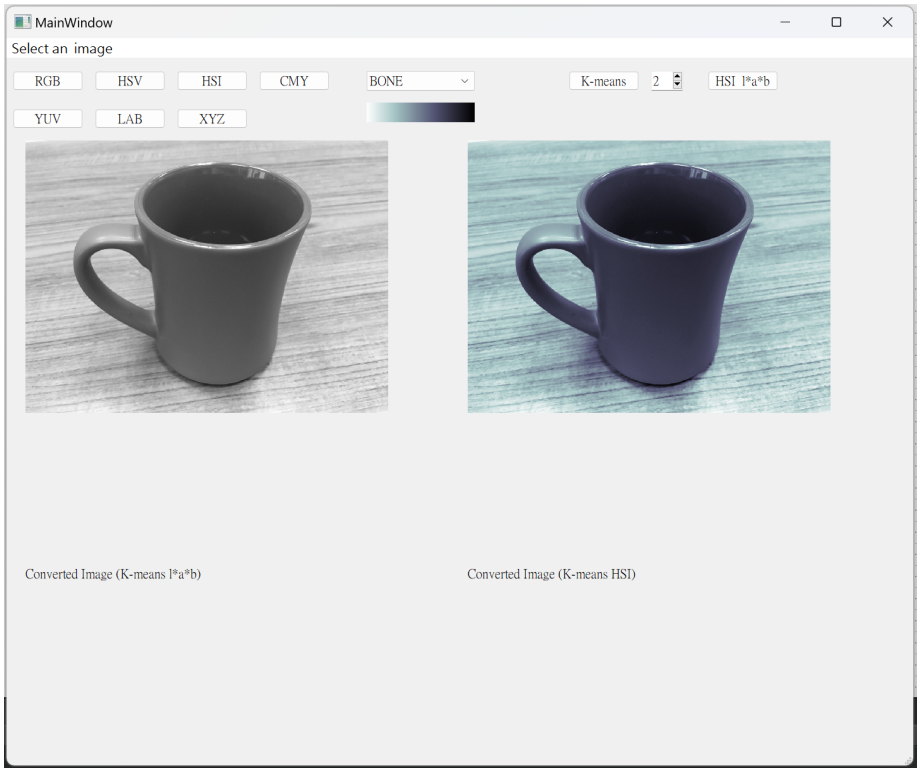
B09611007 陳柏霖

介面截圖

(請點擊 Select an image 以選取欲測試照片)

調整後的圖片會顯示在右側

HIS 與 Lab 的 Kmeans 結果則會顯示在下方



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

```
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。

```
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。

```
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();
}
```

