

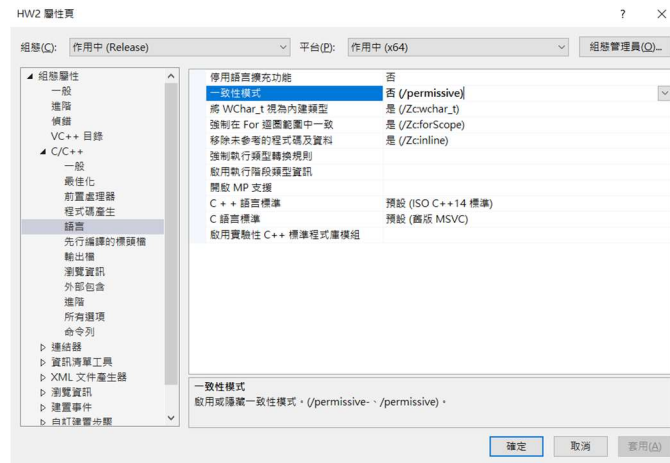
高等電腦視覺 作業四

姓名: 陳祐毅 學號:109618028

OpenCV

編譯環境 : Visual Studio 2019 (16.11.5) 、 opencv (4.4.0)

如果遇到 `const char to char` 相關的 error , 請在
屬性 > C/C++ > 語言 > 一致性模式 改成否



輸入影片請放到 “HW4_109618028” 資料夾內

bonus	2021/12/30 上午 01:19	檔案資料夾	
cv2	2021/12/30 上午 01:14	檔案資料夾	
WM61b1c81394b01.avi	2021/12/9 下午 08:53	AVI 檔案	7,991 KB
第四次作業_陳祐毅_109618028.docx	2021/12/30 上午 01:08	Microsoft Word ...	4,995 KB

有需要修改輸入影片路徑的話請修改：

```
47  const char* src = "../WM61b1c81394b01.avi"; // 修改輸入影片路徑
```

