1. 背景知識

1.1. 影片

影片是由一連串連續播放的靜態影像所組成的，當這些影像以足夠快的速度播放時，由於人類視覺系統的特性，我們的大腦會將這些單獨的影像合成為流暢的運動，使得靜態影像好像真的動起來一樣。

組成影片的每張靜態影像稱為一幀（Frame），我們可以透過以下程式碼提取出影片中的每一幀，同學們可以自己試試看：

```cpp
#include <opencv2/opencv.hpp>

int main() {
    cv::VideoCapture cap("video.mp4");          // 替換成你的影片路徑
    cv::Mat frame;
    while (cap.read(frame) && !frame.empty()) { // 讀取一幀，如果失敗或影片結束則跳出循環
        cv::imshow("frame", frame);             // 顯示出該幀
        if (cv::waitKey(100) == 'q')
            break;                              // 如果按下 'q' 則結束循環
    }
}
```

程式碼連結

而幀率（Frame Rate）則是指每秒鐘播放多少幀，以 FPS（Frames Per Second，每秒幀數）為單位。幀率越高影片看起來會越流暢，延遲感也會降低。普通影片使用 24 到 30 FPS 的幀率，因為這樣就足以讓大多數人覺得畫面流暢，但在電競遊戲中，由於每一毫秒都可能影響勝負，遊戲畫面的幀率可能高達上百 FPS。

1.2. 物件偵測（Object Detection）

物件偵測是一種電腦視覺技術，它讓電腦能夠識別和定位出影像中的物體，並標出它們的位置，如圖 1。這項技術通常依賴於機器學習或深度學習模型，這些模型會透過大量的數據學習物體的外觀和特徵。當模型訓練完成後，它可以用來分析新的影像，識別並定位其中的物體。

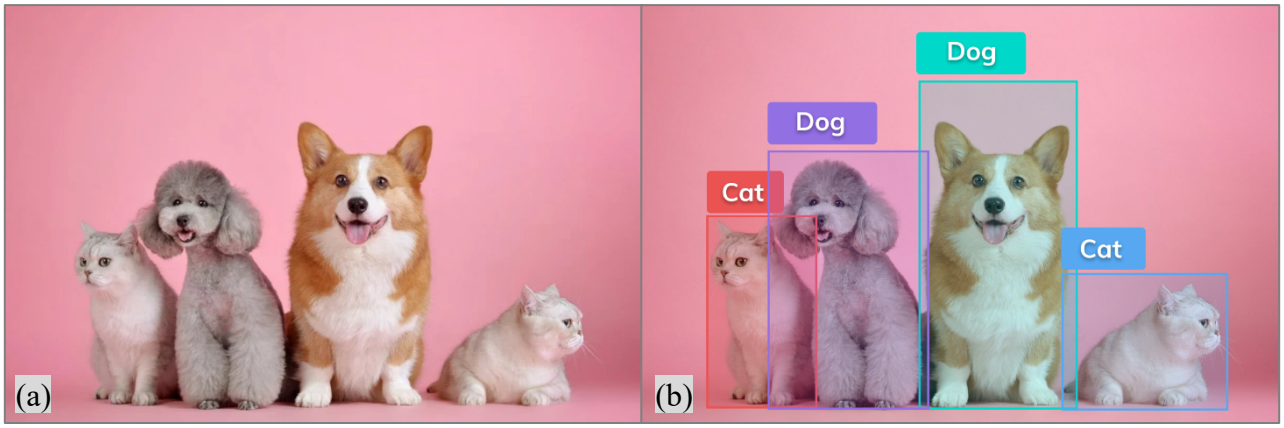


圖 1 物件偵測出貓和狗的位置 (a) 原圖；(b) 偵測結果

1.3. 物件追蹤（Object Tracking）

物件追蹤是一種電腦視覺技術，它可以讓電腦在影片中持續追蹤特定物體的位置和運動，例如在網球比賽中追蹤網球以判斷是否出界（https://www.youtube.com/watch?v=KCqN2Z0VUqA），或者在道路監視器影像中追蹤車輛以分析車流量（https://www.youtube.com/watch?v=Avpce9ouYJQ）。

