Principles and Applications of Digital Image Processing

B09611007
陳柏霖

【Fall, 2024】

Homework 2

Part 1: (30%)

Solve the problems 2.12, 2.16, 2.18, 2.37, 3.12, 3.18 in the textbook.

2.12* Suppose that a flat area with center at (x_0, y_0) is illuminated by a light source with intensity distribution

$$i(x,y) = Ke^{-[(x-x_0)^2 + (y-y_0)^2]}$$

Assume for simplicity that the reflectance of the area is constant and equal to 1.0, and let $K = 255$. If the intensity of the resulting image is quantized using k bits, and the eye can detect an abrupt change of eight intensity levels between adjacent pixels, what is the highest value of k that will cause visible false contouring?

illumination reflectance

$f(x,y) = i(x,y) \cdot r(x,y)$

$= 255 e^{-[(x-x_0)^2+(y-y_0)^2]}$

$(0 \sim 255)$

False contouring:

$\frac{256}{2^k} > 8$

$\Rightarrow 2^{8-k} > 2^3 \Rightarrow k < 5$ #

2.16 Develop an algorithm for converting a one-pixel-thick m-path to a 4-path.

1. identify all pixels in the m-path
2. examine each pixel
3. replace diagonal paths with two-step process that adheres to 4-adjacency

m-path

1 0 0
0 1 0
0 0 1

4-path

1 1 0
0 1 1
0 0 1

2.18 Consider the image segment shown in the figure that follows.

(a)* As in Section 2.5, let $V = \{0,1\}$ be the set of intensity values used to define adjacency. Compute the lengths of the shortest 4-, 8-, and m-path between p and q in the following image. If a particular path does not exist between these two points, explain why.

(a) $V = \{0,1\}$

3 1 2 1 (q)
2 2 0 2
1 2 1 1
(p) 1 0 1 2

4-path: X

3 1 2 1 (q)
2 2 0 2
1 2 1 1
(p) 1 0 1 2

8-path: 4

3 1 2 1 (q)
2 2 0 2
1 2 1 1
(p) 1 0 1 2

m-path: 5

(b) Repeat (a) but using $V = \{1,2\}$.

(b) $V = \{1,2\}$

3 1 2 1 (q)
2 2 0 2
1 2 1 1
(p) 1 0 1 2

4-path: 6

3 1 2 1 (q)
2 2 0 2
1 2 1 1
(p) 1 0 1 2

8-path: 4

3 1 2 1 (q)
2 2 0 2
1 2 1 1
(p) 1 0 1 2

m-path: 6

2.37 We know from Eq. (2-45) that an affine transformation of coordinates is given by

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \mathbf{A}\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

where (x', y') are the transformed coordinates, (x, y) are the original coordinates, and the elements of $\mathbf{A}$ are given in Table 2.3 for various types of transformations. The inverse transformation, $\mathbf{A}^{-1}$, to go from the transformed back to the original coordinates is just as important for performing inverse mappings.

(a)* Find the inverse scaling transformation.

(b) Find the inverse translation transformation.

(c) Find the inverse vertical and horizontal shearing transformations.

(d)* Find the inverse rotation transformation.

(e)* Show a composite inverse translation/rotation transformation.

(e) $A^{-1} = T^{-1}R^{-1} = \begin{bmatrix} \cos\theta & \sin\theta & -t_x \\ -\sin\theta & \cos\theta & -t_y \\ 0 & 0 & 1 \end{bmatrix}$
(b)(d)

$\begin{bmatrix} + & - & + \\ - & + & - \\ + & - & + \end{bmatrix}$

(a) scaling: $A = \begin{bmatrix} C_x & 0 & 0 \\ 0 & C_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$, $|A| = C_x C_y$

$A^{-1} = \frac{1}{C_x C_y}\begin{bmatrix} C_y & 0 & 0 \\ 0 & C_x & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \frac{1}{C_x} & 0 & 0 \\ 0 & \frac{1}{C_y} & 0 \\ 0 & 0 & 1 \end{bmatrix}$

(b) translation: $A = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix}$

$A^{-1} = \begin{bmatrix} 1 & 0 & -t_x \\ 0 & 1 & -t_y \\ 0 & 0 & 1 \end{bmatrix}$

(c) $A = \begin{bmatrix} 1 & S_v & 0 \\ S_h & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$