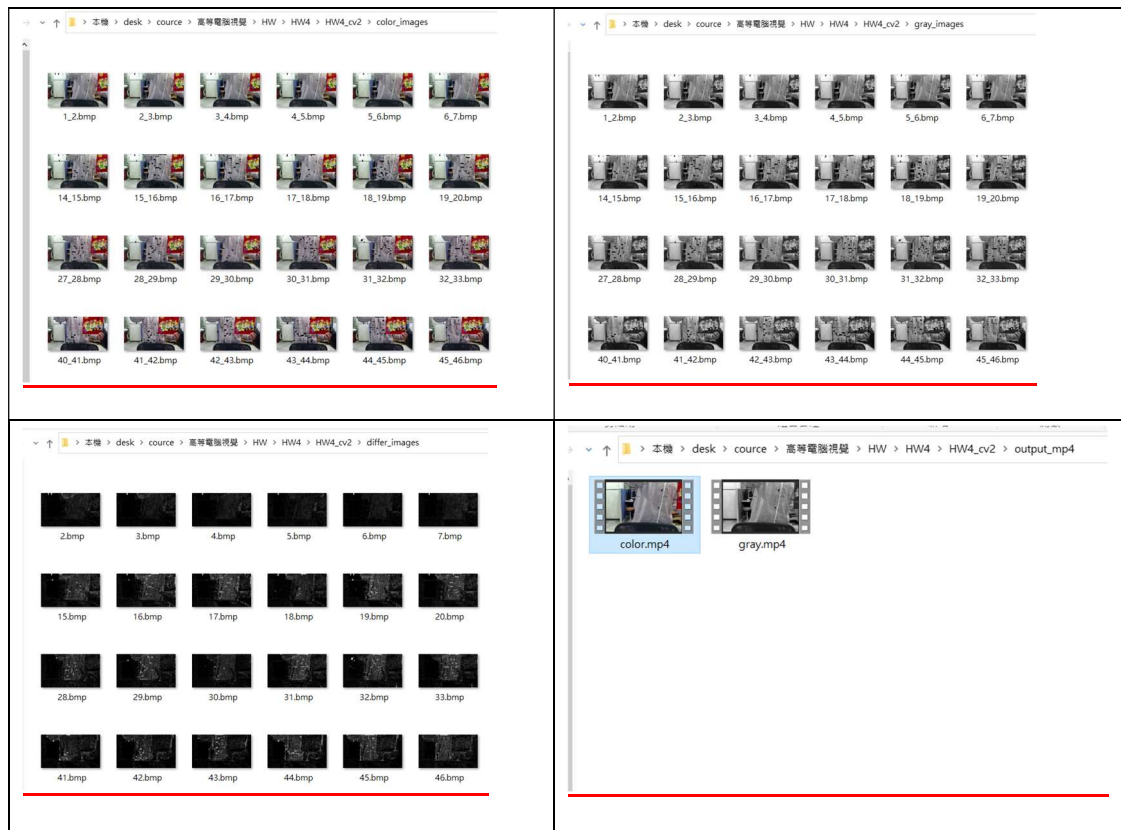
輸出會自動建立四個資料夾，分別是：

color_images：預測的所有影像（彩色）

gray_images：預測的所有影像（灰階）

differ_images：所有的實際影像與預測影像的差值

output_mp4：彩色/灰階 的預測影像之影片



引用函式庫

```
1 #include<opencv2/opencv.hpp>
2 #include<opencv2/highgui.hpp>
3 #include<iostream>
4 #include<string>
5 #include <io.h>    //創 img資料夾用
6 #include <direct.h> //創 img資料夾用
```

過程介紹

當影片可以順利串流後，我將影像處理的程序放在 **while** 迴圈內，每一輪都會記錄下當前的灰階影像(以下簡稱 **now**) 和 上一幀的灰階影像(以下簡稱 **pre**)。

```
65 //影片串流
66 while (capture.read(frame)) {
67     cvtColor(frame, grayImg, COLOR_BGR2GRAY);
68     frame.copyTo(colorImg);
69     grayImg.copyTo(nowImg);
70     //影像處理
71     if (count>0) {
72
73         // 影像處理
74         outputImg = template_matching(preImg, nowImg, colorImg, &output_grayImg, &output_difImg);
75
76         //輸出影片
77         imshow("video", outputImg);
78         waitKey(10);
79
80
81         //儲存影片
82         writer.write(outputImg);
83         writer_gray.write(output_grayImg);
84
85         //儲存照片
86         sprintf_s(text, "./gray_images/%d_%d.bmp", count, count + 1);
87         imwrite(text, output_grayImg);
88         sprintf_s(text, "./differ_images/%d.bmp", count + 1);
89         imwrite(text, output_difImg);
90         sprintf_s(text, "./color_images/%d_%d.bmp", count, count+1);
91         imwrite(text, outputImg);
92
93     }
94
95     count++;
96     nowImg.copyTo(preImg);
97 }
98 }
```

將 **pre** 影像分成許多 40x40 的 grid(template)，由於輸入的影像是 1280x720，所以總共會有 32x18 個 grids，使用 **for** 迴圈對遍歷每個 template：

```
194     for (int c = 0; c < grid_c; c++) {
195         for (int r = 0; r < grid_r; r++) {
```

在迴圈內分別為每個 **template** 對 **now** 影像中相同位置的 **search range**(-20~20)作模板匹配：

```
212         //模板匹配
213         matchTemplate(exhaustive_img, temp_img, match_img, TM_CCOEFF_NORMED);
214         minMaxLoc(match_img, &minVal, &maxVal, &minLoc, &maxLoc);
215     }
```

匹配完成後分別將 **pre** 影像中的所有 **grid** 移動到匹配到的新位置，得到一張預測影像：

```
224     for (int r = 0; r < 40; r++) {
225         for (int c = 0; c < 40; c++) {
226             for (int i = 0; i < 3; i++) {
227                 // 移動grid到新的位置 ( 彩色 )
228                 out_img.at<Vec3b>(aft_y + r, aft_x + c)[i] = colorImg.at<Vec3b>(pre_y + r, pre_x + c)[i];
229             }
230             // 移動grid到新的位置 ( 灰色 )
231             out_gray_img.at<uchar>(aft_y + r, aft_x + c) = pre.at<uchar>(pre_y + r, pre_x + c);
232         }
233     }
```