物件追蹤的實現通常包含兩個步驟：

(1) 物件偵測：目的是在某幀影像中識別出要追蹤的物件，在網球比賽中我們會想辨識出球、在路口監視器影像中我們會想辨識出車輛。

(2) 追蹤器（Tracker）：目的是判斷前後兩幀偵測到的物件是否相同，如果相同代表物件移動了，如果不同則代表這是新出現的物件。

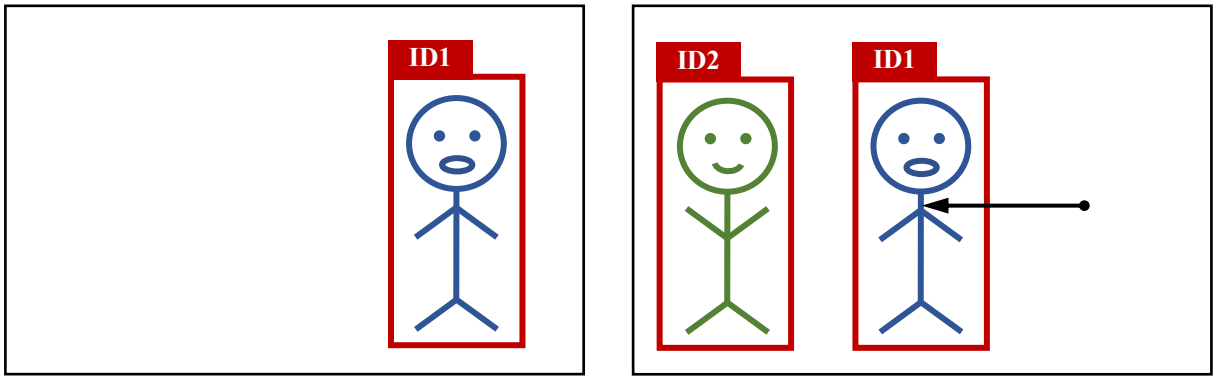


圖 2 物件追蹤範例 Frame t 與 Frame t+1 為影片的連續兩幀。Frame t 中偵測到一個藍色小人，由於是新出現的物件，追蹤器給予它 ID1 的編號；在下一幀 Frame t+1 中，同樣有偵測到一個藍色小人，追蹤器判斷出與前一幀 ID1 的藍色小人相同，可得出 ID1 物件的移動軌跡（黑色箭頭），此