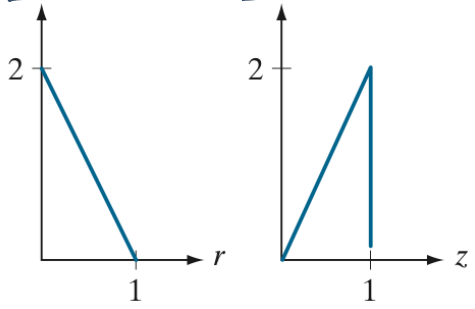
$A^{-1} = \frac{1}{1 - S_v S_h}\begin{bmatrix} 1 & -S_v & 0 \\ -S_h & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$

(d) rotation $A = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$

$A^{-1} = \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$

3.12 An image with intensities in the range [0,1] has the PDF, $p_r(r)$, shown in the following figure. It is desired to transform the intensity levels of this image so that they will have the specified $p_z(z)$ shown in the figure. Assume continuous quantities, and find the transformation (expressed in terms of r and z) that will accomplish this.

PDF $p_r(r) = 2 - 2r$ $\quad p_z(z) = 2z$



Histogram Eq. Transform

$s = T(r) = \int_0^r p_r(w)\,dw$

$= 2\int_0^r (1-w)\,dw$

$= 2\left(w - \frac{w^2}{2}\right)\Big|_0^r = 2r - r^2$

$s' = T(z) = \int_0^z p_z(w)\,dw$

$= \int_0^z 2w\,dw = w^2\Big|_0^z = z^2$

when $s = s'$,

$2r - r^2 = z^2$

$\Rightarrow z = \sqrt{2r - r^2}$

3.18 You are given the following kernel and image:

$$w = \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} \qquad f = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

(a) $w = \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$

$f' = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \end{bmatrix}$

(a)* Give a sketch of the area encircled by the large ellipse in Fig. 3.28 when the kernel is centered at point (2,3) (2nd row, 3rd col) of the image shown above. Show specific values of w and f.

(b)
with padding = 1.

(1) on point (2,3)

$w \cdot f = 6$ #

(b)* Compute the convolution $w \star f$ using the *minimum* zero padding needed. Show the details of your computations when the kernel is centered on point (2,3) of f; and then show the final full convolution result.

(2) full

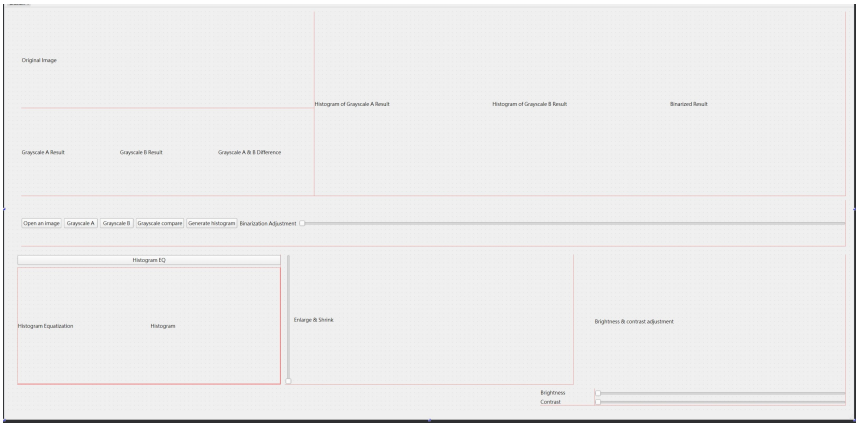
$$w * f = \begin{bmatrix} 0 & 1 & 2 & 1 & 0 \\ 0 & 3 & 6 & 3 & 0 \\ 0 & 4 & 8 & 4 & 0 \\ 0 & 3 & 6 & 3 & 0 \\ 0 & 1 & 2 & 1 & 0 \end{bmatrix}$$

(c) Repeat (b), but for correlation, $w \star f$.