最後算 **now** 影像與預測影像的差值：

(在這我直接使用 **NOR** 邏輯，用這個的好處是運算比較快)








```
240     //算 now 與 predict 差值
241     out_dif_img = now ^ out_gray_img;
```

儲存


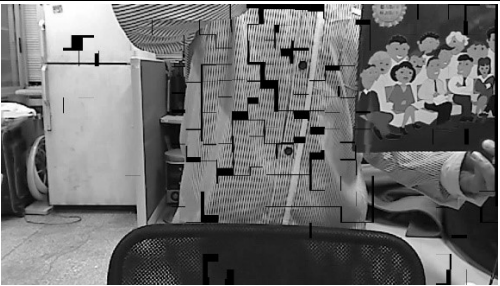
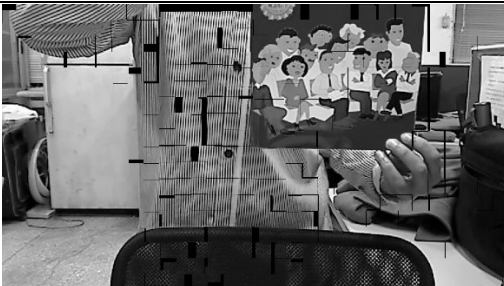
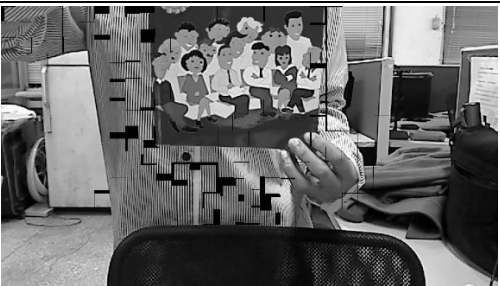



```
81     //儲存影片
82     writer.write(outputImg);
83     writer_gray.write(output_grayImg);
84
85     //儲存照片
86     sprintf_s(text, "./gray_images/%d_%d.bmp", count, count + 1);
87     imwrite(text, output_grayImg);
88     sprintf_s(text, "./differ_images/%d.bmp", count + 1);
89     imwrite(text, output_difImg);
90     sprintf_s(text, "./color_images/%d_%d.bmp", count, count + 1);
91     imwrite(text, outputImg);
```

結果



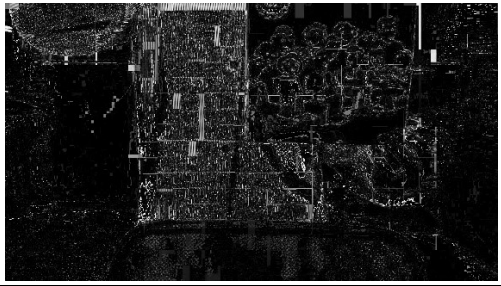
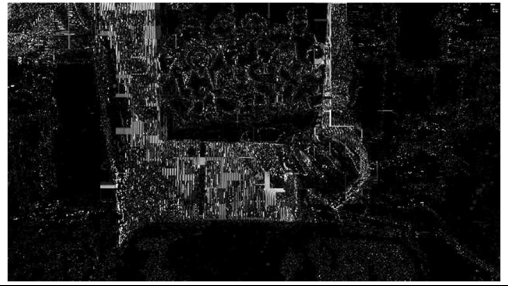
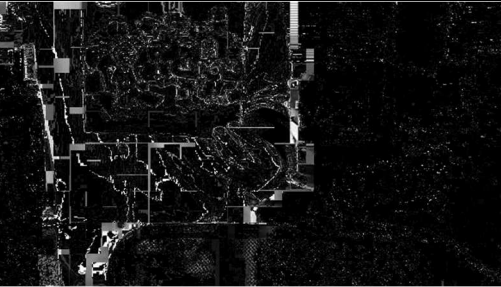
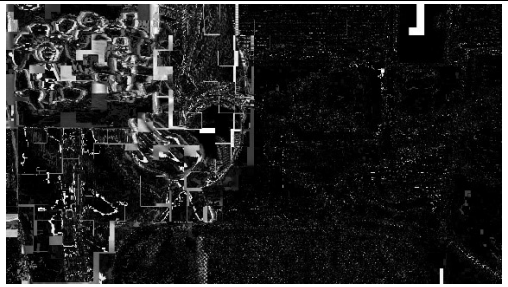

