

Lab 11: 基於 LBP feature 的人臉偵測

404410030 資工三 鄭光宇

## 實驗目的

學習使用 Raspberry Pi 做一些實際的應用。嘗試用 Pi+OpenCV 執行 Real-time 的人臉偵測。

## 實驗步驟

首先在 Pi 上安裝 OpenCV

這部分可以參見 Lab.9

## 更新 Kernel 與套件

```
sudo rpi-update  
reboot # 重開機  
sudo apt-get update # 更新套件清單  
sudo apt-get upgrade # 更新套件  
sudo apt-get install build-essential git cmake pkg-config # 安裝編譯所需套件  
sudo apt-get install libjpeg-dev libtiff-dev libjasper-dev libpng-dev libavcodec-dev  
libavformat-dev libswscale-dev libv4l-dev libxvidcore-dev libx264-dev libgtk2.0  
-dev libatlas-base-dev gfortran
```

## 安裝 OpenCV

查詢可用的套件版本號

```
apt-cache search opencv
```

假設找到的是 2.4 版

```
sudo apt-get install libcv2.4 libcvaux2.4 libhighgui2.4  
sudo apt-get install libcv-dev libcvaux-dev libhighgui-dev libopencv-dev
```

## USB Camera

使用 UV4L 控制 USB Camera 如果你的 Raspbian 版本是 wheezy

```
curl http://www.linux-project.org/listing/uv4l_repo/lrkey.asc | sudo apt-key add -  
-  
echo "deb http://www.linux-project.org/listing/uv4l_repo/raspbian/ wheezy main" >  
> /etc/apt/sources.list
```

如果是 scratch

```
curl http://www.linux-project.org/listing/uv4l_repo/lpkey.asc | sudo apt-key add -  
-  
echo "deb http://www.linux-project.org/listing/uv4l_repo/raspbian/stretch stretch  
main" >> /etc/apt/sources.list
```

更新套件清單

```
sudo apt-get update
```

安裝

```
sudo apt-get install uv4l uv4l-raspicam uv4l-raspicam-extras
```

啟動服務

```
sudo service uv4l_raspicam restart
```

安裝 uv4l 的工具軟體

```
sudo apt-get install v4l-utils fswebcam
```

測試

```
# 測試，拍張照  
fswebcam hello.jpg
```

## 撰寫 OpenCV 程式

用 C++ 寫一隻 OpenCV 程式，完成以下步驟

1. 初始化 GPIO 4, GPIO 5
2. 從 Webcam 讀入幀
3. 影像前處理 (轉灰階、Histogram Equalization)
4. 使用 OpenCV 上訓練好的 lbp 人臉分類器來偵測人臉
5. 若人臉在左，GPIO 4 亮起；若人臉在右，GPIO 5 亮起。其他情況均暗
6. 顯示人臉偵測結果

