数字媒体处理综合实验

姓名： 桑燊

班级： 2014211602

学号： 2014212128

时间： 2017年03月10日

实验目标

- 在实验二的基础上，实现如下图像处理的功能功能：
  - 彩色图像转为灰度图像
  - 彩色图像颜色反转
  - 图像做log变换
  - 图像做灰度直方图均衡
  - 图像做RGB三通道分别直方图均衡
  - 图像做HSI色域直方图均衡

实验步骤

彩色图像转灰度图

1. 在原工程的基础上，新建一个新的函数，用于获取当前正在显示的图片：

```
bool MainWindow::getDisplayImage(QImage & img) const
{
    if(gs->items().length() != 0){

        QList<QGraphicsItem *> list = ui->graphicsView-
                              >scene()->items();
        QGraphicsPixmapItem * item =
                              (QGraphicsPixmapItem*)list.at(0);
        img = item->pixmap().toImage();

        return true;
    }
    return false;
}
```

   其中，返回true表示当前获取到了图片，false表示当前没有在显示的图片。

2. 在ImageProcessing类中添加函数 `rgb2gray(const QImage & img)` 用于实现图像RGB转灰度，代码如下：

```
QImage ImageProcessing::rbg2gray(const QImage & img)
{
    // 复制一个新的图像
    QImage ret(img);
    for(int i = 0; i< ret.width(); i++){
        for(int j = 0; j < ret.height(); j++){
            // 获取当前图像的颜色
            QRgb rgb = img.pixel(i, j);

            // 转为灰度
            int grayValue = qGray(rgb);
            ret.setPixelColor(i, j, qRgb(grayValue, grayValue, grayValue));
        }
    }
    return ret;
}
```

该方法输入参数为一个QImage类型的引用且为常量，防止对输入的图片产生变化，返回处理后的新图像。

3. 在menuBar中创建新的action，添加triggered()信号触发的槽函数：

```
void MainWindow::on_actionrgb2gray_triggered()
{
    QImage img;

    if(getDisplayImage(img)){
        img = ImageProcessing::rbg2gray(img);
        showImage(img);
    }
}
```

这样RGB转灰度图就完成了。

彩色图像颜色反转

1. 实现过程与上述相同，只是在ImageProcessing中添加一个新的处理函数：

```
QImage ImageProcessing::pixelReverse(const QImage & img)
{
    QImage ret(img);
    for(int i = 0; i< ret.width(); i++){
        for(int j = 0; j < ret.height(); j++){
            QRgb rgb = img.pixel(i, j);

            ret.setPixelColor(i, j, qRgb(-qRed(rgb), -qGreen(rgb), -qBlue(rgb)
));
        }
    }
    return ret;
}
```

2. 添加槽函数（不再说明）。

图像的log变换

1. 由于在做log变换的时候需要手动输入参数，所以需要一个弹窗。先创建一个新的Qt设计师界面类，命名为 `LogTransfromationDlg` ，先给界面添加一个 `LineEdit` 用于手动输入参数，再给界面上的**OK**按钮添加一个accepted信号触发的槽函数：

```
void LogTransformationDlg::on_buttonBox_accepted()
{
    // isOK用于检验QString转double是否成果
    bool * isOk;
    double c = ui->lineEdit->text().toDouble(isOk);

    // std::cout << c << std::endl;
    if(isOk){

        // std::cout << "OK" << std::endl;
        emit confirmed(c);
    }
}
```

   该槽函数首先获取LineEdit中的内容，将QString转为double，然后抛出一个 `confirmed(double)` 信号。

   confirmed(double)是一个自定义信号，double表示信号携带一个double类型的参数。

2. 对于主界面中log变换对应的action，槽函数调用 `exec()` 弹出并显示刚才定义的窗口（要先实例化一个 `LogTransformDlg` 对象）。

3. 在MainWindow中添加一个同于接受上述confirmed()信号的槽函数：

```
void MainWindow::on_confirmed_accepted(double c)
{
    // qDebug() << "accepted number is:" <<c << endl;

    QImage img;

    if(getDisplayImage(img)){
        img = ImageProcessing::logTransformation(img, c);
        showImage(img);
    }
}
```

4. 在MainWindow的构造函数中绑定连接（亦可将该连接放在log变换action的槽函数中）：

```
connect(ltDlg,SIGNAL(confirmed(double)),this,SLOT(on_confirmed_accepted(double
)));
```

