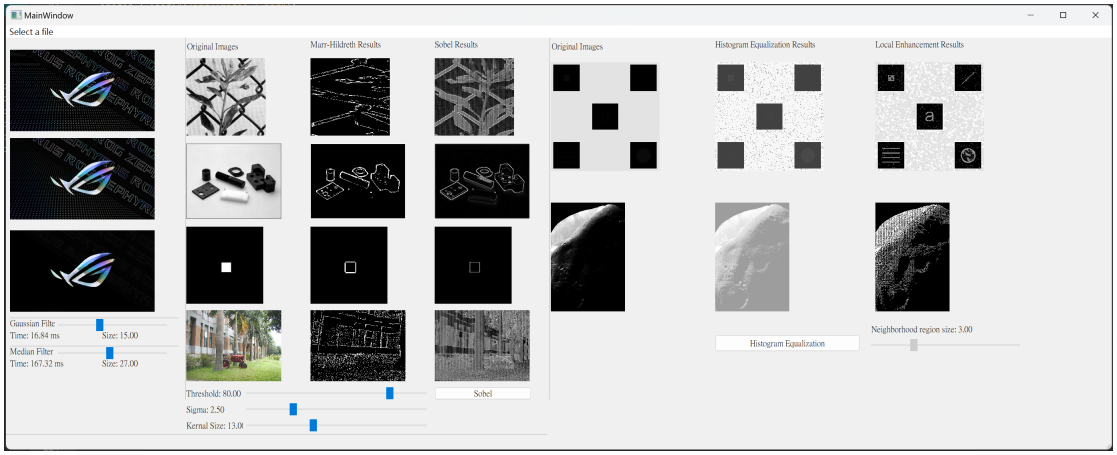
B09611007 陳柏霖
介面截圖：

(請點擊左上 ”Select a file” 以選取欲測試照片)



功能介紹：

Part 2: (25%)

Design a computer program for spatial filtering operations using various types of masks. Test your program with several images and report your results. Discuss the effect of mask size on the processed images and the computation time. You should design a mask operation function that is flexible for adjusting mask size and setting coefficients in the mask.

ShowTime 函數用於計算代碼片段的執行時間並在標籤上顯示它。

```
void MainWindow::ShowTime(QLabel* label, const std::function<void()>& function)
{
    QElapsedTimer timer;
    timer.start();

    // Execute the provided function
    function();

    qint64 elapsed = timer.nsecsElapsed();
    double milliseconds = static_cast<double>(elapsed) / 1000000.0;

    // Display the time on the label
    label->setText(QString("Time: %1 ms").arg(milliseconds, 0, 'f', 2));
}
```

QImage2CvMat 函數用於將 QImage 轉換為 OpenCV 的 Mat 對象，以便進行圖像處理。

```cpp
cv::Mat MainWindow::QImage2CvMat(QImage &image)
{
    cv::Mat mat;
    //qDebug() << image.format();
    switch (image.format())
    {
    case QImage::Format_ARGB32:
        mat = cv::Mat(image.height(), image.width(), CV_8UC4, (void*)image.constBits(), image.bytesPerLine());
        break;
    case QImage::Format_RGB32:
        mat = cv::Mat(image.height(), image.width(), CV_8UC3, (void*)image.constBits(), image.bytesPerLine());
        //cv::cvtColor(mat, mat, CV_BGR2RGB);
        break;
    case QImage::Format_ARGB32_Premultiplied:
        mat = cv::Mat(image.height(), image.width(), CV_8UC4, (void*)image.constBits(), image.bytesPerLine());
        break;
    case QImage::Format_RGB888:
        mat = cv::Mat(image.height(), image.width(), CV_8UC3, (void*)image.constBits(), image.bytesPerLine());
        //cv::cvtColor(mat, mat, CV_BGR2RGB);
        break;
    case QImage::Format_Indexed8:
        mat = cv::Mat(image.height(), image.width(), CV_8UC1, (void*)image.constBits(), image.bytesPerLine());
        break;
    case QImage::Format_Grayscale8:
        mat = cv::Mat(image.height(), image.width(), CV_8UC1, (void*)image.constBits(), image.bytesPerLine());
        break;
    }
    return mat;
}
```