## 7. 若程式結束。關閉、釋放 GPIO 4, GPIO 5

以下這支程式將灰階影像以 0.666 (1/1.5) 倍解析度輸入分類器。

```
#include <iostream>
#include <ctime>
#include <sys/time.h>
#include <cstdio>
#include <opencv2/objdetect/objdetect.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <opencv2/imgproc/imgproc.hpp>

using namespace std;
using namespace cv;

/** Function Headers */
float detectAndDisplay( Mat frame );

/** Global variables */
String face_cascade_name = "lbpcascade_frontalface_improved.xml"; // 這裡放你的分類器
描述檔 (*.xml)
CascadeClassifier face_cascade;
string window_name = "Capture - Face detection";
RNG rng(12345);

void initinalize_gpio(void) {
/*
 * 開啟、初始化 GPIO port
 */
FILE *export_io = NULL;

export_io = fopen("/sys/class/gpio/export", "w");
fprintf	export_io, "4");
if (export_io!=NULL) fclose(export_io);
export_io = NULL;

export_io = fopen("/sys/class/gpio/export", "w");
fprintf	export_io, "5");
if (export_io!=NULL) fclose(export_io);
export_io = NULL;

export_io = fopen("/sys/class/gpio/gpio4/direction", "w");
fprintf	export_io, "out");
if (export_io!=NULL) fclose(export_io);
export_io = NULL;

export_io = fopen("/sys/class/gpio/gpio5/direction", "w");
fprintf	export_io, "out");
```

```

if (export_io!=NULL) fclose(export_io);
export_io = NULL;
}

void destruct_gpio(void) {
/*
 * 關閉, unexport GPIO port
 */
FILE *export_io = NULL;

export_io = fopen("/sys/class/gpio/gpio4/value", "w");
fprintf(export_io, "0");
if (export_io!=NULL) fclose(export_io);
export_io = NULL;

export_io = fopen("/sys/class/gpio/gpio5/value", "w");
fprintf(export_io, "0");
if (export_io!=NULL) fclose(export_io);
export_io = NULL;

export_io = fopen("/sys/class/gpio/unexport", "w");
fprintf(export_io, "4");
if (export_io!=NULL) fclose(export_io);
export_io = NULL;

export_io = fopen("/sys/class/gpio/unexport", "w");
fprintf(export_io, "5");
if (export_io!=NULL) fclose(export_io);
export_io = NULL;
}

void indicate_bias(float bias) {
/*
 * 判斷人臉在左 -> GPIO 4 亮 GPIO 5 暗
 * 判斷人臉在右 -> GPIO 4 暗 GPIO 5 亮
 * 其他情況全暗
 */
FILE *gpio4 = NULL;
FILE *gpio5 = NULL;
gpio4 = fopen("/sys/class/gpio/gpio4/value", "w");
gpio5 = fopen("/sys/class/gpio/gpio5/value", "w");
if (bias<0) {
    fprintf(gpio4, "1");
    fprintf(gpio5, "0");
} else if (bias>0) {
    fprintf(gpio4, "0");
    fprintf(gpio5, "1");
} else {

```

```

        fprintf(gpio4, "0");
        fprintf(gpio5, "0");
    }
    if (gpio4!=NULL) fclose(gpio4); gpio4=NULL;
    if (gpio5!=NULL) fclose(gpio5); gpio5=NULL;
}

/** @function main */
int main( int argc, const char** argv ) {
    CvCapture* capture;
    Mat frame;

    //-- 1. Load the cascades
    if( !face_cascade.load( face_cascade_name ) ) {
        printf("--(!)Error loading\n");
        return -1;
    }

    initialize_gpio(); // 初始化 GPIO

    //-- 2. Read from webcam
    capture = cvCaptureFromCAM( 0 );
    if( capture ) {
        while( true ) {
            frame = cvArrToMat(cvQueryFrame( capture ));

            //-- 3. Apply the classifier to the frame
            if( !frame.empty() ) {
                float bias = detectAndDisplay( frame ); // 找人臉偏移 x 軸正中央的位移
                indicate_bias(bias); // 輸入偏移量 判斷左右
            } else {
                printf(" --(!) No captured frame -- Break!");
                break;
            }

            if( waitKey(10)>=0 ) break;
        }
    }
    destruct_gpio(); // 關閉 GPIO
    return 0;
}

/** @function detectAndDisplay */
float detectAndDisplay( Mat frame ) {
    struct timeval tv1, tv2;
    struct timezone tz1, tz2;
    const double down_scale=1.5; // 進入分類器前 downscale 多少解析度
    static double sum = 0.0;
}