彩色影像(每 30 個 frame)

Frame = 2	Frame = 32
	
Frame = 62	Frame = 92
	
Frame = 122	Frame = 152
	
Frame = 176	
	

灰階影像(每 30 個 frame)

Frame = 2	Frame = 32
	
Frame = 62	Frame = 92
	
Frame = 122	Frame = 152
	
Frame =176	
	

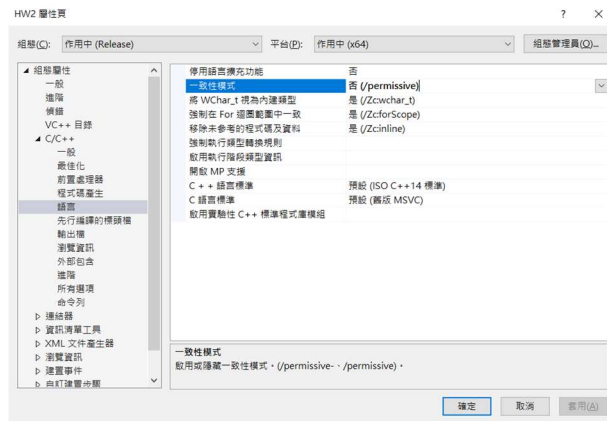
差值影像 (每 30 個 frame)

Frame = 2	Frame = 32
	
Frame = 62	Frame = 92
	
Frame = 122	Frame = 152
	
Frame =176	
	

Bonus

編譯環境：Visual Studio 2019 (16.11.5) 、 opencv (4.4.0)

如果遇到 `const char to char` 相關的 error，請在
屬性 > C/C++ > 語言 > 一致性模式 改成否



輸入影片請放到 **“HW4_109618028”** 資料夾內

有需要修改輸入影片路徑的話請修改：

```
42      const char* src = "../WM61b1c81394b01.avi"; // 修改輸入影片路徑
```