displayImageOnLabel 函數將 QImage 顯示在 QLabel 上，用於在應用程序中顯示圖像。

```cpp
void MainWindow::displayImageOnLabel(const QImage& image, QLabel* label)
{
//    label->setPixmap(QPixmap::fromImage(image));
//    label->setScaledContents(true);
//    label->show();
    const int w = label ->width();
    const int h = label ->height();
    label ->setPixmap(QPixmap::fromImage(image.scaled(w, h, Qt::KeepAspectRatio)));
}
```

cvMat_to_QImage 函數用於將 OpenCV 的 Mat 對象轉換為 QImage，以便進行圖像處理。

```cpp
QImage MainWindow::cvMat_to_QImage(const cv::Mat &mat ) {
    switch ( mat.type() )
    {
    // 8-bit, 4 channel
    case CV_8UC4:
    {
        QImage image( mat.data, mat.cols, mat.rows, mat.step, QImage::Format_RGB32 );
        return image;
    }

        // 8-bit, 3 channel
    case CV_8UC3:
    {
        QImage image( mat.data, mat.cols, mat.rows, mat.step, QImage::Format_RGB888 );
        return image.rgbSwapped();
    }

        // 8-bit, 1 channel
    case CV_8UC1:
    {
        static QVector<QRgb>  sColorTable;
        // only create our color table once
        if ( sColorTable.isEmpty() )
        {
            for ( int i = 0; i < 256; ++i )
                sColorTable.push_back( qRgb( i, i, i ) );
        }
        QImage image( mat.data, mat.cols, mat.rows, mat.step, QImage::Format_Indexed8 );
        image.setColorTable( sColorTable );
        return image;
    }

    default:
        qDebug("Image format is not supported: depth=%d and %d channels\n", mat.depth(), mat.channels());
        break;
    }
    return QImage();
}
```

使左上按鈕能用於開啟圖片

```
void MainWindow::on_actionOpen_an_image_triggered()
{
    filePath = QFileDialog::getOpenFileName(this, tr("Open Image"), "", tr("Images (*.bmp *.jpg *.jpeg *.png)"));
//    qDebug()<<filePath;
    QPixmap pixmap(filePath);
//    ui->label->setPixmap(pixmap);
//    ui->label->setScaledContents(true);
//    ui->label->show();

    rgbImage.load(filePath);
    displayImageOnLabel(rgbImage, ui->label);
    filename = filePath.toStdString();
//    img_new = imread(filename);
//    imshow("", img_new);
    img_new = QImageToCvMat(rgbImage, true);
//    img_new = QImage2CvMat(rgbImage);
}
```

