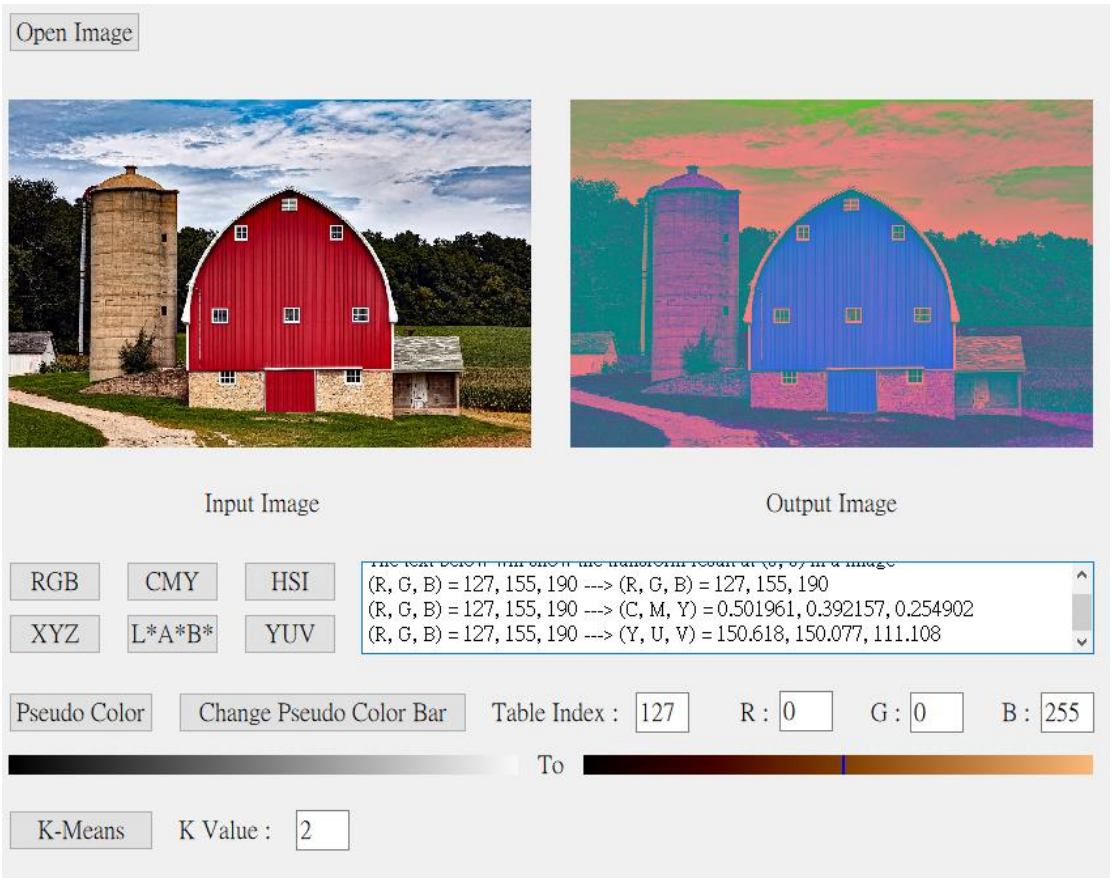
Digital Image Processing HW05 Report

生機碩一 R09631007 吳乙澤

一、軟體簡介

本次作業所寫介面如下圖所示。一開始須先 Open Image 後，才能做後續的色彩轉換。按下 RGB、CMY、HSI……等按鈕後，程式便會把結果圖顯示在Output Image處。透過右方的顯示框，可以看到圖片座標為(0, 0)之像素值經色彩轉換後的結果。轉換後，我會斟酌乘於255使其對比加大，以利顯示清楚。下方的Pseudo Color主要是對灰階圖片起作用，軟體本身有預設一種假彩色的效果。若要自訂，使用者可於 index 和 R、G、B 處，輸入0~255的數值。輸入完畢後，按下Change Pseudo Color Bar，色彩條便會改變。接著再按下Pseudo Color，便能改變假彩色的效果。最後，K-Means的部分，需先按下 RGB、HSI、L*A*B* 等按鈕並輸入K值後，再按下K-Means，程式便會以 imshow 另開一個視窗，並將色彩分割的結果顯示在該視窗上。