5. 接下来实现log变换的核心算法：

```
QImage ImageProcessing::logTransformation(const QImage & img ,int c = 1)
{
    QImage ret(img);

    for(int i = 0; i< ret.width(); i++){
        for(int j = 0; j < ret.height(); j++){

            QRgb rgb = img.pixel(i, j);

            int grayValue = c * log(qGray(rgb)/255.0 + 1) * 255;
            ret.setPixelColor(i, j, qRgb(grayValue, grayValue, grayValue));
        }
    }
    return ret;
}
```

这样，log变换就完成了。

灰度直方图均衡

1. 这里实现的方式与之前相同，下面给出直方图均衡算法：

```
QImage ImageProcessing::histEquilibrium(const QImage & img){

    QImage ret(img);

    int width = img.width();
    int height = img.height();
    int N = width * height;

    // 统计各级灰度
    int hist[256];
    std::fill(hist, hist + 256, 0);

    for(int i = 0; i < width; i++){
        for(int j = 0; j < height; j++){
            hist[qGray(img.pixel(i, j))]++;
        }
    }

    // 累计灰度计算
    int map[256];
    double sum = 0;
    for(int i = 0; i < 256; i++){
        sum += hist[i];
        map[i] = round(sum / N * 255);
    }

    // 映射
    for(int i = 0; i < width; i++){
        for(int j = 0; j < height; j++){
            int g = map[qGray(img.pixel(i, j))];
            ret.setPixel(i, j, qRgb(g, g, g));
        }
    }
    return ret;
}
```

2. 添加槽函数与之前类似。

RGB通道直方图均衡

提供两种实现方法：

1. 直接对三个通道分别进行上述算法同样的计算：

```
QImage ImageProcessing::histEquilibriumForRgb(const QImage & img){

    QImage ret(img);

    int width = img.width();
    int height = img.height();
    int N = width * height;

    // count histgram
    int hist[3][256];
    std::fill(hist[0], hist[0] + 3 * 256, 0);

    for(int i = 0; i < width; i++){
        for(int j = 0; j < height; j++){
            hist[0][qRed(img.pixel(i, j))]++;
            hist[1][qGreen(img.pixel(i, j))]++;
            hist[2][qBlue(img.pixel(i, j))]++;
        }
    }

    // calculate
    int map[3][256];
    double sum[3] = {0};
    for(int i = 0; i < 256; i++){
        sum[0] += hist[0][i];
        sum[1] += hist[1][i];
        sum[2] += hist[2][i];

        map[0][i] = round(sum[0] / N * 255);
        map[1][i] = round(sum[1] / N * 255);
        map[2][i] = round(sum[2] / N * 255);
    }

    // map the pixel
    for(int i = 0; i < width; i++){
        for(int j = 0; j < height; j++){
            int r = map[0][qRed(img.pixel(i, j))];
            int g = map[1][qGreen(img.pixel(i, j))];
            int b = map[2][qBlue(img.pixel(i, j))];
            ret.setPixel(i, j, qRgb(r, g, b));
        }
    }
    return ret;
}
```

2. 对每个通道分别进行存储然后调用现有的灰度直方图均衡算法计算，计算后再叠加在一起：

```
Image ImageProcessing::histEquilibriumForRgbNoUse(const QImage & img){

    QImage ret(img);

    QImage imgR(img);
    QImage imgG(img);
    QImage imgB(img);

    for(int i = 0; i < img.width(); i++){
        for(int j = 0; j < img.height(); j++){

            int r = qRed(img.pixel(i, j));
            int g = qGreen(img.pixel(i, j));
            int b = qBlue(img.pixel(i, j));

            imgR.setPixel(i, j, qRgb(r, r, r));
            imgG.setPixel(i, j, qRgb(g, g, g));
            imgB.setPixel(i, j, qRgb(b, b, b));
        }
    }

    imgR = histEquilibrium(imgR);
    imgG = histEquilibrium(imgG);
    imgB = histEquilibrium(imgB);

    for(int i = 0; i < img.width(); i++){
        for(int j = 0; j < img.height(); j++){

            int r = qGray(imgR.pixel(i, j));
            int g = qGray(imgG.pixel(i, j));
            int b = qGray(imgB.pixel(i, j));

            ret.setPixel(i, j, qRgb(r, g, b));
        }
    }

    return ret;
}
```