外追蹤器也會發現多偵測到一個未曾出現過的綠色小人，給予它 ID2 的編號，開始追蹤綠色小人。

1.4. 質心追蹤演算法（Centroid Tracking Algorithm）

質心追蹤演算法是最容易實現的物件追蹤演算法，它的核心想法是「以質心的位移大小判斷是否為同一物體」，其大致可描述為以下步驟，實例如圖 3：

(1) 讀取一幀，透過物件偵測得到物件的邊界框（Bounding Box）。

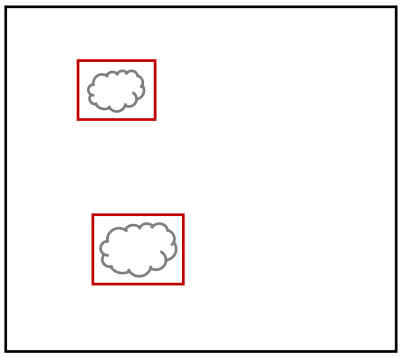
(2) 計算每個物件的質心，也就是邊界框的中心點。

(3) 將當前幀中的質心與前一幀中的質心進行配對，若距離小於閾值就視為是同一個物件。

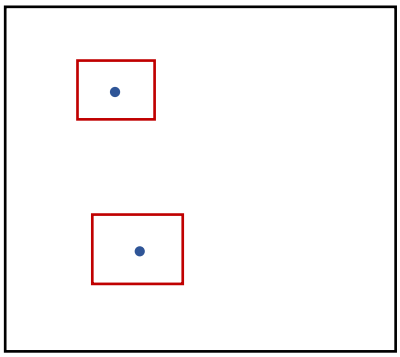
(4) 對於新的物件（前一幀不存在，這幀才存在），給予一個識別 ID，然後加入追蹤名單並記錄下目前位置；對於舊的物件（前一幀存在，這幀也存在），在追蹤名單上更新該 ID 的目前位置並記錄軌跡；對於消失的物件（前一幀存在，這幀不存在），將該 ID 從追蹤名單上剔除。

(5) 回到步驟 (1) 繼續讀取下一幀。

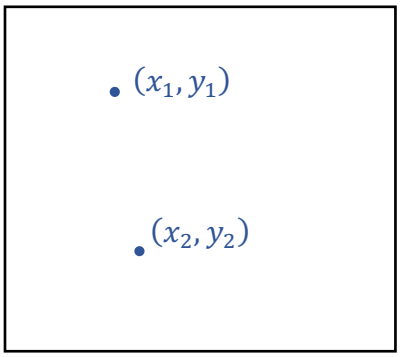
[Frame 1]



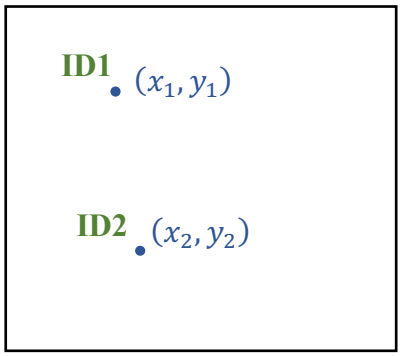
Frame 1 (1)



Frame 1 (2)
（紅框表邊界框，藍點表質心）



Frame 1 (3)

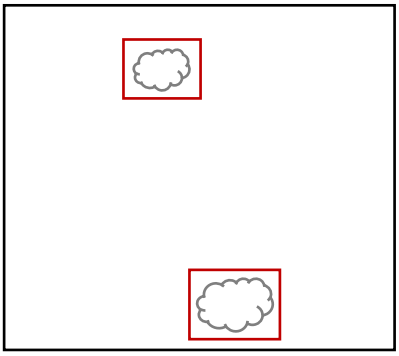


Frame 1 (4)

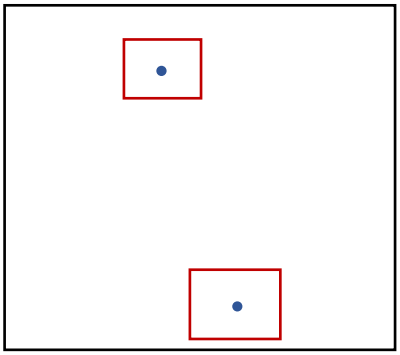
追蹤名單

ID 編號	軌跡
ID1	(x_1, y_1)
ID2	(x_2, y_2)

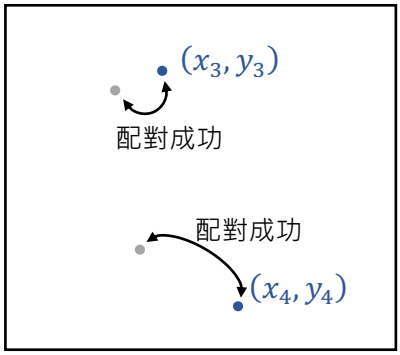
[Frame 2]



