NATIONAL TAIPEI UNIVERSITY OF TECHNOLOGY

2018 FALL

245765 - ADVANCED DIGITAL IMAGE PROCESSING

HW#3 Grey Level Transformation & Histogram Equalization

Author
106368002 張昌祺
CHANG-QI ZHANG
justin840727@gmail.com

Advisor
電子所
高立人 副教授

October 16, 2018

TAIPEI TECH Since1912 國立臺北科技大學 National Taipei University of Technology

Problem 1 Grey Level Transformation (C/C++) (40%)

a. Enhance the image cat_bright.raw and cat_dark.raw by Power-Law and Piecewise-Linear transformation that learned in class. Show the best parameters, the gray-level transform curve and output images. (Figure, 20%; Discussion, 10%)

Ans

Piecewise-Linear transformation

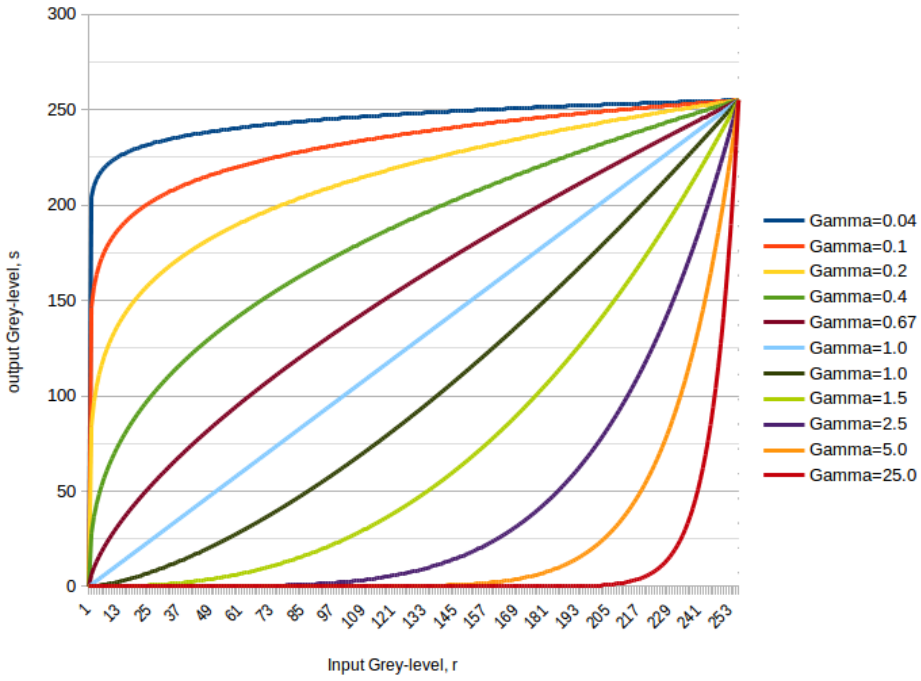


Figure 1: Power-Law Transformation in different Gamma.

(a) Cat bright image source.

(b) Power-Law Transformation.

Figure 2: Power-Law Transformation bright image with best **Gamma 10.0**.

(a) Cat dark image source.

(b) Power-Law Transformation.

Figure 3: Power-Law Transformation bright image with best **Gamma 0.20**.

Piecewise-Linear transformation

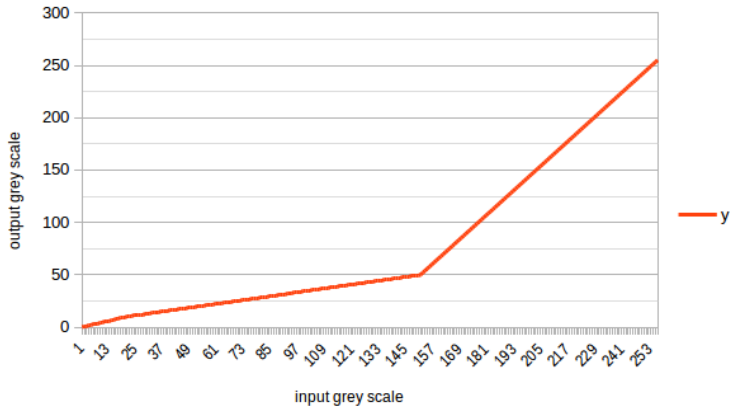


Figure 4: Power-Law Transformation in different Gamma.