輸出會自動建立四個資料夾，分別是：

color_images：預測的所有影像（彩色）

gray_images：預測的所有影像（灰階）

differ_images：所有的實際影像與預測影像的差值

output_mp4：彩色/灰階 的預測影像之影片

在 terminal 上會顯示每次的 SNR 以及計算時間（最後會顯示平均 SNR 以及總花費時間）：

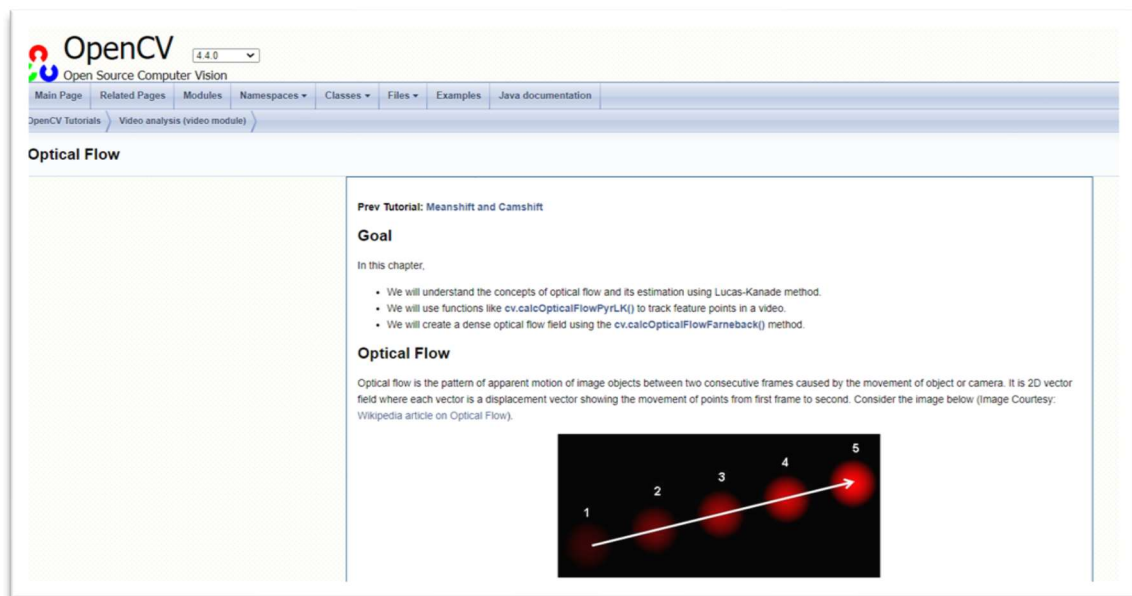
由下表可以看出，光流法花費的時間大約是模板匹配的三倍。

Microsoft Visual Studio 偵錯主控台

Frame	SNR	光流法花費時間	模板匹配花費時間
148	21.0048	0.169(s)	0.058(s)
149	27.2122	0.176(s)	0.063(s)
150	28.7486	0.173(s)	0.052(s)
151	29.1663	0.165(s)	0.049(s)
152	28.8224	0.17(s)	0.058(s)
153	29.6208	0.171(s)	0.061(s)
154	28.7778	0.167(s)	0.06(s)
155	26.3989	0.168(s)	0.058(s)
156	34.9326	0.174(s)	0.05(s)
157	30.4268	0.168(s)	0.059(s)
158	22.7147	0.172(s)	0.058(s)
159	33.38	0.17(s)	0.06(s)
160	24.5968	0.165(s)	0.062(s)
161	24.4277	0.176(s)	0.059(s)
162	24.6139	0.165(s)	0.059(s)
163	23.9502	0.174(s)	0.061(s)
164	21.6735	0.168(s)	0.059(s)
165	21.4369	0.164(s)	0.058(s)
166	19.4258	0.173(s)	0.061(s)
167	16.9527	0.163(s)	0.061(s)
168	19.9696	0.165(s)	0.06(s)
169	19.4508	0.176(s)	0.053(s)
170	18.7075	0.165(s)	0.052(s)
171	17.5489	0.167(s)	0.063(s)
172	19.0781	0.169(s)	0.064(s)
173	20.3771	0.183(s)	0.048(s)
174	22.267	0.173(s)	0.059(s)
175	19.5415	0.171(s)	0.058(s)

平均 SNR = 23.633，光流法花費總時間：29.796(s)，模板匹配花費總時間：10.27(s)

光流法採用官方教學的 Optical Flow：


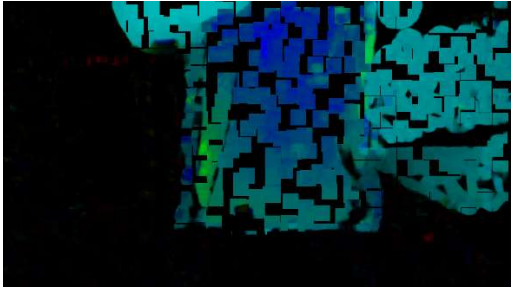
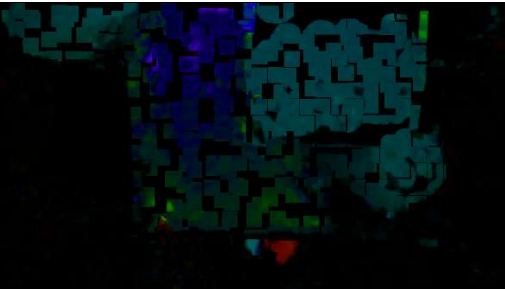
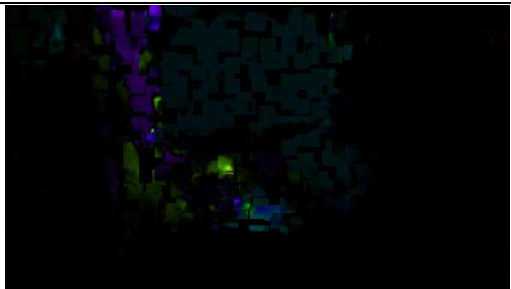