第二种方法在实现上相对更便利，可以直接调用已有的算法，但是在性能上比第一种较差，不推荐使用。

HSI通道直方图均衡

这种均衡的主要思想是将RGB颜色空间转为HSI空间，然后对I分量进行直方图均衡，均衡后再转回RGB空间进行显示。

1. 定义新的结构体存储一个HSI的值：

```
struct HSI{
    double h;
    double s;
    double i;
};
```

2. 实现RGB颜色空间向HSI空间转化：

```
HSI ImageProcessing::Rgb2Hsi(const QRgb rgb){

    HSI hsi;

    // 归一化
    double R = qRed(rgb) / 255.0;
    double G = qGreen(rgb) / 255.0;
    double B = qBlue(rgb) / 255.0;

    // 求范围
    double min = std::min(std::min(R, G), B);
    double max = std::max(std::max(R, G), B);
    double deltaMax = max - min;

    double H;
    double S;
    double I = (max + min) / 2;

    if (deltaMax == 0 ){
        H = 0;
        S = 0;
    }else{
        H = 0;
        if (I < 0.5)
            S = deltaMax / (max + min);
        else
            S = deltaMax / (2 - max - min);

        double deltaR = (((max - R) / 6.0) + (deltaMax / 2.0)) / deltaMax;
        double deltaG = (((max - G) / 6.0) + (deltaMax / 2.0)) / deltaMax;
        double deltaB = (((max - B) / 6.0) + (deltaMax / 2.0)) / deltaMax;

        if (R == max){
            H = deltaB - deltaG;
        }else if (G == max){
            H = 1 / 3.0 + deltaR - deltaB;
        }else if (B == max){
```

```
            H = 2 / 3.0 + deltaG - deltaR;
        }

        if (H < 0)
            H += 1;
        if (H > 1)
            H -= 1;
    }

    hsi.h = H;
    hsi.s = S;
    hsi.i = I;

    return hsi;
}
```

3. 实现HSI向RGB空间转化：

```
QRgb ImageProcessing::Hsi2Rgb(const HSI hsi){

    double H = hsi.h;
    double S = hsi.s;
    double I = hsi.i;

    int R = 0;
    int G = 0;
    int B = 0;

    double v1 = 0;
    double v2 = 0;

    if (S == 0) {
        R = I * 255;
        G = I * 255;
        B = I * 255;
    }else {
        if (I < 0.5){
            v2 = I * (1 + S);
        }else{
            v2 = (I + S) - (S * I);
        }

        v1 = 2 * I - v2;

        R = 255 * ImageProcessing::Hue2Rgb(v1, v2, H + 1/3.0);
        G = 255 * ImageProcessing::Hue2Rgb(v1, v2, H);
        B = 255 * ImageProcessing::Hue2Rgb(v1, v2, H - 1/3.0);
    }
    return qRgb(R, G, B);
}
```

其中Hue2Rgb实现如下：

```
double ImageProcessing::Hue2Rgb(double v1, double v2,double vH)
{
    if (vH < 0)
        vH += 1;
    if (vH > 1)
        vH -= 1;

    if ((6 * vH) < 1)
        return v1 + (v2 - v1) * 6 * vH;

    if ((2 * vH) < 1)
        return v2;

    if ((3 * vH) < 2)
        return v1 + (v2 - v1) * (2 / 3.0 - vH) * 6;

    return v1;
}
```

4. 直方图均衡：

```
QImage ImageProcessing::histEquilibriumByHSI(const QImage & img){

    QImage ret(img);

    int width = img.width();
    int height = img.height();
    int N = width * height;

    // count histgram
    int hist[256];
    std::fill(hist, hist + 256, 0);

    for(int i = 0; i < width; i++){
        for(int j = 0; j < height; j++){

            // 此处I的取值范围为0-1，所以需要乘以255
            hist[(int)(ImageProcessing::Rgb2Hsi(img.pixel(i, j)).i * 255)]++;
        }
    }

    // calculate
    int map[256];
    double sum = 0;
    for(int i = 0; i < 256; i++){
        sum += hist[i];
        map[i] = round(sum / N * 255);
    }

    // map the pixel
    for(int i = 0; i < width; i++){
        for(int j = 0; j < height; j++){
            HSI temp = ImageProcessing::Rgb2Hsi(img.pixel(i,j));
            temp.i = map[(int)(ImageProcessing::Rgb2Hsi(img.pixel(i, j)).i * 2
55)] / 255.0;

            ret.setPixel(i, j, ImageProcessing::Hsi2Rgb(temp));
        }
    }
    return ret;
}
```