(a) Cat bright image source. (b) Transformed image.

Figure 5: Piecewise-Linear transformation bright image with **Figure 4 curve**.

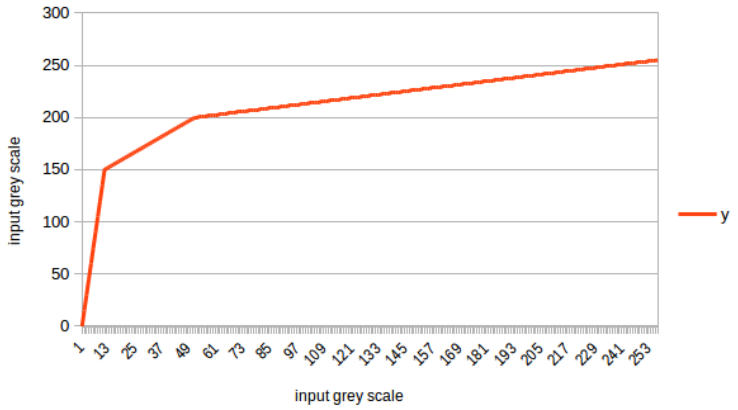


Figure 6: Power-Law Transformation in different Gamma.

(a) Cat dark image source. (b) Transformed image.

Figure 7: Piecewise-Linear transformation dark image with **Figure 6 curve**.

b. Compare and discuss the results obtained by the two methods and explain the difference.(Discussion, 10%)

Ans

Source code for Problem 1

grey_level_transformation.hpp

```
#include <iostream>
#include <fstream>
#include <opencv2/opencv.hpp>
```

```
#include <opencv2/highgui/highgui.hpp>

const std::string SAVE_IMG_FOLDER = "../result_img/";
const double GAMMAS[] = {0.04, 0.1, 0.2, 0.4, 0.67, 1.0, 1.5, 2.5, 5.0, 10.0,
    25.0};

void loadRawFile(cv::Mat &dst_img, std::string file_path, int width, int
    height);
void showImage(std::string win_name, cv::Mat &show_img);
void saveImage(cv::Mat &img, std::string folder, std::string file_name);
double powerLaw(double L, double c, double r, double gamma);
void PowerLawTransformation(cv::Mat &src, cv::Mat &dst, double gamma);
void showAllImages(std::vector<cv::Mat> &list, std::string prefix);
void saveAllImages(std::vector<cv::Mat> &list, std::string floder, std::
    string prefix);
void plotCurves(cv::Mat &plot, std::vector<std::vector<cv::Point2f> >
    curvesPoints);
double linearFunc(uint8_t value, cv::Point2f r1s1, cv::Point2f r2s2, double L
    );
void piecewiseLinearTF(cv::Mat &src_img,
                       cv::Mat &dst_img,
                       cv::Point2f r1s1,
                       cv::Point2f r2s2);
void writeCSV( std::string folder,
               std::string file_name,
               std::vector<std::vector<cv::Point2f> > curvesPoints);
void writeCSV( std::string folder,
               std::string file_name,
               std::vector<cv::Point2f> curvesPoints);
```