PART 1: 以下是將 RGB 轉換為 CMY、HIS、XYZ、Lab、YUV 的函數

```
// Function to convert RGB to CMY
QImage MainWindow::RGBtoCMY(const QImage &inputImage) {
    // Convert QImage to cv::Mat
    cv::Mat rgbMat = QImageToCvMat(inputImage);

    // Your conversion logic here
    cv::Mat img_cmy = cv::Scalar(255, 255, 255) - rgbMat;

    // Convert cv::Mat to QImage
    QImage outputImage = cvMatToQImage(img_cmy);
    return outputImage;
}
```

```cpp
Mat RGB2HSI(const Mat & rgb){
    Mat hsi(rgb.rows, rgb.cols, rgb.type());
    float  H=0, S=0, I=0;
    for(int i=0; i < rgb.rows; i++)
        for(int j=0; j < rgb.cols; j++){
            float B = rgb.at<Vec3b>(i, j)[0] / 255.f,
                G = rgb.at<Vec3b>(i, j)[1] / 255.f,
                R = rgb.at<Vec3b>(i, j)[2] / 255.f;

            float num = (R - G + R - B) / 2,
                den = sqrt((R - G) * (R - G) + (R - B) * (G - B)),
                theta = acos(num/den);
            if(den == 0) H = 0; // 分母不能为0
            else H = B <= G ? theta / (2 * MY_PI) : 1 - theta / (2 * MY_PI);

            float sum = B + G + R;
            if(sum == 0) S = 0;
            else S = 1 - 3 * min(min(B, G), R) / sum;

            I = sum/3.0;

            hsi.at<Vec3b>(i, j)[0] = H*255;
            hsi.at<Vec3b>(i, j)[1] = S*255;
            hsi.at<Vec3b>(i, j)[2] = I*255;
        }
    return hsi;
}
```

```cpp
// Function to convert RGB to XYZ
QImage MainWindow::RGBtoXYZ(const QImage &inputImage) {

    int width = inputImage.width();
    int height = inputImage.height();

    QImage outputImage(width, height, QImage::Format_RGB32);

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

            double r = qRed(pixel) / 255.0;
            double g = qGreen(pixel) / 255.0;
            double b = qBlue(pixel) / 255.0;

            // Apply sRGB to XYZ conversion matrix
            double xValue = 0.4124564 * r + 0.3575761 * g + 0.1804375 * b;
            double yValue = 0.2126729 * r + 0.7151522 * g + 0.0721750 * b;
            double zValue = 0.0193339 * r + 0.1191920 * g + 0.9503041 * b;

            // Clip values to the valid range
            xValue = qBound(0.0, xValue, 1.0);
            yValue = qBound(0.0, yValue, 1.0);
            zValue = qBound(0.0, zValue, 1.0);

            // Convert back to RGB values
            int rOut = static_cast<int>(xValue * 255.0);
            int gOut = static_cast<int>(yValue * 255.0);
            int bOut = static_cast<int>(zValue * 255.0);

            // Set RGB values in the output image
            outputImage.setPixel(x, y, qRgb(rOut, gOut, bOut));
        }
    }

    return outputImage;
}
```

```cpp
QImage MainWindow::RGBtoLab(const QImage &inputImage){
    int width = inputImage.width();
    int height = inputImage.height();

    QImage outputImage(width, height, QImage::Format_RGB32);

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

            double r = qRed(pixel) / 255.0;
            double g = qGreen(pixel) / 255.0;
            double b = qBlue(pixel) / 255.0;

            // Apply sRGB to XYZ conversion matrix
            double xValue = 0.4124564 * r + 0.3575761 * g + 0.1804375 * b;
            double yValue = 0.2126729 * r + 0.7151522 * g + 0.0721750 * b;
            double zValue = 0.0193339 * r + 0.1191920 * g + 0.9503041 * b;

            // Normalize XYZ values to D65 illuminant
            xValue /= 0.950456;
            zValue /= 1.088754;

            // Apply XYZ to Lab conversion
            xValue = (xValue > 0.008856) ? std::cbrt(xValue) : ((903.3 * xValue) + 16.0) / 116.0;
            yValue = (yValue > 0.008856) ? std::cbrt(yValue) : ((903.3 * yValue) + 16.0) / 116.0;
            zValue = (zValue > 0.008856) ? std::cbrt(zValue) : ((903.3 * zValue) + 16.0) / 116.0;

            double lValue = (116.0 * yValue) - 16.0;
            double aValue = (xValue - yValue) * 500.0;
            double bValue = (yValue - zValue) * 200.0;

            // Clip values to the valid range
            lValue = qBound(0.0, lValue, 100.0);
            aValue = qBound(-128.0, aValue, 127.0);
            bValue = qBound(-128.0, bValue, 127.0);

            // Convert back to RGB values
            int rOut = static_cast<int>(lValue * 2.55);
            int gOut = static_cast<int>((aValue + 128.0) * 0.7874);
            int bOut = static_cast<int>((bValue + 128.0) * 0.7874);

            // Set RGB values in the output image
            outputImage.setPixel(x, y, qRgb(rOut, gOut, bOut));
        }
    }

    return outputImage;
}
```

```cpp
QImage MainWindow::RGBtoYUV(const QImage &inputImage){
//    Mat RGB_img = QImageToCvMat(inputImage);
//    Mat YUV_img;
//    Mat_RGBToYUV_NV12(RGB_img, YUV_img);
//    return cvMatToQImage(YUV_img);
    int width = inputImage.width();
    int height = inputImage.height();

    QImage outputImage(width, height, QImage::Format_RGB32);

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

            int r = qRed(pixel);
            int g = qGreen(pixel);
            int b = qBlue(pixel);

            // RGB to YUV conversion
            int yValue = 0.299 * r + 0.587 * g + 0.114 * b;
            int uValue = 0.492 * (b - yValue);
            int vValue = 0.877 * (r - yValue);

            // Clip values to the valid range
            yValue = qBound(0, yValue, 255);
            uValue = qBound(0, uValue + 128, 255);
            vValue = qBound(0, vValue + 128, 255);

            // Set RGB values in the output image
            outputImage.setPixel(x, y, qRgb(yValue, uValue, vValue));
        }
    }

    return outputImage;
}
```

