solution(2)*aux) + (solution(3));    #a b c
23         endfor
24
25         #plot (0:255,new_vetor); #plot color map
26
27         for aux_i= (1:size_X)
28             for aux_j= (1:size_Y)
29                 val_orig = (image01(aux_i,aux_j,i));
30                 val_new = new_vetor(val_orig+1);
31                 saida(aux_i,aux_j) = val_new;
32             endfor
33         endfor
34
35         poinSorted = randi(3);           #point to sort (p1,p2,
36                                         p3)
37         poinSorted_aux = randi(2);    #axis (x,y)
38         switch(poinSorted)
39             case 1
40                 if ((x1 <= 80) || (y1 <= 80))    #p1 interval
41                     (1-80)
42                     if (poinSorted_aux == 1)
43                         x1+=5;
44                     else
45                         y1+=5;
46                     endif
47             else
48                 if (poinSorted_aux == 1)
49                     x1 = randi(85);
50                 else
51                     y1 = randi(85);
52                 endif
53             endif
54         endif
55     endif
56 
```

```
51      case 2
52          if ((x2 <= 165) || (y2 <= 165)) #p2 interval
53              (1-165)
54                  if (poinSorted_aux == 1)
55                      x2+=5;
56                  else
57                      y2+=5;
58                  endif
59          else
60              if (poinSorted_aux == 1)
61                  x2 = randi([85,170]);
62              else
63                  y2 = randi([85,170]);
64              endif
65          endif
66      case 3
67          if ((x3 <= 251) || (y3 <= 251)) #p3 interval
68              (1-251)
69                  if (poinSorted_aux == 1)
70                      x3+=5;
71                  else
72                      y3+=5;
73                  endif
74          else
75              if (poinSorted_aux == 1)
76                  x3 = randi([170,256]);
77              else
78                  y3 = randi([170,256]);
79              endif
80          otherwise
81              printf("Error 0x01\n")
82      endswitch
83
84      new_histogram = imhist(saida);      #out iteration
85      histogram
86      new_dif = D(new_histogram,image02_histogram);      #
87      diff between new histogram and image suport
88      histogram
89  endwhile
90
91  printf("New Channel %d...", i);
92
93  #save each channel in variables
94  switch(i)
95      case 1
```

```
92         image01_out_R = saida;
93         diff_red = new_dif;
94     case 2
95         image01_out_G = saida;
96         diff_green = new_dif;
97     case 3
98         image01_out_B = saida;
99         diff_blue = new_dif;
100    otherwise
101        printf("Error 0x02\n");
102    endswitch
103
104    saida = zeros(size_X,size_Y); #reset var saida
105    i++;                         #incremente channel
106 endwhile
107
108 image_final = cat(3,image01_out_R,image01_out_G,
109                     image01_out_B);
110 figure('name','Final Image Iteration','numbertitle',
111         'off'), imshow(uint8(image_final));
112
113 if ((diff_red < best_difR) && (diff_green < best_difG)
114     && (diff_blue < best_difB))
115     best_difR = diff_red;
116     best_difG = diff_green;
117     best_difB = diff_blue;
118     best_image = image_final;
119 endif
120
121 temp--; #decrease temperature
122 endwhile
```

## Resultados

Inicialmente foram utilizadas duas imagens como entrada, sendo elas: "*data/img1\_patch.png*" e "*data/img2\_patch.png*", a segunda imagem foi escolhida como base de comparação de saída ou seja, a figura gerada deverá se aproximar mais da segunda imagem , tendo como ponto inicial de partida a figura "*img1\_patch.png*". A seguir a figura 1 apresenta as imagens usadas como entrada:

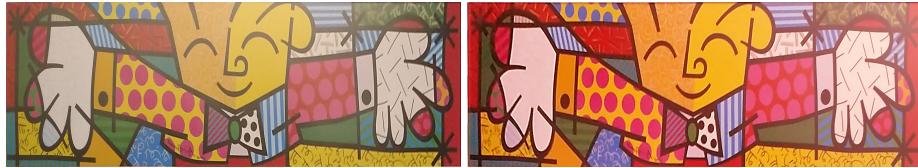


Figure 1: Conjunto das imagens originais, img1\_path à direita e img2\_path à esquerda.