grey_level_transformation.cpp

```
#include "grey_level_transformation.hpp"

void loadRawFile(cv::Mat &dst_img, std::string file_path, int width, int
    height)
{
  std::FILE* f = std::fopen(file_path.c_str(), "rb");
  // std::vector<char> buf(width*height);    // char is trivally copyable
  unsigned char buf[width][height];
  std::fread(&buf[0], sizeof buf[0], width*height, f);
  for (int i = 0; i < dst_img.rows; i++)
  {
    for (int j = 0; j < dst_img.cols; j++)
    {
      dst_img.at<char>(i, j) = buf[i][j];
    }
  }
  std::fclose(f);
}
```

```
18
19 void showImage(std::string win_name, cv::Mat &show_img)
20 {
21   static int win_move_x = 50;
22   static int win_move_y = 50;
23   cv::namedWindow(win_name, 0);
24   cv::resizeWindow(win_name, show_img.cols, show_img.rows);
25   cv::moveWindow(win_name, win_move_x, win_move_y);
26   cv::imshow(win_name, show_img); //display Image
27   win_move_x += show_img.cols;
28   if (win_move_x > 1920-256)
29   {
30     win_move_x = 50;
31     win_move_y += (show_img.rows+35);
32   }
33 }
34
35 void saveImage(cv::Mat &img, std::string folder, std::string file_name)
36 {
37   std::string save_file = folder + file_name + ".png";
38   cv::imwrite(save_file, img);
39 }
40
41 double powerLaw(double L, double c, double r, double gamma)
42 {
43   return L * c * pow(r / L, gamma);
44 }
45
46 void PowerLawTransformation(cv::Mat &src, cv::Mat &dst, double gamma)
47 {
48   double c = 1.0;
49   double L = 255;
50   for (int i = 0; i < src.rows; i++)
51   {
52     for (int j = 0; j < src.cols; j++)
53     {
54       double src_value = src.at<unsigned char>(i, j);
55       double dst_value = powerLaw(L, c, src_value, gamma);
56       dst.at<char>(i, j) = (char) dst_value;
57     }
58   }
59 }
60
61 void showAllImages(std::vector<cv::Mat> &list, std::string prefix)
62 {
63   for (int i = 0; i < list.size(); i++)
64   {
65     std::string gamma = std::to_string(GAMMAS[i]);
66     gamma.erase ( gamma.find_last_not_of('0') + 2, std::string::npos );
```

```
67      showImage(prefix + " " + gamma + "gamma", list[i]);
68    }
69  }
70
71  void saveAllImages(std::vector<cv::Mat> &list, std::string floder, std::
        string prefix)
72  {
73    for (int i = 0; i < list.size(); i++)
74    {
75      std::string gamma = std::to_string(GAMMAS[i]);
76      gamma.erase ( gamma.find_last_not_of('0') + 2, std::string::npos );
77      std::string save_file = floder + prefix + gamma + ".png";
78      cv::imwrite(save_file, list[i]);
79    }
80  }
81
82  void plotCurves(cv::Mat &plot, std::vector<std::vector<cv::Point2f> >
        curvesPoints)
83  {
84    for (int i = 0; i < curvesPoints.size(); i++)
85    {
86      cv::Mat curve(curvesPoints[i], true);
87      curve.convertTo(curve, CV_32S); //adapt type for polylines
88      polylines(plot, curve, false, cv::Scalar(255), 2, CV_AA);
89    }
90  }
91
92  double linearFunc(uint8_t value, cv::Point2f r1s1, cv::Point2f r2s2, double L
        = 255)
93  {
94    if (value >= 0 && value < r1s1.x)
95    {
96      double m = r1s1.y / r1s1.x;
97      return m*value;
98    }
99    else if(value >= r1s1.x && value < r2s2.x)
100   {
101     double m = (r1s1.y - r2s2.y) / (r1s1.x - r2s2.x);
102     double c = r1s1.y - m * r1s1.x;
103     return m * value + c;
104   }
105   else if(value >= r2s2.x)
106   {
107     double m = (L - r2s2.y) / (L - r2s2.x);
108     double c = r2s2.y - m * r2s2.x;
109     return m * value + c;
110   }
111   else
112   {
```

```
113     return -1;
114   }
115 }
116
117 void piecewiseLinearTF(cv::Mat &src_img,
118                        cv::Mat &dst_img,
119                        cv::Point2f r1s1,
120                        cv::Point2f r2s2)
121 {
122   for (int i = 0; i < src_img.rows; i++)
123   {
124     for (int j = 0; j < src_img.cols; j++)
125     {
126       dst_img.at<char>(i, j) = linearFunc(src_img.at<char>(i, j), r1s1, r2s2)
      ;
127     }
128   }
129 }
130
131 void writeCSV( std::string folder,
132                std::string file_name,
133                std::vector<std::vector<cv::Point2f> > curvesPoints)
134 {
135   std::ofstream myfile(folder+file_name+".csv");
136   myfile << "x";
137   for (int i = 0; i < curvesPoints.size(); i++)
138   {
139     myfile << ",y" + std::to_string(i);
140   }
141   myfile << std::endl;
142   for (int i = 0; i < curvesPoints[0].size(); i++)
143   {
144     myfile << curvesPoints[0][i].x << ",";
145     for (int j = 0; j < curvesPoints.size(); j++)
146     {
147       myfile << curvesPoints[j][i].y;
148       if (curvesPoints.size()-1 != j)
149       {
150         myfile << ",";
151       }
152     }
153     myfile << std::endl;
154   }
155   myfile.close();
156 }
157 void writeCSV( std::string folder,
158                std::string file_name,
159                std::vector<cv::Point2f> curvesPoints)
160 {
```