軟體介面圖

二、演算法說明與結果討論

本作業有三個功能需完成。以下依序說明這些功能的演算法並討論結果。

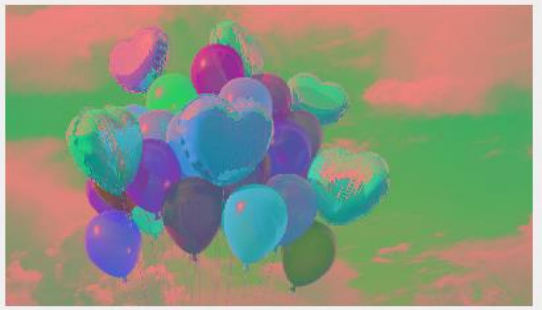
2-1 Color Model Conversion

利用課本上的公式對圖片做色彩模型的轉換。見下方程式碼，該處舉 CMY 色彩空間為例。以兩個 for 迴圈從圖片取出像素值後，再根據公式轉換及可。

```
for (int i = 0 ; i < imgCols ; i++)
    for (int j = 0 ; j < imgRows ; j++)
       double C = (255-R)/255, M = (255-G)/255, Y = (255-B)/255;
```

轉換後，我直接以 R、G、B 色彩空間將其顯示出來，因此色彩上看起來會有些怪異。尤其是 L*A*B 有負值的緣故，因此若以 RGB 色彩空間來顯示 L*A*B 轉換後的結果，圖片常常會偏暗一些，乘以 255 後會漂亮許多。另外，L*A*B* 的轉換結果會根據光源而有所不同。本程式根據 CIE L*a*b* Color Scale (rdsor.ro) 中的敘述，將光源設成 CIE 10 Degree Standard Observer 中的 D60 光源。