Após carregar ambas as imagens foi necessário calcular o histograma de cada imagem separando em canais vermelho, verde e azul (RGB), sendo assim o histograma gerado contendo os três canais é apresentado a seguir pela figura 2.

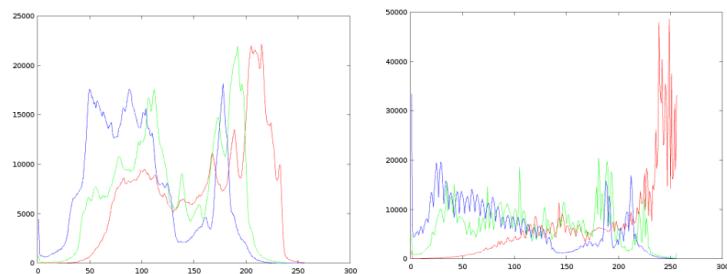


Figure 2: Histogramas referentes as imagens img1\_path e img2\_path respectivamente.

Para a primeira execução, foi escolhido como temperatura inicial o valor 5, após 8 a 10 minutos obtivemos os seguintes resultados como apresentado pelas imagens 3, 4, 5 e 6, não foram colocadas todas as imagens geradas a cada iteração pois foram geradas muitas e apenas as mais relevantes foram selecionadas. A imagem 3 apresentada a seguir é referente à imagem inicial antes da execução do algoritmo.



Figure 3: Imagem inicial usada como entrada do algoritmo.

A imagem apresentada pela figura 4, apresenta a imagem após o primeiro decremento da temperatura. Em comparação à imagem original de entrada é possível perceber uma leve melhora na distribuição das cores, tornando assim a imagem mais nítida.



Figure 4: Imagem gerada após a execução do arquivo "*Code.m*" com temperatura igual a 4.

A imagem seguinte apresentada pela figura 5, é referente ao segundo decremento da temperatura, sendo assim é possível notar que as cores da imagem tornaram -se mais escuras, dando a impressão que a imagem foi gerada em um ambiente com baixa luminosidade, podendo esta descrição ser observada a seguir:



Figure 5: Imagem gerada após a execução do arquivo "*Code.m*" com temperatura igual a 2.

Como resultado final, dentre todas as imagens geradas com temperatura inicial igual a cinco, o algoritmo selecionou a imagem que é apresentada pela figura 6, como sendo a imagem mais próxima da "*img2\_patch.png*" utilizada como comparação.

Após a execução é possível perceber que as cores da imagem gerada em comparação à imagem inicial tornaram -se mais nítidas, aumentando assim o contraste entre as cores.



Figure 6: Resultado da melhor imagem gerada para temperatura iniciada em 5.

Para outro teste realizamos a execução do código com a variável "*temp*" iniciada em 15, na qual esta variável representa o número de iterações que o código irá realizar para encontrar a melhor faixa de cores em cada canal vermelho, verde e azul.

Sendo assim, após a execução, os resultados obtidos levaram em média de 20 à 25 minutos. A seguir serão apresentadas algumas das imagens referentes aos resultados obtidos para temperatura igual à 15. Para a terceira iteração do código com temperatura igual à 12, a imagem resultante em comparação à imagem original apresentou um escurecimento na tonalidade das cores, sendo observada pela figura 7.



Figure 7: Imagem gerada após a execução do arquivo "*Code.m*" com temperatura igual a 12.

Com temperatura igual a 9, a imagem resultante apresentou uma pequena melhora nas cores da imagem, tornando ela um pouco mais clara, sendo assim, tais observações são apresentadas pela figura 8



Figure 8: Imagem gerada após a execução do arquivo "*Code.m*" com temperatura igual a 9.

Após a conclusão do algoritmo a imagem final gerada é apresentada pela figura 9. É possível perceber que houve uma grande saturação das cores, aumentando muito sua intensidade, além de perder completamente a cor laranja no rosto do personagem principal retratado.



Figure 9: Imagem final gerada com temperatura inicial igual a 15.