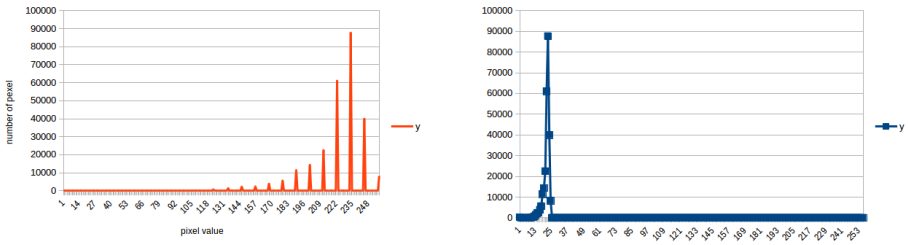
```
  std::ofstream myfile(folder+file_name+".csv");
  myfile << "x,y" << std::endl;
  for (int i = 0; i < curvesPoints.size(); i++)
  {
    myfile << curvesPoints[i].x << "," << curvesPoints[i].y << std::endl;
  }
  myfile.close();
}

int main(int argc, char **argv)
{
  cv::Mat cat_b_src(256, 256, CV_8UC1);
  cv::Mat cat_d_src(256, 256, CV_8UC1);
  loadRawFile(cat_b_src, "../images/cat_bright.raw", 256, 256);
  loadRawFile(cat_d_src, "../images/cat_dark.raw", 256, 256);

  // Power-Law Transformation
  std::vector<cv::Mat> cat_b_img_lst;
  std::vector<cv::Mat> cat_d_img_lst;
  std::vector<std::vector<cv::Point2f> > curvesPoints;
  for (int i = 0; i < sizeof(GAMMAS)/sizeof(double); i++)
  {
    cv::Mat cat_b_transtormed(256, 256, CV_8UC1);
    cv::Mat cat_d_transtormed(256, 256, CV_8UC1);
    PowerLawTransformation(cat_b_src, cat_b_transtormed, GAMMAS[i]);
    PowerLawTransformation(cat_d_src, cat_d_transtormed, GAMMAS[i]);
    cat_b_img_lst.push_back(cat_b_transtormed);
    cat_d_img_lst.push_back(cat_d_transtormed);
    // insert data to curve
    std::vector<cv::Point2f> curvePoints;
    for (int j = 0; j < 256; j++)
    {
      cv::Point2f point(j, powerLaw(255, 1.0, j, GAMMAS[i]));
      curvePoints.push_back(point);
    }
    curvesPoints.push_back(curvePoints);
  }
  cv::Mat plot_img(256, 256, CV_8UC1, cv::Scalar(0));
  plotCurves(plot_img, curvesPoints);

  // Piecewise-Linear Transformation
  cv::Mat cat_b_plt(256, 256, CV_8UC1);
  cv::Mat cat_d_plt(256, 256, CV_8UC1);
  piecewiseLinearTF(cat_b_src, cat_b_plt, cv::Point2f(20,10), cv::Point2f
    (150,50));
  piecewiseLinearTF(cat_d_src, cat_d_plt, cv::Point2f(10,150), cv::Point2f
    (50,200));
  std::vector<cv::Point2f> piecewise_curve_bright;
  std::vector<cv::Point2f> piecewise_curve_dark;
```

```
208   for (int i = 0; i < 256; i++)
209   {
210     piecewise_curve_bright.push_back(cv::Point2f(i, linearFunc(i, cv::Point2f
        (20, 10), cv::Point2f(150,50))));
211     piecewise_curve_dark.push_back(cv::Point2f(i, linearFunc(i, cv::Point2f
        (10, 150), cv::Point2f(50,200))));
212   }
213
214   // showImage("Power Law", plot_img);
215   // showAllImages(cat_b_img_lst, "cat b");
216   // showAllImages(cat_d_img_lst, "cat d");
217   // showImage("src cat bright", cat_b_src);
218   // showImage("PLT cat bright", cat_b_plt);
219   // showImage("src cat dark", cat_d_src);
220   // showImage("PLT cat dark", cat_d_plt);
221   writeCSV("../result_plot_data/", "Power-Law", curvesPoints);
222   writeCSV("../result_plot_data/", "piecewise_curve_bright",
        piecewise_curve_bright);
223   writeCSV("../result_plot_data/", "piecewise_curve_dark",
        piecewise_curve_dark);
224   saveAllImages(cat_b_img_lst, "../result_img/problem1/power_law/", "
        cat_bright");
225   saveAllImages(cat_d_img_lst, "../result_img/problem1/power_law/", "cat_dark
        ");
226   saveImage(cat_b_src, "../result_img/problem1/", "cat_bright_src");
227   saveImage(cat_b_plt, "../result_img/problem1/", "cat_bright_plt");
228   saveImage(cat_d_src, "../result_img/problem1/", "cat_dark_src");
229   saveImage(cat_d_plt, "../result_img/problem1/", "cat_dark_plt");
230   cv::waitKey(0);
231   return 0;
232 }
```

