想法（演算法）

構思是以人眼視覺判定，再轉為程式，此做法較為直觀、好理解，且除錯較簡單。

原始圖片有許多的線條，剛開始是對不消去的原圖做偵測，直接做偵測，但每條線條的灰階值不一，甚至有些圖片的線條不清楚，或根本沒有，常常偵測時，過度敏感，使大部分的偵測產生奇怪的結果。此演算法會變的比不做邊緣模糊來的複雜許多，為模糊邊緣以達到消除原圖線條，使用了高斯消去法，以5做核心，在缺乏基礎知識的情況下，常常將核心值(kernelSize)胡亂設定，導致程式直接進入無止盡的工作。

原先使用的方法是以第一個值做參考就將該值列為參考，再與下值做比較，同樣使用Value做變化容許值。若在可容許範圍，改為該值做參考，在往下比較，以此類推。

這方法會被瑕疵周邊的變化量影響，也使變化容許值的範圍減少，無法彈性使用其他演算法，且每張圖的容許變化量無法統一，使偵測的演算法變複雜。

後改良為只取第一點做參考，此方法無法使"下方白線"與汙3指出正確的瑕疵點，反而偵測到非瑕疵的地方。也會使偵測的演算法變的複雜。

最後使用現在的方法，將每行平均取值，此法可改善上述的辦法，擴大變化容許值，以支援各演算法，使汙3指出正確的瑕疵點，但是"下方白線" 會無法偵測到正常的瑕疵點，因為此瑕疵點與其他的圖片顏色相近，想以求邊緣的演算法進行過濾，剛開始使用拉普拉斯(Laplace)，但效果不加，為使附近的顏色有明顯的區別，改成使用Sobel 運算子，明顯的加大變化量，在使用該方法後，因Sobel運算子會使圖片顏色變化極大，原本預設的變化容許值會無法正常運作，所以將容許值放大8倍，方可找到正確的瑕疵點。

全域變數

Image<Rgb, Byte> Source, Display;

Bitmap MapSoure;

Source 原圖 ， Display展示圖

MapSoure 原圖的Bitmap

button7\_Click

OpenFileDialog OP = new OpenFileDialog();

if (OP.ShowDialog() == DialogResult.OK)

{

Source = new Image<Rgb, Byte>(OP.FileName);

MapSoure = Source.Bitmap;

Display = Source;

pictureBox1.Image = MapSoure;

label1.Text = "品質:";

}

簡單的 開起圖片與初始化資料

為將OPencv 格式轉成C#可處理，直接以原圖Bitmap做複製

Void 函數Func

(Bitmap toDisplay, int Value)

int Width = MapSoure.Width, Height = MapSoure.Height, WHBound = 10;

取得圖片的寬高。

設定邊界，以避免高斯消去法產生的邊界極值

Boolean Flag = false

最終要傳回的值

for (int X = WHBound; X < Width - WHBound; X++)

{

int Sum = 0;

初始加總值

for (int Y = WHBound; Y < Height - WHBound; Y++)

Sum += toDisplay.GetPixel(X, Y).R;

取得加總值

int Average = Sum / (Height - WHBound \* 2);

除以抓取的元素的數量

for (int Y = WHBound; Y < Height - WHBound; Y++)

{

int R = toDisplay.GetPixel(X, Y).R;

抓取當前的灰階值R

if (Value < Math.Abs(Average - R))

取參考值與當前灰階的距離，當大於容許值

{

MapSoure.SetPixel(X, Y, Color.Red);

將該值設為紅色

if (!Flag) 檢察旗標

{

Color pixel = MapSoure.GetPixel(X - 1, Y), pixel2 = MapSoure.GetPixel(X, Y - 1);

抓取左與上的pixel

if ((pixel.R == 255 && pixel.R != pixel.G) || (pixel2.R == 255 && pixel2.R != pixel2.G))

若為紅色連點，旗標設為真

Flag = true;

}

}

}

if (Flag)

pictureBox1.Image = MapSoure;

旗標為真，直接顯示

return Flag ;

傳回旗標

}

button8\_Click

此為演算法的函數

int Value = 10;

此為容許值

for (int i = 0; i < Value; i++)

{

Display.\_SmoothGaussian(5);

}

重複高斯消去法

if (Func(Display.Bitmap, Value))

{

使用函數1進行檢查

label1.Text = "品質: 瑕疵";

若成立則直接判為瑕疵

}

else

若無，進行二次檢查

{

Display = Display.Sobel(0, 1, 25).Convert<Rgb, Byte>();

做Sobel運算子

Value \*= 8;

將容許值加大

if (Func(Display.Bitmap, Value))

{

再次檢查

label1.Text = "品質: 瑕疵";

若成立，則為瑕疵

}

else

label1.Text = "品質: OK";

若無則OK

}

}

}