PART2: 利用 Combobox 來給使用者選取想要的濾鏡效果

```
void MainWindow::on_comboBox_activated(int index)
{
    // Assuming 'id' is a member variable representing the selected pseudo-color map
    id = index;

    cv::Mat original_img = QImageToCvMat(rgbImage);
    // Apply pseudo-color mapping here
    if (original_img.empty()) {
        QMessageBox::warning(this, "WARNING", "The input image is empty");
        return;
    }

    cv::Mat img_gray;
    cv::cvtColor(original_img, img_gray, cv::COLOR_BGR2GRAY);
    showGrayImage(img_gray);

    cv::Mat img_nor;
    cv::normalize(img_gray, img_nor, 0, 255, cv::NORM_MINMAX);

    cv::Mat img_output;
    cv::applyColorMap(img_nor, img_output, id);

    QImage qimage = cvMatToQImage(img_output);

    // Assuming ui->output is the QLabel where you want to display the processed image
    displayImageOnLabel(qimage, ui->Converted_Img_label);

    // Update the colormap display
    cv::Mat gradientImage = cv::Mat(256, 1, CV_8UC1);
    for (int i = 0; i < 256; ++i) {
        gradientImage.at<uchar>(i, 0) = static_cast<uchar>(i);
    }

    cv::Mat colorMap;
    // Map ComboBox index to OpenCV colormap ID
    int cvColormapId = -1;  // Default to an invalid value

    switch (id) {
    case 0: cvColormapId = cv::COLORMAP_AUTUMN; break;
    case 1: cvColormapId = cv::COLORMAP_BONE; break;
    case 2: cvColormapId = cv::COLORMAP_JET; break;
    case 3: cvColormapId = cv::COLORMAP_WINTER; break;
    case 4: cvColormapId = cv::COLORMAP_RAINBOW; break;
    case 5: cvColormapId = cv::COLORMAP_OCEAN; break;
    case 6: cvColormapId = cv::COLORMAP_SUMMER; break;
    case 7: cvColormapId = cv::COLORMAP_SPRING; break;
    case 8: cvColormapId = cv::COLORMAP_COOL; break;
    case 9: cvColormapId = cv::COLORMAP_HSV; break;
    case 10: cvColormapId = cv::COLORMAP_PINK; break;
    case 11: cvColormapId = cv::COLORMAP_HOT; break;
    case 12: cvColormapId = cv::COLORMAP_PARULA; break;
    case 13: cvColormapId = cv::COLORMAP_MAGMA; break;
    case 14: cvColormapId = cv::COLORMAP_INFERNO; break;
    case 15: cvColormapId = cv::COLORMAP_PLASMA; break;
    case 16: cvColormapId = cv::COLORMAP_VIRIDIS; break;
    case 17: cvColormapId = cv::COLORMAP_CIVIDIS; break;
    case 18: cvColormapId = cv::COLORMAP_TWILIGHT; break;
    case 19: cvColormapId = cv::COLORMAP_TWILIGHT_SHIFTED; break;

    default:
        // Handle invalid index or provide a default colormap
        QMessageBox::warning(this, "WARNING", "Invalid colormap selection");
        return;
    }

    if (cvColormapId == -1) {
        // Handle the case when the colormap ID is not valid
        QMessageBox::warning(this, "WARNING", "Invalid colormap selection");
        return;
    }

    cv::applyColorMap(gradientImage, colorMap, cvColormapId);

    // Display the colormap gradient
    QImage colormapImage = cvMatToQImage(colorMap);
    // Rotate the colormap image by 90 degrees clockwise
    colormapImage = colormapImage.transformed(QMatrix().rotate(90));
    ui->Colormap_label->setPixmap(QPixmap::fromImage(colormapImage).scaled(ui->Colormap_label->width(), ui->Colormap_label->height()));
}
```