Problem 2 Histogram Equalization (C/C++) (60%)

a. Plot the histogram of livingroom_bright.raw and livingroom_dark.raw. Discuss the difference among these histograms. (Figure, 10%; Discussion, 10%)

Ans

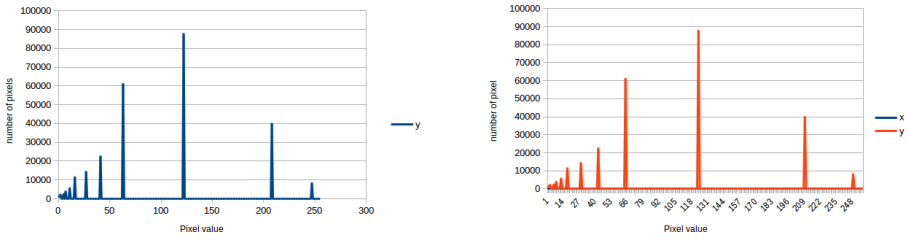


(a) Living room bright. (b) Living room dark.

Figure 8: Histofram of living room image

b. Perform histogram equalization on livingroom_bright.raw and livingroom_dark.raw. Plot their histograms after equalization and compare the results, will the result be the same and why? (Figure, 10%; Discussion, 10%)

Ans



(a) Living room bright. (b) Living room dark.