(c) $w \star f = \begin{bmatrix} 0 & 1 & 2 & 1 & 0 \\ 0 & 3 & 6 & 3 & 0 \\ 0 & 4 & 8 & 4 & 0 \\ 0 & 3 & 6 & 3 & 0 \\ 0 & 1 & 2 & 1 & 0 \end{bmatrix}$ (the kernel is symmetric)

Part 2: (70%) Image File Reading, Display and Basic Processing

介面截圖



功能介紹：

1. Read a color BMP or JPEG image file and display it on the screen. You may use the functions provided by Qt, OpenCV, or MATLAB to read and display an image file. (10%)

使用 Qt 的 QFileDialog 模塊來打開圖像。如果用戶選擇了一個圖像，該函數將將圖像加載到 rgbImage 變量中。

```
void MainWindow::on_pushButton_clicked()
{
    QString filePath = QFileDialog::getOpenFileName(this, tr("Open Image"), "", tr("Images (*.bmp *.jpg *.jpeg *.png)"));

    QPixmap pixmap(filePath);
    ui->label->setPixmap(pixmap);
    ui->label->setScaledContents(true);
    ui->label->show();

    rgbImage.load(filePath);
}
```

2. Convert a color image into a grayscale image using the following equations:

   A. GRAY = (R+G+B)/3.0

   B. GRAY = 0.299*R + 0.587*G + 0.114*B

   Compare the grayscale images obtained from the above equations. One way to compare the difference between two images is by image subtraction (5%)

```cpp
void MainWindow::on_grayscale_Button_A_clicked()
{
   if (!rgbImage.isNull()) {
//       grayImage = rgbImage.convertToFormat(QImage::Format_Grayscale8);
////       displayImage(grayImage);
//       grayImage.setColorCount(256);
//       for(int i = 0; i < 256; i++)
//       {
//           grayImage.setColor(i, qRgb(i, i, i));
//       }
//       QPixmap pixmap1(QPixmap::fromImage (grayImage));

       unsigned char *grayData;                                  // 定義字符型指標數組grayData用於儲存灰度數據

       QPixmap pixmap(QPixmap::fromImage (rgbImage));
       unsigned char *data = rgbImage.bits ();                        // 第一個像素
       int width = rgbImage.width ();                                 // 寬度
       int height = rgbImage.height ();                               // 高度
       int bytePerLine = rgbImage.bytesPerLine();                     // 圖像每行字節數
       grayData = new unsigned char [bytePerLine * height];
       unsigned char red = 0;                                    // 紅色分量
       unsigned char green = 0;                                  // 綠色分量
       unsigned char blue = 0;                                   // 藍色分量
       for (int i = 0; i < height; i++)                          // 遍歷每一行
       {
           for ( int j = 0; j < width; j++ )                     // 遍歷每一列
           {
               red = *(data + 2);                                // 獲取目前像素紅色分量
               green = *(data + 1);                              // 獲取目前像素綠色分量
               blue = *(data);                                   // 獲取目前像素藍色分量
               grayData[i * bytePerLine + j * 3] = (red  + green  + blue ) / 3;
               grayData[i * bytePerLine + j * 3+1]=(red  + green  + blue ) / 3;
               grayData[i * bytePerLine + j * 3+2]=(red  + green  + blue ) / 3;
               data += 4;
           }
       }
       //        輸出灰階圖
       grayImage = QImage(grayData, width, height, bytePerLine, QImage::Format_RGB888);

       ui->label_2->setPixmap(QPixmap::fromImage(grayImage));
       ui->label_2->setScaledContents(true);
       ui->label_2->show();
   }
}
```