模板匹配與影像重建的部分與上面我寫的 opencv 程式一樣，SNR 的部分如下：
(比較的對象是預測的灰階影像(reconstruct)和原始的灰階影像)




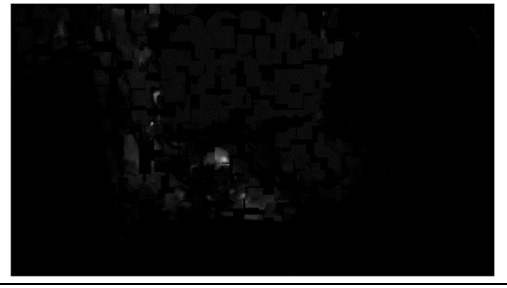


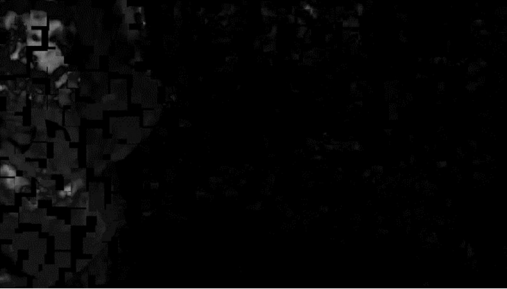
```
154 double snr(const Mat& I1, const Mat& I2) {
155     int n = I1.rows * I1.cols;
156     Scalar avg_ = sum(I1) / n;
157
158     Mat diff_img;
159     absdiff(I2, I1, diff_img);
160     Scalar avg_n = sum(diff_img) / n;
161
162     double vs = 0;
163     double vn = 0;
164
165     for (int r = 0; r < I1.rows; r++) {
166         for (int c = 0; c < I1.cols; c++) {
167             vs += pow((I1.at<uchar>(r, c) - avg_[0]), 2);
168             vn += pow((I2.at<uchar>(r, c) - I1.at<uchar>(r, c) - avg_n[0]), 2);
169         }
170     }
171     vs = vs / n;
172     vn = vn / n;
173     double snr = 20 * log10((pow(vs, 0.5), pow(vn, 0.5)));
174
175     return snr;
176 }
177 }
```

結果


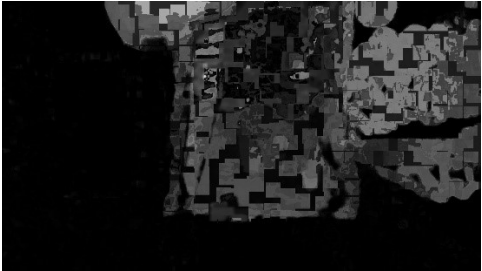
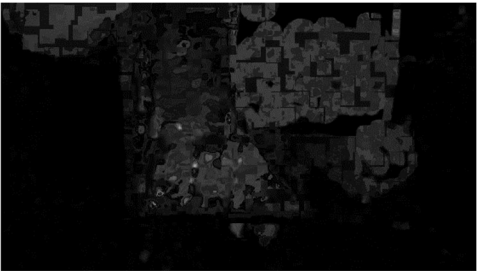


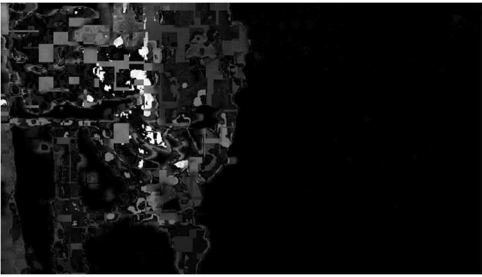
彩色影像(每 30 個 frame)

Frame = 2	Frame = 32
	
Frame = 62	Frame = 92
	
Frame = 122	Frame = 152
	
Frame =175	
	

灰階影像(每 30 個 frame)

Frame = 2	Frame = 32
	
Frame = 62	Frame = 92
	
Frame = 122	Frame = 152
	
Frame =175	
	

差值影像 (每 30 個 frame)

Frame = 2	Frame = 32
	
Frame = 62	Frame = 92
	
Frame = 122	Frame = 152
	
Frame =175	