```

```

static double num = 0.0;
vector<Rect> faces;
Mat frame_gray;

cvtColor( frame, frame_gray, CV_BGR2GRAY ); // 轉灰階
equalizeHist( frame_gray, frame_gray ); // Histogram Equalization

//-- Detect faces
gettimeofday(&tv1, &tz1);
face_cascade.detectMultiScale( frame_gray, faces, down_scale, 2, 0, Size(10, 1
0) ); // 找臉
gettimeofday(&tv2, &tz2);
double diff = 1.0e6 * (tv2.tv_sec-tv1.tv_sec) + (tv2.tv_usec - tv1.tv_usec);
sum += diff;
num += 1.0;

cerr << frame.cols << "x" << frame.rows << "/" << down_scale << ", " << (sum/n
um) << endl; // 輸出找臉時間花費

float mean_cx = 0;
for( size_t i = 0; i < faces.size(); ++i ) {
    float cx = faces[i].x + faces[i].width*0.5;
    float cy = faces[i].y + faces[i].height*0.5;
    mean_cx += cx;
    Point center( cx, cy );
    ellipse( frame, center, Size( faces[i].width*0.5, faces[i].height*0.5 ), 0,
0, 360, Scalar( 255, 0, 255 ), 4, 8, 0 );
}
if (faces.size()>0) {
    mean_cx /= (float) faces.size();
    Point p0( mean_cx, 0.0 );
    Point p1( mean_cx, (float)frame.rows );
    line( frame, p0, p1, Scalar( 255, 0, 0 ), 5, CV_AA );
}

//-- Show what you got
imshow( window_name, frame );
return (faces.size()>0) ? mean_cx - 0.5 * frame.cols : 0; // 輸出偏移量
}

```

## 編譯

```

g++ `pkg-config --cflags opencv` detect_face_lbp.cpp -o detect_face_lbp `pkg-confi
g --libs opencv` -O2

```

或是直接使用與此文件同目錄下的 `Makefile`

```
make -j2
```

## 與 Lab.9 提供基於 Haar 分類器做速度上的比較

解析度 : 640x480

Downscale : 0.666 (1/1.5)

也就是說，用 426x320 解析度來跑分類器

跑 Haar (Lab.9) 分類器：

640x480/1.5, 330654  
640x480/1.5, 328745  
640x480/1.5, 326953  
640x480/1.5, 325266  
640x480/1.5, 323690  
640x480/1.5, 322196  
640x480/1.5, 320797  
640x480/1.5, 319593  
640x480/1.5, 318520  
640x480/1.5, 317450  
640x480/1.5, 316392  
640x480/1.5, 315550  
640x480/1.5, 314701  
640x480/1.5, 313956  
640x480/1.5, 313217  
640x480/1.5, 312506  
640x480/1.5, 312281  
640x480/1.5, 315168  
640x480/1.5, 317979  
640x480/1.5, 320672

跑 LBP 分類器：

640x480/1.5, 197560  
640x480/1.5, 198446  
640x480/1.5, 199381  
640x480/1.5, 200291  
640x480/1.5, 201190  
640x480/1.5, 202025  
640x480/1.5, 202829  
640x480/1.5, 203630  
640x480/1.5, 204412  
640x480/1.5, 205183  
640x480/1.5, 205007  
640x480/1.5, 204414  
640x480/1.5, 203839  
640x480/1.5, 203274  
640x480/1.5, 202727  
640x480/1.5, 202192  
640x480/1.5, 201668  
640x480/1.5, 201149  
640x480/1.5, 200638  
640x480/1.5, 200137  
640x480/1.5, 199643

左側的數字是解析度與 Downscale 比例，右邊的數字是跑 `detectMultiScale` 這個方法，經過的百萬秒 (us)

可以看出

LBP 大約都是 `200000 us` (大約 0.2 秒)

Haar 大約 `320000 us` (大約 0.3 秒)

在相同的設定下，並且兩支程式編譯時都使用 `-O2` 最佳化

LBP 執行速度大概是 Haar 分類器 `1.6` 倍左右

## 執行結果

可見我 Google Drive 上的 [影片](#)

[https://drive.google.com/open?id=1Ktj3Ha636DaUKMvTEIEQ4uxX0j\\_EmhSj](https://drive.google.com/open?id=1Ktj3Ha636DaUKMvTEIEQ4uxX0j_EmhSj)