首先，使用一個外部循環遍歷彩色圖像的所有行。對於每行，它使用一個內部循環遍歷該行中的所有列。在每個像素處，獲取該像素的 RGB 分量，並將三個分量相加，得到該像素的灰度值。最後，它將灰度值存儲到灰度圖像的數據中。

```cpp
void MainWindow::on_grayscale_Button_B_clicked()
{
   if (!rgbImage.isNull()) {
//       grayImage = QImage(rgbImage.size(), QImage::Format_Grayscale8);
//       for (int y = 0; y < rgbImage.height(); y++) {
//           for (int x = 0; x < rgbImage.width(); x++) {
//               QRgb color = rgbImage.pixel(x, y);
//               int grayValue = qGray(color);
//               grayImage.setPixel(x, y, qRgb(grayValue, grayValue, grayValue));
//           }
//       }
       /* 灰度化 */

       unsigned char *grayData;                                  // 定義字符型指標數組grayData用於儲存灰度數據

       QPixmap pixmap(QPixmap::fromImage (rgbImage));
       unsigned char *data = rgbImage.bits ();                        // 第一個像素
       int width = rgbImage.width ();                                 // 寬度
       int height = rgbImage.height ();                               // 高度
       int bytePerLine = rgbImage.bytesPerLine();                     // 圖像每行字節數
       grayData = new unsigned char [bytePerLine * height];
       unsigned char red = 0;                                    // 紅色分量
       unsigned char green = 0;                                  // 綠色分量
       unsigned char blue = 0;                                   // 藍色分量
       for (int i = 0; i < height; i++)                          // 遍歷每一行
       {
           for ( int j = 0; j < width; j++ )                     // 遍歷每一列
           {
               red = *(data + 2);                                // 獲取目前像素紅色分量
               green = *(data + 1);                              // 獲取目前像素綠色分量
               blue = *(data);                                   // 獲取目前像素藍色分量
               grayData[i * bytePerLine + j * 3] = (red * 299 + green * 587 + blue * 114) / 1000;
               grayData[i * bytePerLine + j * 3+1]=(red * 299 + green * 587 + blue * 114) / 1000;
               grayData[i * bytePerLine + j * 3+2]=(red * 299 + green * 587 + blue * 114) / 1000;
               data += 4;
           }
       }
//        輸出灰階圖
       grayImage2 = QImage(grayData, width, height, bytePerLine, QImage::Format_RGB888);
       ui->label_3->setPixmap(QPixmap::fromImage(grayImage2));
       ui->label_3->setScaledContents(true);
       ui->label_3->show();
   }
}
```

grayscale_Button_B 不同之處在於它使用加權法來計算每個像素的灰度值，權重為 (299*Red + 587*Green + 114*Blue) / 1000

3. Determine and display the histogram of a grayscale image. (10%)

```cpp
void MainWindow::on_Hist_Button_clicked()
{
    if (!grayImage.isNull()) {
        // 使用Qt的繪圖功能繪製直方圖
        QVector<int> histogram(256, 0);
        // 計算每個像素的值
        for (int y = 0; y < grayImage.height(); y++) {
            for (int x = 0; x < grayImage.width(); x++) {
                QRgb color = grayImage.pixel(x, y);
                int grayValue = qGray(color);
                histogram[grayValue]++;
            }
        }

        // 找到直方圖中的最大值，以便歸一化
        int maxCount = 0;
        for (int i = 0; i < histogram.size(); i++) {
            if (histogram[i] > maxCount) {
                maxCount = histogram[i];
            }
        }

        QImage histogramImage(256, 150, QImage::Format_RGB32);
        histogramImage.fill(Qt::white);
        QPainter painter(&histogramImage);

        painter.setPen(Qt::blue);

       // 繪製直方圖
        for (int i = 0; i < histogram.size(); i++) {
            int height = histogram[i] * 150 / maxCount;
            painter.drawLine(i, 150, i, 150 - height);
        }

        ui->label_4->setPixmap(QPixmap::fromImage(histogramImage));
        ui->label_4->setScaledContents(true);
        ui->label_4->show();
    }
```

```cpp
    if (!grayImage2.isNull()) {
        // 使用Qt的繪圖功能繪製直方圖

        QVector<int> histogram(256, 0);

        // 計算每個像素的值
        for (int y = 0; y < grayImage2.height(); y++) {
            for (int x = 0; x < grayImage2.width(); x++) {
                QRgb color = grayImage2.pixel(x, y);
                int grayValue = qGray(color);
                histogram[grayValue]++;
            }
        }

        // 找到直方圖中的最大值，以便歸一化
        int maxCount = 0;
        for (int i = 0; i < histogram.size(); i++) {
            if (histogram[i] > maxCount) {
                maxCount = histogram[i];
            }
        }

        QImage histogramImage(256, 150, QImage::Format_RGB32);
        histogramImage.fill(Qt::white);
        QPainter painter(&histogramImage);

        painter.setPen(Qt::blue);

        // 繪製直方圖
        for (int i = 0; i < histogram.size(); i++) {
            int height = histogram[i] * 150 / maxCount;
            painter.drawLine(i, 150, i, 150 - height);
        }

        ui->label_5->setPixmap(QPixmap::fromImage(histogramImage));
        ui->label_5->setScaledContents(true);
        ui->label_5->show();
    }
}
```