Figure 9: Histofram of living room image

c. If you perform histogram equalization on cat_bright.raw and cat_dark.raw, will the result look good? Show the output images and explain what causes this situation and how to improve it. (Figure, 10%, Discussion, 10%)

Ans

(a) Cat dark image source.

(b) Transformed image.

Figure 10: Piecewise-Linear

(a) Cat dark image source.

(b) Transformed image.

Figure 11: Piecewise-Linear

Source code for Problem 1

histogram_equalization.hpp

```
#include <iostream>
#include <fstream>
```

```cpp
#include <opencv2/opencv.hpp>
#include <opencv2/highgui/highgui.hpp>

class HistogramEq
{
  public:
  std::vector<int> histogram;
  std::vector<int> cdf;
  std::vector<int> v_map;
  int cdf_min;
  int L;
  int pixel_num;
  HistogramEq(cv::Mat &src_img, int L);
  std::vector<int> getHistogram(cv::Mat &src_img, int L);
  std::vector<int> getCDF();
  int getHv(int v);
  std::vector<int> getEqHistofram();
  void ComputeVmap(std::vector<int> &v_map);
  cv::Mat getEqImage(cv::Mat &img_src);
};
```

histogram_equalization.cpp

```cpp
#include "histogram_equalization.hpp"

void loadRawFile(cv::Mat &dst_img, std::string file_path, int width, int
    height)
{
  std::FILE* f = std::fopen(file_path.c_str(), "rb");
  // std::vector<char> buf(width*height);    // char is trivally copyable
  unsigned char buf[width][height];
  std::fread(&buf[0], sizeof buf[0], width*height, f);
  for (int i = 0; i < dst_img.rows; i++)
  {
    for (int j = 0; j < dst_img.cols; j++)
    {
      dst_img.at<char>(i, j) = buf[i][j];
    }
  }
  std::fclose(f);
}

void showImage(std::string win_name, cv::Mat &show_img)
{
  static int win_move_x = 50;
  static int win_move_y = 50;
  cv::namedWindow(win_name, 0);
  cv::resizeWindow(win_name, show_img.cols, show_img.rows);
  cv::moveWindow(win_name, win_move_x, win_move_y);
  cv::imshow(win_name, show_img); //display Image
```

```
27    win_move_x +=  show_img.cols;
28    if (win_move_x > 1920-256)
29    {
30      win_move_x = 50;
31      win_move_y += (show_img.rows+35);
32    }
33  }
34
35  void writeCSV( std::string folder,
36                 std::string file_name,
37                 std::vector<int> curvesPoints)
38  {
39    std::ofstream myfile(folder+file_name+".csv");
40    myfile << "x,y" << std::endl;
41    for (int i = 0; i < curvesPoints.size(); i++)
42    {
43      myfile << i << "," << curvesPoints[i] << std::endl;
44    }
45    myfile.close();
46  }
47
48  void saveImage(cv::Mat &img, std::string folder, std::string file_name)
49  {
50    std::string save_file = folder + file_name + ".png";
51    cv::imwrite(save_file, img);
52  }
53
54  HistogramEq::HistogramEq(cv::Mat &src_img, int L=256)
55  {
56    this->histogram = this->getHistogram(src_img, L);
57    this->cdf = this->getCDF();
58    this->L = L;
59    this->pixel_num = src_img.cols * src_img.rows;
60    this->ComputeVmap(this->v_map);
61  }
62
63  std::vector<int> HistogramEq::getHistogram(cv::Mat &src_img, int L=256)
64  {
65    std::vector<int> his(L, 0.0);
66    for (int i = 0; i < src_img.rows; i++)
67    {
68      for (int j = 0; j < src_img.cols; j++)
69      {
70        his[src_img.at<uint8_t>(i, j)] += 1.0;