on_horizontalSlider_G_actionTriggered() 函數的功能是根據水平滑塊 G 的值對圖像進行高斯模糊處理，並將處理後的圖像顯示在標籤 Image 上。
該函數首先設置水平滑塊 G 的最小值為 1，最大值為 41，步長為 2。然後，根據水平滑塊 G 的值設置高斯模糊的核大小 kernel。如果 kernel 是偶數，則將其減 1，使其成為奇數。這是因為高斯模糊核的大小必須是奇數，才能保證處理後的圖像不會出現偏差。
接下來，該函數調用 cv::GaussianBlur() 函數對圖像進行高斯模糊處理。
cv::GaussianBlur() 函數的參數分別是：
img_new：原始圖像
img_new_output：處理後的圖像
Size(kernel, kernel)：高斯模糊核的大小
0：高斯模糊的 X 軸標準差
0：高斯模糊的 Y 軸標準差
最後，該函數將處理後的圖像 img_new_output 轉換為 QImage 格式，並將其顯示在標籤上。

```cpp
void MainWindow::on_horizontalSlider_G_actionTriggered(int action)
{

//    qDebug()<<filename;
//    qDebug()<<filePath.toStdString();
//    Mat img_new = imread(filePath.toStdString());
    ShowTime(ui->label_Time, [&]() {

        ui->horizontalSlider_G->setMinimum(1);
        ui->horizontalSlider_G->setMaximum(41);
        ui->horizontalSlider_G->setSingleStep(2);
        kernel = ui->horizontalSlider_G->value();

        if(kernel%2==0)
            kernel=kernel-1;
        else
            kernel=kernel;

        // Display the size on the label
        ui->label_Size->setText(QString("Size: %1 ").arg(kernel , 0, 'f', 2));

        Mat img_new_output;

        if (!img_new.empty()) {
            cv::GaussianBlur(img_new, img_new_output, Size(kernel, kernel), 0, 0);

            // Check if the cvMat_to_QImage conversion is successful
            QImage qimg = cvMat_to_QImage(img_new_output);
            if (!qimg.isNull()) {
                const int w = ui->label_Image->width();
                const int h = ui->label_Image->height();
                ui->label_Image->setPixmap(QPixmap::fromImage(qimg.scaled(w, h, Qt::KeepAspectRatio)));
            } else {
                qDebug("cvMat_to_QImage conversion failed.");
            }
        } else {
            qDebug("Image not loaded or empty.");
        }
    });
}
```

applyMedianFilter() 函數的功能是根據指定的核大小 kernelSize 對圖像進行中值濾波處理，並將處理後的圖像顯示在標籤 M 上。

該函數首先檢查圖像 img_new 是否為空。如果為空，則輸出錯誤信息並退出。

接下來，該函數檢查 kernelSize 是否為偶數。如果是，則將 kernelSize 加 1，使其成為奇數。這是因為中值濾波核的大小必須是奇數，才能保證處理後的圖像不會出現偏差。

然後，該函數調用 cv::medianBlur() 函數對圖像 img_new 進行中值濾波處理。

cv::medianBlur() 函數的參數分別是：

img_new：原始圖像

img_new_output：處理後的圖像

kernelSize：中值濾波核的大小

最後，該函數將處理後的圖像 img_new_output 轉換為 QImage 格式，並將其顯示在標籤上。

```cpp
void MainWindow::applyMedianFilter(int kernelSize)
{
    if (!img_new.empty()) {
        if (kernelSize % 2 == 0) {
            kernelSize += 1; // Ensure that kernelSize is odd
        }
        Mat img_new_output;

        cv::medianBlur(img_new, img_new_output, kernelSize);
        // Display the time on the label
        ui->label_Size2->setText(QString("Size: %1 ").arg(kernelSize, 0, 'f', 2));

        // Check if the cvMat_to_QImage conversion is successful
        QImage qimg = cvMat_to_QImage(img_new_output);
        if (!qimg.isNull()) {
//            const int w = ui->label_M->width();
//            const int h = ui->label_M->height();
//            ui->label_M->setPixmap(QPixmap::fromImage(qimg.scaled(w, h, Qt::KeepAspectRatio)));
            displayImageOnLabel(qimg, ui->label_M);
        } else {
            qDebug("cvMat_to_QImage conversion failed.");
        }
    } else {
        qDebug("Image not loaded or empty.");
    }
}
```

Part 3: (25%)

The Marr-Hildreth edge detection method operates by convolving the image with the Laplacian of Gaussian operators. Laplacian of Gaussian (LoG) is a second derivative of a Gaussian filter. The LoG can be broken up into two steps. First, smooth the image with a Gaussian filter, and second, convolve the image with a Laplacian mask. Read the Section 10.2 of our textbook for a detailed theory and procedure of this edge detection method. Implement a computer program for edge detection using the Marr-Hildreth edge detector. Test your program with at least 4 images and compare the results with those processed with the Sobel operator. Sample images are provided in the NTU COOL course website. To locate the edges of the images processed with the Marr-Hildreth detector, you will need to further process the images with the zero-crossing detector as described in the textbook. Discuss the effect of zero-crossing threshold on the Marr-Hildreth edge detection method.