計算 A、B 兩種算法灰度圖像的直方圖。直方圖是一個長度為 256 的數組，用於記錄每個灰度值的像素數量。

找到直方圖中的最大值，以便將直方圖歸一化。

創建一個新的圖像，用藍色線條繪製直方圖，並將其顯示在 UI 上。

```cpp
void MainWindow::on_gray_compare_clicked()
{
    if (!grayImage.isNull()&& !grayImage2.isNull()) {
////        grayImage2 = QImage(grayImage.size(), QImage::Format_Grayscale8);
//        for (int y = 0; y < grayImage.height(); y++) {
//            for (int x = 0; x < grayImage.width(); x++) {
//                QRgb color = grayImage.pixel(x, y);
//                int grayValue = qGray(color);
//                grayImage2.setPixel(x, y, qRgb(grayValue, grayValue, grayValue));
//            }
//        }
        QImage imageSubtraction = QImage(grayImage.size(), QImage::Format_Grayscale8);
        for (int y = 0; y < grayImage.height(); y++) {
            for (int x = 0; x < grayImage.width(); x++) {
                int grayValue = qGray(grayImage.pixel(x, y)) - qGray(grayImage2.pixel(x, y));
                imageSubtraction.setPixel(x, y, qRgb(grayValue, grayValue, grayValue));
            }
        }
        ui->label_6->setPixmap(QPixmap::fromImage(imageSubtraction));
        ui->label_6->setScaledContents(true);
        ui->label_6->show();
    }
}
```

on_gray_compare_clicked() 函數比較兩張灰度圖像。該函數首先將兩張圖像相減，然後將結果圖像顯示在螢幕上。

4. Implement a manual threshold function to convert a grayscale image into a binary image. (10%)

```cpp
void MainWindow::on_horizontalSlider_actionTriggered(int action)
{
    int threshold = ui->horizontalSlider->value();
//#Set the range of the slider to 0~255.
    ui->horizontalSlider->setRange(0, 255);
    if (!grayImage.isNull()) {
        QImage binaryImage(grayImage.size(), QImage::Format_Grayscale8);
        for (int y = 0; y < grayImage.height(); y++) {
            for (int x = 0; x < grayImage.width(); x++) {
                int grayValue = qGray(grayImage.pixel(x, y));
                if (grayValue > threshold) {
                    binaryImage.setPixel(x, y, qRgb(255, 255, 255));
                } else {
                    binaryImage.setPixel(x, y, qRgb(0, 0, 0));
                }
            }
        }
        ui->label_7->setPixmap(QPixmap::fromImage(binaryImage));
        ui->label_7->setScaledContents(true);
        ui->label_7->show();
    }
}
```

on_horizontalSlider_actionTriggered() 函數將灰度圖像轉換為二值圖像。該函數使用一個滑塊來控制閾值。如果灰度值大於閾值，則該像素將被設置為白色。否則，該像素將被設置為黑色。

5. Implement a function to adjust the spatial resolution (enlarge or shrink) and grayscale levels of an image. Use an interpolation method on enlarging an image. (10%)

```cpp
void MainWindow::on_verticalSlider_actionTriggered(int action)
{
    int zoomRate = ui->verticalSlider->value(); // Adjust the zoom rate using the vertical slider

    if (!grayImage.isNull()) {
        int R = grayImage.height();
        int r = 1; // Default to no change in resolution

        if (zoomRate != 0) {
            r = R / abs(zoomRate); // Adjust spatial resolution based on the slider, avoid division by zero
        }

        double n = static_cast<double>(R) / r;
        int c = static_cast<int>(grayImage.width() / n);

        QImage zoomedGrayImage(c, r, QImage::Format_Grayscale8);

        for (int i = 0; i < r; i++) {
            for (int j = 0; j < c; j++) {
                double x = (j + 0.5) * n; // Center point calculation
                double y = (i + 0.5) * n; // Center point calculation

                int zoomValue = 0; // Initialize the variable

                if (zoomRate > 0) {
                    // Enlarge: Bilinear interpolation
                    int x0 = static_cast<int>(x);
                    int y0 = static_cast<int>(y);
                    int x1 = std::min(x0 + 1, grayImage.width() - 1);
                    int y1 = std::min(y0 + 1, grayImage.height() - 1);
                    double dist_x = x - x0;
                    double dist_y = y - y0;

                    if (x0 >= 0 && y0 >= 0 && x1 >= 0 && y1 >= 0) {
                        int pixel00 = qGray(grayImage.pixel(x0, y0));
                        int pixel01 = qGray(grayImage.pixel(x1, y0));
                        int pixel10 = qGray(grayImage.pixel(x0, y1));
                        int pixel11 = qGray(grayImage.pixel(x1, y1));

                        zoomValue = static_cast<int>((pixel00 * (1 - dist_x) * (1 - dist_y) +
                                                      pixel01 * dist_x * (1 - dist_y) +
                                                      pixel10 * (1 - dist_x) * dist_y +
                                                      pixel11 * dist_x * dist_y));
                    }
                } else {
                    // Shrink: Simple downsampling
                    int x0 = static_cast<int>(x);
                    int y0 = static_cast<int>(y);

                    if (x0 >= 0 && y0 >= 0) {
                        zoomValue = qGray(grayImage.pixel(x0, y0));
                    }
                }

                // Clip the values to the 0-255 range
                zoomValue = std::min(255, std::max(0, zoomValue));

                zoomedGrayImage.setPixel(j, i, qRgb(zoomValue, zoomValue, zoomValue));
            }
        }
        ui->label_8->setPixmap(QPixmap::fromImage(zoomedGrayImage));
        ui->label_8->setScaledContents(true);
        ui->label_8->show();

    }
}
```