PART 3: 讓使用者以 Spinbox 選擇 number of cluster

```
void MainWindow::on_Cluster_spinBox_valueChanged(int arg1)
{
    // Define the number of clusters
    ui->Cluster_spinBox->setMinimum(2);
    ui->Cluster_spinBox->setMaximum(10);
    num_clusters = ui->Cluster_spinBox->value();
}
```

K-means 的轉換:

```
void MainWindow::on_Kmeans_pushButton_clicked()
{
    // Read the input image
    Mat image = QImageToCvMat(rgbImage);

    if (image.empty()) {
        std::cerr << "Error: Could not read the image." << std::endl;
    }

    // Reshape the image to a 2D array of pixels (each row is a pixel, and each column is a color channel)
    Mat reshaped_image = image.reshape(1, image.rows * image.cols);
    reshaped_image.convertTo(reshaped_image, CV_32F);

    // Apply k-means clustering
    Mat labels, centers;
    kmeans(reshaped_image, num_clusters, labels, TermCriteria(TermCriteria::EPS + TermCriteria::MAX_ITER, 100, 0.2), 3, KMEANS_RANDOM_CENTERS, centers);

    // Reshape the result back to the original image size
    Mat segmented_image = Mat::zeros(image.size(), image.type());
    MatIterator_<Vec3b> it, end;
    int i = 0;
    for (it = segmented_image.begin<Vec3b>(), end = segmented_image.end<Vec3b>(); it != end; ++it, ++i) {
        int cluster_idx = labels.at<int>(i);
        (*it)[0] = centers.at<float>(cluster_idx, 0);
        (*it)[1] = centers.at<float>(cluster_idx, 1);
        (*it)[2] = centers.at<float>(cluster_idx, 2);
    }
    QImage Qsegmented_image = cvMatToQImage(segmented_image);

    // Display the original and segmented images
//    const int w = ui->label_2->width();
//    const int h = ui->label_2->height();
//    ui->label_2->setPixmap(QPixmap::fromImage(MatToImage(segmented_image).scaled(w,h,Qt::KeepAspectRatio)));
    displayImageOnLabel(Qsegmented_image, ui->Converted_Img_label);

}
```