RGB 轉成 HSI

RGB 轉成 YUV

RGB 轉成 L*A*B，未乘 255

RGB 轉成 L*A*B，有乘 255

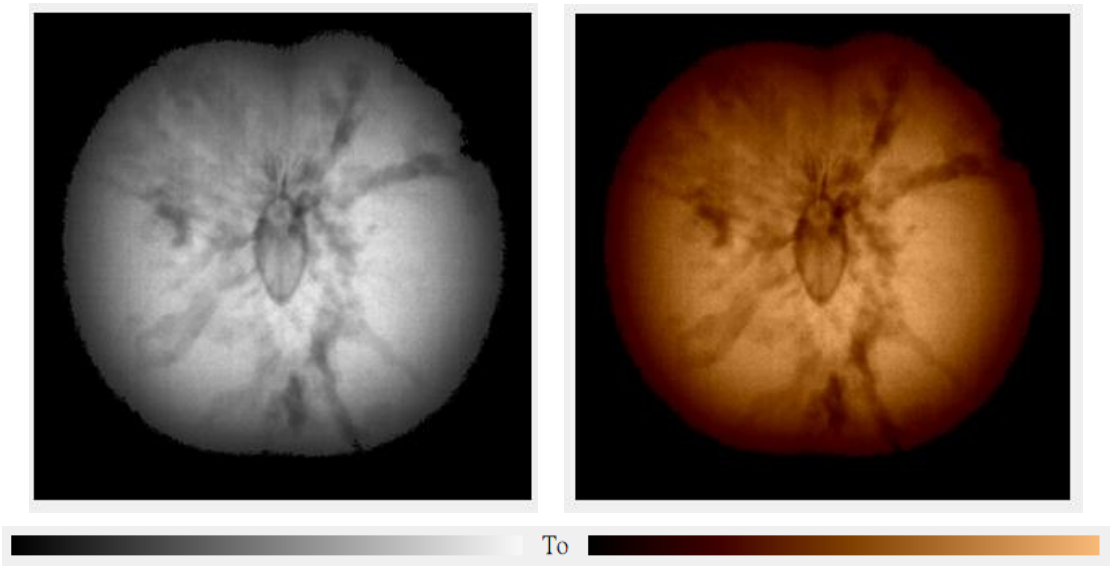
2-2 Pseudo-color Image

該題是將灰階影像以假彩色的方式顯示出來。見下方程式碼，我以一個二維陣列 pseudoColorTable，儲存灰階圖片至彩色圖片的對應表。在使用者自訂 index 和 R、G、B 的分量後，程式會依據 index 更改 pseudoColorTable 中對應的元素值。如此一來，使用者便能以 RGB 色彩空間，自定義 pseudoColorTable，進而變更灰階影像的假彩色效果。

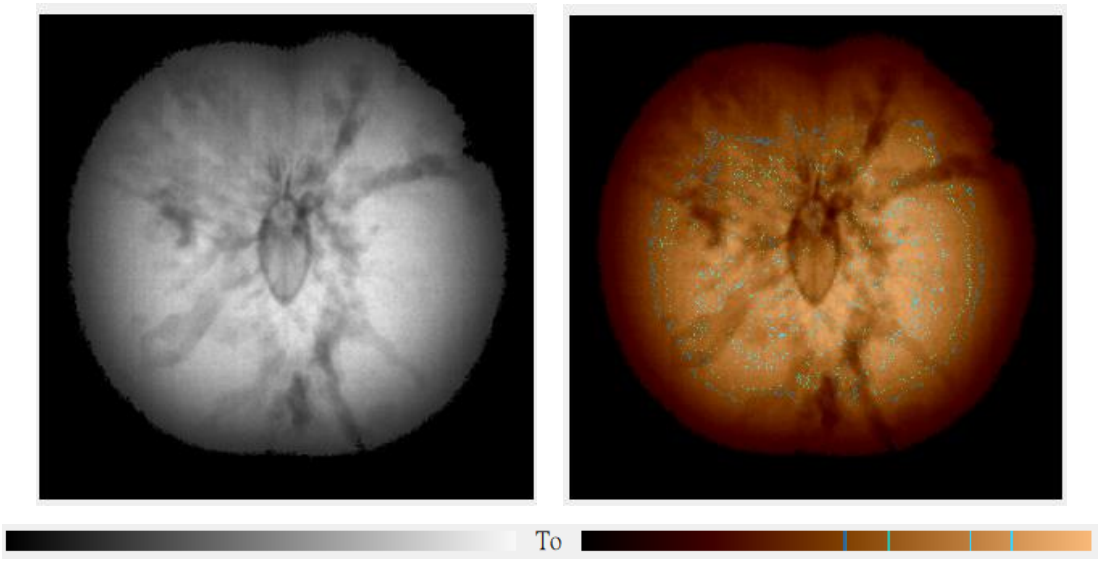
```
for (int i = 0 ; i < 256 ; i++)
{
    if (i == 0)
        R = G = B = 0;
    else
        R = i;
        G = i-64;
        B = i-128;
```

```
    pseudoColorTable[i][0] = int(checkPixel(R));
    pseudoColorTable[i][1] = int(checkPixel(G));
    pseudoColorTable[i][2] = int(checkPixel(B));
}
```

見下圖，該圖為經過預設的 pseudoColorTable 後，所變換出的假彩色灰階影像。影像下方則是色彩轉換前後的色彩條。經過我隨意調整 pseudoColorTable 中的四個數值後，會出現一點一點的雜點。推測是我隨意調整 index 和色彩值，進而造成圖片有色彩不連續的像素點產生。