on_verticalSlider_actionTriggered()函數對灰度圖像進行縮放。該函數將縮放率作為輸入，該值介於 -1 和 1 之間。縮放率為 1 表示不應縮放圖像。縮放率大於 1 表示應放大圖像，而縮放率小於 1 表示應縮小圖像。

該函數首先計算縮放圖像的新高和寬。新高是透過將原始高除以縮放率來計算

的。新寬是通過將原始寬除以縮放率來計算的。

然後，該函數創建一個新的 QImage 對象來儲存縮放圖像。新圖像使用計算出來的高度、寬度和格式創建。

接下來，該函數遍歷縮放圖像的像素。對於每個像素，該函數計算原始圖像中相應的坐標。然後，該函數獲取原始圖像中該坐標處像素的灰度值。縮放圖像中像素的新灰度值設置為原始圖像中相應像素的灰度值。

最後，在標籤中顯示縮放圖像。

6. Implement a function to adjust the brightness and contrast of an image. (10%)

```
void MainWindow::on_horizontalSlider_2_actionTriggered(int action)
{
//#Set the range of the slider to -100~100.
   ui->horizontalSlider_2->setRange(-100, 100);
   ui->horizontalSlider_3->setRange(100, 1000);
   if (!grayImage.isNull()) {
       int brightness = ui->horizontalSlider_2->value();
       int contrast = ui->horizontalSlider_3->value();
       QImage brightnessImage(grayImage.size(), QImage::Format_Grayscale8);
       for (int y = 0; y < grayImage.height(); y++) {
           for (int x = 0; x < grayImage.width(); x++) {
               int grayValue = qGray(grayImage.pixel(x, y));
               grayValue = grayValue * contrast / 100 + brightness;
//               grayValue = grayValue + brightness;
               if (grayValue < 0) {
                   grayValue = 0;
               } else if (grayValue > 255) {
                   grayValue = 255;
               }
               brightnessImage.setPixel(x, y, qRgb(grayValue, grayValue, grayValue));
           }
       }
       ui->label_9->setPixmap(QPixmap::fromImage(brightnessImage));
       ui->label_9->setScaledContents(true);
       ui->label_9->show();
   }
}
```

on_horizontalSlider_2_actionTriggered() 函數用於調整圖像的亮度。滑塊的值表示亮度的增量或減量。例如，如果滑塊值為 100，則圖像的亮度將增加 100。如果滑塊值為 -100，則圖像的亮度將減少 100。

該函數首先獲取滑塊的值。然後，該函數遍歷圖像的像素。對於每個像素，該函數將像素的灰度值增加或減少滑塊的值。

```
void MainWindow::on_horizontalSlider_3_actionTriggered(int action)
{
    ui->horizontalSlider_2->setRange(-100, 100);
    ui->horizontalSlider_3->setRange(100, 1000);
    if (!grayImage.isNull()) {
        int brightness = ui->horizontalSlider_2->value();
        int contrast = ui->horizontalSlider_3->value();
        QImage brightnessImage(grayImage.size(), QImage::Format_Grayscale8);
        for (int y = 0; y < grayImage.height(); y++) {
            for (int x = 0; x < grayImage.width(); x++) {
                int grayValue = qGray(grayImage.pixel(x, y));
                grayValue = grayValue * contrast / 100 + brightness;
//                grayValue = grayValue + brightness;
                if (grayValue < 0) {
                    grayValue = 0;
                } else if (grayValue > 255) {
                    grayValue = 255;
                }
                brightnessImage.setPixel(x, y, qRgb(grayValue, grayValue, grayValue));
            }
        }
        ui->label_9->setPixmap(QPixmap::fromImage(brightnessImage));
        ui->label_9->setScaledContents(true);
        ui->label_9->show();
    }
}
```