71      }
72    }
73    return his;
74  }
75
```

```
76  std::vector<int> HistogramEq::getCDF()
77  {
78    std::vector<int> his_src = this->histogram;
79    std::vector<int> cdf(his_src.size());
80    int cdf_count = his_src[0];
81    int cdf_min = 0;
82    for (int i = 0; i < his_src.size(); i++)
83    {
84      cdf[i] = cdf_count;
85      if (cdf_min == 0 && cdf_count != 0) cdf_min = cdf_count;
86      cdf_count += his_src[i];
87    }
88    this->cdf_min = cdf_min;
89    return cdf;
90  }
91
92  int HistogramEq::getHv(int v)
93  {
94    return ((double)(this->cdf[v] - this->cdf_min) / (double)(this->pixel_num))
       * (this->L - 1);
95  }
96
97  std::vector<int> HistogramEq::getEqHistofram()
98  {
99    std::vector<int> eq_his(this->histogram.size());
100   for (int i = 0; i < eq_his.size(); i++)
101   {
102     eq_his[this->getHv(i)] = this->histogram[i];
103   }
104   return eq_his;
105 }
106
107 void HistogramEq::ComputeVmap(std::vector<int> &v_map)
108 {
109   v_map.resize(this->histogram.size());
110   for (int i = 0; i < this->histogram.size(); i++)
111   {
112     v_map[i] = this->getHv(i);
113   }
114 }
115
116 cv::Mat HistogramEq::getEqImage(cv::Mat &img_src)
117 {
118   cv::Mat eq_img(img_src.rows, img_src.cols, CV_8UC1);
119   for (int i = 0; i < img_src.rows; i++)
120   {
121     for (int j = 0; j < img_src.cols; j++)
122     {
123       eq_img.at<char>(i, j) = this->v_map[img_src.at<uint8_t>(i, j)];
```

```
124       }
125     }
126     return eq_img;
127 }
128 
129 int main(int argc, char **argv)
130 {
131     cv::Mat lvroom_b_src(512, 512, CV_8UC1);
132     cv::Mat lvroom_d_src(512, 512, CV_8UC1);
133     loadRawFile(lvroom_b_src, "../images/livingroom_bright.raw", 512, 512);
134     loadRawFile(lvroom_d_src, "../images/livingroom_dark.raw", 512, 512);
135     HistogramEq hiseq_living_b = HistogramEq(lvroom_b_src);
136     HistogramEq hiseq_living_d = HistogramEq(lvroom_d_src);
137     writeCSV("../result_plot_data/", "livingRoomBrightHis", hiseq_living_b.
        histogram);
138     writeCSV("../result_plot_data/", "livingRoomDarkHis", hiseq_living_d.
        histogram);
139     writeCSV("../result_plot_data/", "livingRoomBrightEqHis", hiseq_living_b.
        getEqHistofram());
140     writeCSV("../result_plot_data/", "livingRoomDarkEqHis", hiseq_living_d.
        getEqHistofram());
141     cv::Mat lvroom_b_eq_img = hiseq_living_b.getEqImage(lvroom_b_src);
142     cv::Mat lvroom_d_eq_img = hiseq_living_d.getEqImage(lvroom_d_src);
143     saveImage(lvroom_b_src, "../result_img/problem2/", "livingroom_bright_src")
        ;
144     saveImage(lvroom_b_eq_img, "../result_img/problem2/", "
        livingroom_eq_bright_src");
145     saveImage(lvroom_d_src, "../result_img/problem2/", "livingroom_dark_src");
146     saveImage(lvroom_d_eq_img, "../result_img/problem2/", "
        livingroom_eq_dark_src");
147     showImage("livingroom bright", lvroom_b_src);
148     showImage("livingroom eq bright", lvroom_b_eq_img);
149     showImage("livingroom dark", lvroom_d_src);
150     showImage("livingroom eq dark", lvroom_d_eq_img);
151     cv::waitKey(0);
152     return 0;
153 }
```