MarrHildrethEdgeDetection() 函數的功能是根據 Marr-Hildreth 邊緣檢測算法對圖像進行邊緣檢測，並返回處理後的圖像。
該函數的參數分別是：
inputImage：原始圖像
sigma：高斯模糊的標準差
zeroCrossingThreshold：零交叉檢測器的閾值
kernelSize：高斯模糊和 Laplacian 算子的核大小
該函數的返回值是處理後的圖像，邊緣像素值為 255，非邊緣像素值為 0。

```cpp
Mat MainWindow::MarrHildrethEdgeDetection(const Mat& inputImage, double sigma, double zeroCrossingThreshold, int kernelSize) {
    Mat outputImage;
    // Step 1: Gaussian smoothing
    Mat blurredImage;
    qDebug()<<kernelSize;
    GaussianBlur(inputImage, blurredImage, Size(kernelSize, kernelSize), 0, 0);

    // Step 2: Compute Laplacian (LoG) operator with the specified kernel size
    Mat logImage;
    Laplacian(blurredImage, logImage, CV_16S, 5);

    // Convert the Laplacian image to 8-bit for visualization
    Mat logImage8U;
    convertScaleAbs(logImage, logImage8U);

    // Step 3: Apply zero-crossing detector
    Mat edgeImage = Mat::zeros(logImage.size(), CV_8U);
    for (int y = 1; y < logImage.rows - 1; y++) {
        for (int x = 1; x < logImage.cols - 1; x++) {
            short pixelValue = logImage.at<short>(y, x);
            if (pixelValue > 0) {
                bool zeroCrossingDetected = false;
                if (abs(pixelValue - logImage.at<short>(y + 1, x)) > zeroCrossingThreshold ||
                    abs(pixelValue - logImage.at<short>(y - 1, x)) > zeroCrossingThreshold ||
                    abs(pixelValue - logImage.at<short>(y, x + 1)) > zeroCrossingThreshold ||
                    abs(pixelValue - logImage.at<short>(y, x - 1)) > zeroCrossingThreshold) {
                    zeroCrossingDetected = true;
                }
                if (zeroCrossingDetected) {
                    edgeImage.at<uchar>(y, x) = 255;
                }
            }
        }
    }

    // Step 4: Convert to 8-bit image
    edgeImage.convertTo(outputImage, CV_8U);
    return outputImage;
}
```

Sobel_process() 函數的功能是對圖像進行 Sobel 邊緣檢測，並返回處理後的圖像。

該函數的參數是原始圖像 img_gray。該函數的返回值是處理後的圖像 grad，其中邊緣像素值較高，非邊緣像素值較低。

```cpp
Mat MainWindow::Sobel_process(const Mat& img_gray){
//    qDebug()<<"First";
    Mat grad_x1, grad_y1;
//    qDebug()<<"2nd";
    Sobel(img_gray, grad_x1, CV_16S, 1, 0, 3);
    Sobel(img_gray, grad_y1, CV_16S, 0, 1, 3);
//    qDebug()<<"Sobel";
    Mat abs_grad_x1, abs_grad_y1;
    convertScaleAbs(grad_x1, abs_grad_x1);
    convertScaleAbs(grad_y1, abs_grad_y1);
//    qDebug()<<"abs";
    Mat grad;
    addWeighted(abs_grad_x1, 0.5, abs_grad_y1, 0.5, 0, grad);
//    qDebug()<<"grad";
    return grad;
}
```

Part 4: (25%)

Implement a computer program following the local enhancement method described in Example 3.10 of the textbook. You need to complete the following task with your program