預設的假彩色效果



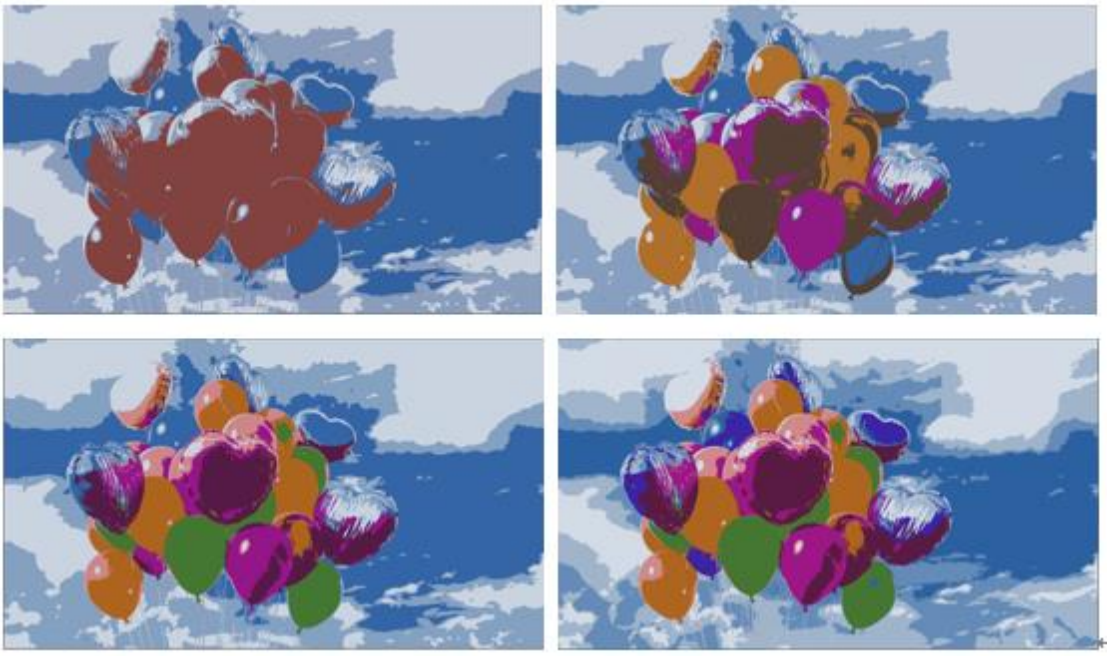
經過使用者自訂後的假彩色效果

2-3 Color Segmentation

見下方程式碼，利用 for 迴圈將圖片資料給 kmeansData 後，利用 OpenCV 的 kmeans 函數，根據 k 值從圖片中找到 k 個色彩中心及每個座標的標籤。之後，根據色彩中心和標籤，便能將色彩切割的結果顯示出來。另外，由於我 HSI 的 kmeans 演算法有些問題，可能會導致部分圖片所算出的 label、center 為空。因此，我利用 if/else 來判斷，避免程式出現 error 而整個當掉。