HIS 與 Lab 的 K-means 轉換:

```
void MainWindow::on_Kmeans_HSI_LAB_pushButton_clicked()
{

    ui->Cluster_spinBox->setValue(3);
    Mat srcImage = QImageToCvMat(rgbImage);
    // Reshape the image to a 2D array of pixels
    Mat reshaped_image = srcImage.reshape(1, srcImage.rows * srcImage.cols);
    reshaped_image.convertTo(reshaped_image, CV_32F);

    // Convert to HSI color space
    Mat hsi_image;
    cvtColor(srcImage, hsi_image, COLOR_BGR2HSV);

    // Reshape the HSI image to a 2D array of pixels
    Mat reshaped_hsi = hsi_image.reshape(1, hsi_image.rows * hsi_image.cols);
    reshaped_hsi.convertTo(reshaped_hsi, CV_32F);

    // Convert to L*a*b* color space
    Mat lab_image;
    cvtColor(srcImage, lab_image, COLOR_BGR2Lab);

    // Reshape the L*a*b* image to a 2D array of pixels
    Mat reshaped_lab = lab_image.reshape(1, lab_image.rows * lab_image.cols);
    reshaped_lab.convertTo(reshaped_lab, CV_32F);

    // Number of clusters (you can adjust this)
    int num_clusters = 3;

    // K-means clustering for BGR
    Mat labels, centers;
    kmeans(reshaped_image, num_clusters, labels, TermCriteria(TermCriteria::EPS + TermCriteria::COUNT, 10, 1.0), 3, KMEANS_RANDOM_CENTERS, centers);

    // Reshape the result back to the original image size
    Mat segmented_image_bgr = Mat::zeros(srcImage.size(), srcImage.type());
    MatIterator_<Vec3b> it_bgr, end_bgr;
    int i_bgr = 0;
    for (it_bgr = segmented_image_bgr.begin<Vec3b>(), end_bgr = segmented_image_bgr.end<Vec3b>(); it_bgr != end_bgr; ++it_bgr, ++i_bgr) {
        int cluster_idx = labels.at<int>(i_bgr);
        (*it_bgr)[0] = centers.at<float>(cluster_idx, 0);
        (*it_bgr)[1] = centers.at<float>(cluster_idx, 1);
        (*it_bgr)[2] = centers.at<float>(cluster_idx, 2);
    }

    // K-means clustering for HSI
    kmeans(reshaped_hsi, num_clusters, labels, TermCriteria(TermCriteria::EPS + TermCriteria::COUNT, 10, 1.0), 3, KMEANS_RANDOM_CENTERS, centers);

    // Reshape the result back to the original image size
    Mat segmented_image_hsi = Mat::zeros(srcImage.size(), srcImage.type());
    MatIterator_<Vec3b> it_hsi, end_hsi;
    int i_hsi = 0;
    for (it_hsi = segmented_image_hsi.begin<Vec3b>(), end_hsi = segmented_image_hsi.end<Vec3b>(); it_hsi != end_hsi; ++it_hsi, ++i_hsi) {
        int cluster_idx = labels.at<int>(i_hsi);
        (*it_hsi)[0] = centers.at<float>(cluster_idx, 0);
        (*it_hsi)[1] = centers.at<float>(cluster_idx, 1);
        (*it_hsi)[2] = centers.at<float>(cluster_idx, 2);
    }

    // K-means clustering for L*a*b*
    kmeans(reshaped_lab, num_clusters, labels, TermCriteria(TermCriteria::EPS + TermCriteria::COUNT, 10, 1.0), 3, KMEANS_RANDOM_CENTERS, centers);

    // Reshape the result back to the original image size
    Mat segmented_image_lab = Mat::zeros(srcImage.size(), srcImage.type());
    MatIterator_<Vec3b> it_lab, end_lab;
    int i_lab = 0;
    for (it_lab = segmented_image_lab.begin<Vec3b>(), end_lab = segmented_image_lab.end<Vec3b>(); it_lab != end_lab; ++it_lab, ++i_lab) {
        int cluster_idx = labels.at<int>(i_lab);
        (*it_lab)[0] = centers.at<float>(cluster_idx, 0);
        (*it_lab)[1] = centers.at<float>(cluster_idx, 1);
        (*it_lab)[2] = centers.at<float>(cluster_idx, 2);
    }

    // Display the segmented images
    displayImageOnLabel(cvMatToQImage(segmented_image_bgr), ui->Converted_Img_label);
    displayImageOnLabel(cvMatToQImage(segmented_image_hsi), ui->Converted_Img_label_2);
    displayImageOnLabel(cvMatToQImage(segmented_image_lab), ui->Converted_Img_label_3);
}
```


MainWindow
Select an image
RGB
HSV
HSI
CMY
SUMMER
K-means
2
HSI_k-means
YUV
LAB
XYZ