Histogram_Eq() 函數的功能是對圖像進行直方圖均衡化，並返回處理後的圖像。

該函數的參數是原始圖像 grayImage。該函數的返回值是處理後的圖像 equalizedImage。

```cpp
QImage MainWindow::Histogram_Eq(QImage& grayImage){
    // Calculate the histogram of the grayscale image
    QVector<int> histogram(256, 0);
    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]++;
        }
    }

    // Perform histogram equalization
    int totalPixels = grayImage.width() * grayImage.height();
    QVector<int> cumulativeHistogram(256, 0);
    cumulativeHistogram[0] = histogram[0];
    for (int i = 1; i < 256; i++) {
        cumulativeHistogram[i] = cumulativeHistogram[i - 1] + histogram[i];
    }

    QImage equalizedImage(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);
            int newGrayValue = (cumulativeHistogram[grayValue] * 255) / totalPixels;
            equalizedImage.setPixel(x, y, qRgb(newGrayValue, newGrayValue, newGrayValue));
        }
    }

    return equalizedImage;
}
```

LocalEnhancement() 函數的功能是對圖像進行局部增強，並返回處理後的圖像。

該函數的參數是原始圖像 originalImage 和局部增強窗口的大小 Sxy。該函數的返回值是處理後的圖像 processedImage。

局部增強後的圖像中的每個像素值都等於該像素的局部均值加上一個由局部標準差決定的常數。局部標準差越大，則常數越大，圖像的對比度也就越高。

局部增強窗口的大小 Sxy 控制了局部均值和局部標準差的計算範圍。Sxy 越大，則局部均值和局部標準差的計算範圍越大，圖像的對比度也就越高。

```cpp
QImage MainWindow::LocalEnhancement(QImage& originalImage, int Sxy) {
    QImage processedImage(originalImage.size(), originalImage.format());

    for (int x = 0; x < originalImage.width(); x++) {
        for (int y = 0; y < originalImage.height(); y++) {
            QRgb pixel = originalImage.pixel(x, y);

            int redSum = 0, greenSum = 0, blueSum = 0;
            int count = 0;

            for (int i = -Sxy; i <= Sxy; i++) {
                for (int j = -Sxy; j <= Sxy; j++) {
                    int newX = x + i;
                    int newY = y + j;

                    if (newX >= 0 && newX < originalImage.width() && newY >= 0 && newY < originalImage.height()) {
                        QRgb neighborPixel = originalImage.pixel(newX, newY);
                        redSum += qRed(neighborPixel);
                        greenSum += qGreen(neighborPixel);
                        blueSum += qBlue(neighborPixel);
                        count++;
                    }
                }
            }

            // Calculate local mean and local standard deviation
            int localMean = (redSum + greenSum + blueSum) / (3 * count);

            int redDev = qRed(pixel) - localMean;
            int greenDev = qGreen(pixel) - localMean;
            int blueDev = qBlue(pixel) - localMean;

            int redEnhanced = localMean + static_cast<int>((redDev * 255) / (sqrt(redDev * redDev + greenDev * greenDev + blueDev * blueDev) + 0.1));
            int greenEnhanced = localMean + static_cast<int>((greenDev * 255) / (sqrt(redDev * redDev + greenDev * greenDev + blueDev * blueDev) + 0.1));
            int blueEnhanced = localMean + static_cast<int>((blueDev * 255) / (sqrt(redDev * redDev + greenDev * greenDev + blueDev * blueDev) + 0.1));

            // Ensure the values are within the [0, 255] range
            redEnhanced = std::min(255, std::max(0, redEnhanced));
            greenEnhanced = std::min(255, std::max(0, greenEnhanced));
            blueEnhanced = std::min(255, std::max(0, blueEnhanced));

            processedImage.setPixel(x, y, qRgb(redEnhanced, greenEnhanced, blueEnhanced));
        }
    }

    return processedImage;
}
```