5. 添加槽函数，同上。

实验结果

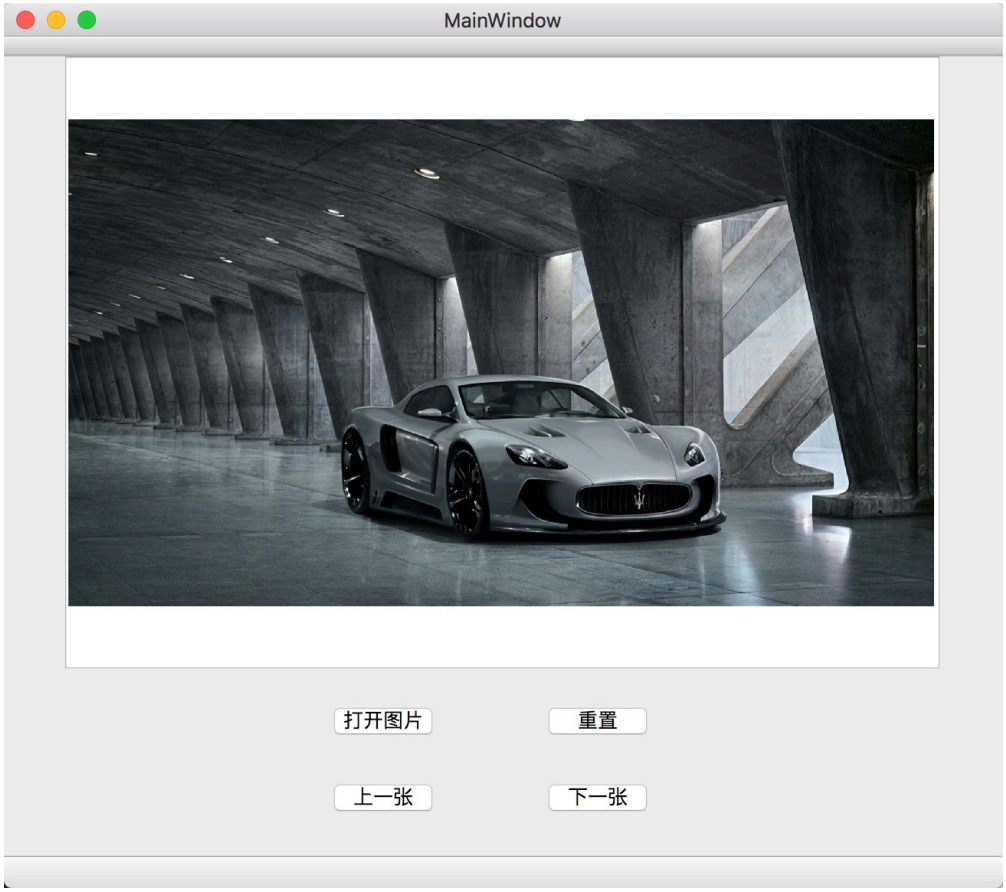

图1. 原图

彩色图像转为灰度图像

图2. 转换后的灰度图像

彩色图像颜色反转

图3. 颜色反转

图像做log变换

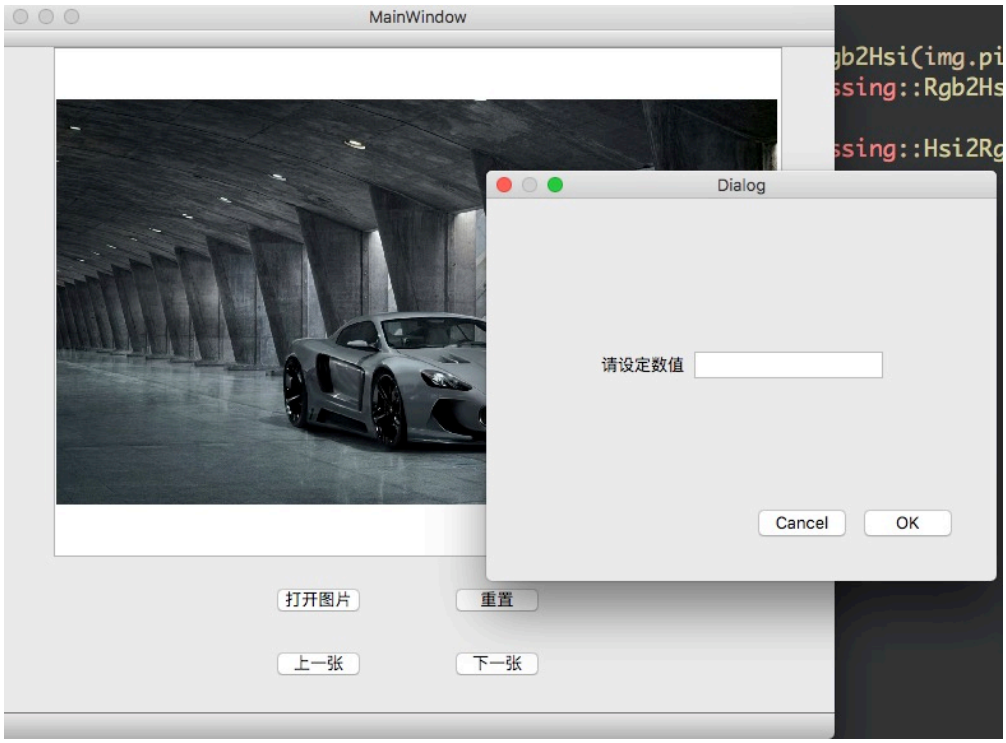