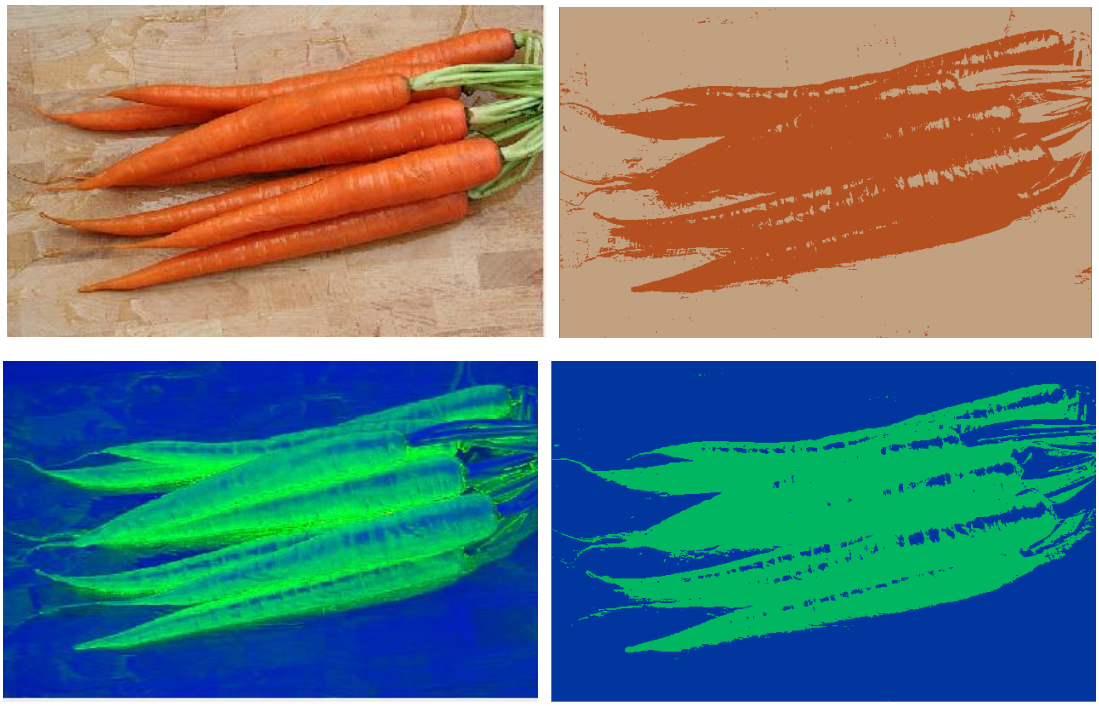
```
for (int i = 0 ; i < imgRows ; i++)
   for (int j = 0 ; j < imgCols ; j++)
      for (int k = 0 ; k < 3 ; k++)
        kmeansData.at<float>(i+j*imgRows,k)=MatImgIn.at<Vec3b>
        (i,j)[k];

kmeans(kmeansData,k,label,TermCriteria(TermCriteria::COUNT+Ter
mCriteria::EPS, 30, 1e-6), 5, KMEANS_RANDOM_CENTERS, center);

if (!label.empty() && !center.empty())
   for (int i = 0 ; i < imgRows ; i++)
      for (int j = 0 ; j < imgCols ; j++)
         kmeansResult.at<Vec3b>(i,j)[k]=center.at<float>(index
         ,k);
else
  cout << "label and center are empty" << endl;
```

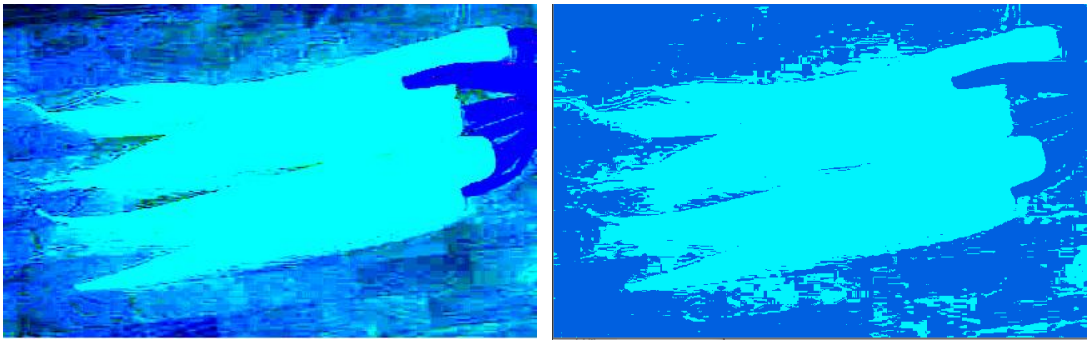
以下顯示同一張圖片，不同 k 值下的色彩分割結果。可以從結果圖明顯地看出，隨著 k 值的上升，圖片會越來越接近原圖，並且有假輪廓的現象發生。想來這跟之前寫過的灰階量化類似，灰階量化、色彩分割雖然皆減少顯示的細膩度，但仍能保有大致輪廓。見下圖，k = 4 時，也就是使用四種色彩，便能看出氣球的邊緣了。

從左上到右下分別為原圖、k = 2、k = 4、k = 6、k = 8、k = 10

最後，下圖是 RGB、HSI、L*A*B*色彩空間中，k = 2 的顯示結果。可以從中看出，不同色彩空間，kmeans 做的事情仍然是一樣的。並且，RGB、HSI 下，胡蘿蔔的輪廓十分明顯，L*A*B*下就比較模糊一些。推測與色彩空間的顯示特性有所關連，才造成相同 k 值下的不同分割結果。





由上至下分別是 RGB、HSI、L*A*B*