on_horizontalSlider_3_actionTriggered() 函數用於調整圖像的對比度。滑塊的值表示對比度的增量或減量。例如，如果滑塊值為 100，則圖像的對比度將增加 100%。如果滑塊值為 -100，則圖像的對比度將減少 100%。

該函數首先獲取滑塊的值。然後，該函數遍歷圖像的像素。對於每個像素，該函數將像素的灰度值乘以滑塊的值。

7. Implement a histogram equalization function for automatic contrast adjustment. (15%)

```cpp
void MainWindow::on_verticalSlider_actionTriggered(int action)
{
    int zoomRate = ui->verticalSlider->value(); // Adjust the zoom rate using the vertical slider

    if (!grayImage.isNull()) {
        int R = grayImage.height();
        int r = 1; // Default to no change in resolution

        if (zoomRate != 0) {
            r = R / abs(zoomRate); // Adjust spatial resolution based on the slider, avoid division by zero
        }

        double n = static_cast<double>(R) / r;
        int c = static_cast<int>(grayImage.width() / n);

        QImage zoomedGrayImage(c, r, QImage::Format_Grayscale8);

        for (int i = 0; i < r; i++) {
            for (int j = 0; j < c; j++) {
                double x = (j + 0.5) * n; // Center point calculation
                double y = (i + 0.5) * n; // Center point calculation

                int zoomValue = 0; // Initialize the variable

                if (zoomRate > 0) {
                    // Enlarge: Bilinear interpolation
                    int x0 = static_cast<int>(x);
                    int y0 = static_cast<int>(y);
                    int x1 = std::min(x0 + 1, grayImage.width() - 1);
                    int y1 = std::min(y0 + 1, grayImage.height() - 1);
                    double dist_x = x - x0;
                    double dist_y = y - y0;

                    if (x0 >= 0 && y0 >= 0 && x1 >= 0 && y1 >= 0) {
                        int pixel00 = qGray(grayImage.pixel(x0, y0));
                        int pixel01 = qGray(grayImage.pixel(x1, y0));
                        int pixel10 = qGray(grayImage.pixel(x0, y1));
                        int pixel11 = qGray(grayImage.pixel(x1, y1));

                        zoomValue = static_cast<int>((pixel00 * (1 - dist_x) * (1 - dist_y) +
                                                      pixel01 * dist_x * (1 - dist_y) +
                                                      pixel10 * (1 - dist_x) * dist_y +
                                                      pixel11 * dist_x * dist_y));
                    }
                } else {
                    // Shrink: Simple downsampling
                    int x0 = static_cast<int>(x);
                    int y0 = static_cast<int>(y);

                    if (x0 >= 0 && y0 >= 0) {
                        zoomValue = qGray(grayImage.pixel(x0, y0));
                    }
                }

                // Clip the values to the 0-255 range
                zoomValue = std::min(255, std::max(0, zoomValue));

                zoomedGrayImage.setPixel(j, i, qRgb(zoomValue, zoomValue, zoomValue));
            }
        }
        ui->label_8->setPixmap(QPixmap::fromImage(zoomedGrayImage));
        ui->label_8->setScaledContents(true);
        ui->label_8->show();

    }
}
```

on_pushButton_2_clicked()函數對灰度圖像進行直方圖均衡。直方圖均衡是一種通過重新分佈像素值以使它們更加均勻分佈來提高圖像對比度的技術。

該函數首先計算灰度圖像的直方圖。直方圖是一個整數向量，其中每個元素表示圖像中具有特定灰度值像素的數量。

然後，該函數對直方圖進行累積求和。這意味着新向量的每個元素都包含原始直方圖中所有元素（包括該元素在內）的和。

最後，該函數計算均衡圖像。對於原始圖像中的每個像素，該函數在直方圖中找到相應的灰度值。然後，像素的新灰度值是通過將累積直方圖值乘以 255 並除以圖像中的總像素數來計算的。