图4. 输入一个参数

图5. log变换后的结果

图像做灰度直方图均衡

图6. 灰度直方图均衡

图像做RGB三通道分别直方图均衡

图7. 三通道直方图均衡

图像做HSI色域直方图均衡

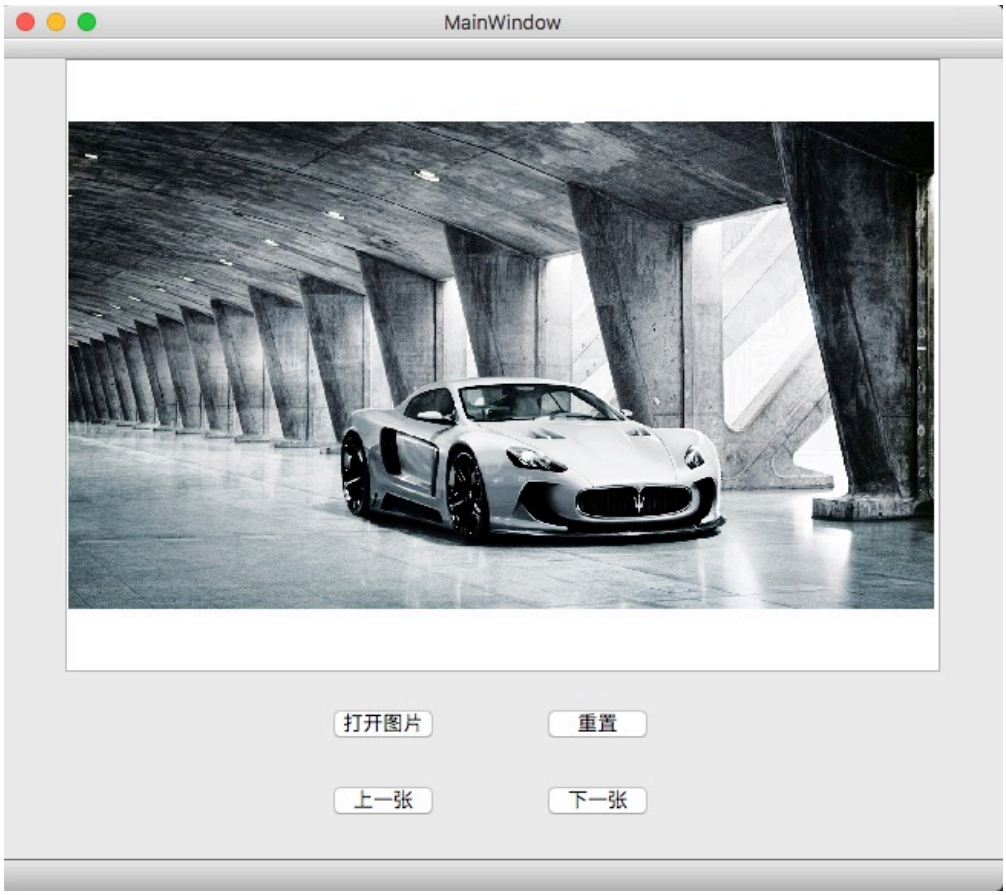

图8. HSI色域直方图均衡