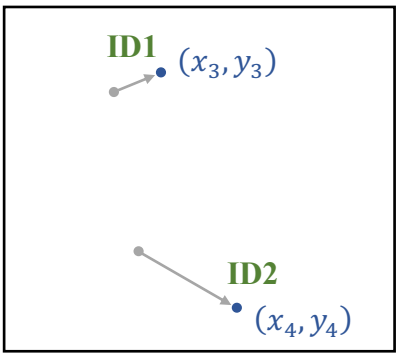
Frame 2 (1)



Frame 2 (2)



Frame 2 (3)
(灰點表前一幀的質心)

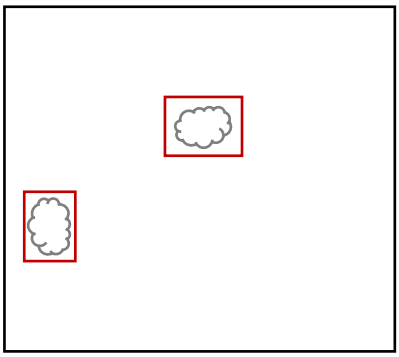


Frame 2 (4)

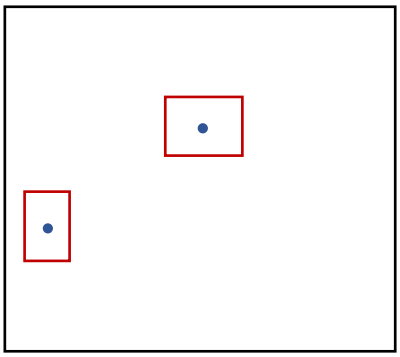
追蹤名單

ID 編號	軌跡
ID1	$(x_3, y_3) \leftarrow (x_1, y_1)$
ID2	$(x_4, y_4) \leftarrow (x_2, y_2)$

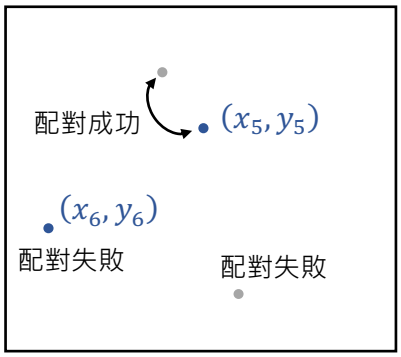
[Frame 3]



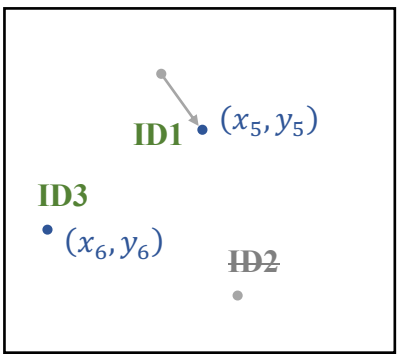
Frame 3 (1)



Frame 3 (2)



Frame 3 (3)
(配對失敗是因為距離過遠)



Frame 3 (4)

追蹤名單

ID 編號	軌跡
ID1	$(x_5, y_5) \leftarrow (x_3, y_3) \leftarrow (x_1, y_1)$
~~ID2~~	~~$(x_4, y_4) \leftarrow (x_2, y_2)$~~
ID3	(x_6, y_6)

圖 3 質心追蹤演算法

值得注意的是，雖然上述演算法的第三步驟是使用距離進行配對，但實際應用此演算法時，我們可能會依情況加入額外的條件來進行更準確的配對。

2. 作業描述

本次作業只有一個部分，有兩個需要輸入的參數，皆以命令列引數的方式給定，說明如表 1。請注意，路徑本身已經含有副檔名。

表 1 命列列引數

輸入方式	位置	本文件中的名字	解釋
命令列引數	argv[1]	inputPath	input.mp4 的路徑，為輸入資料。
	argv[2]	outputPath	output.txt 的路徑，為輸出路徑。

本次作業目的是對影片中的車輛進行物件追蹤，計算車輛移動的路徑長。請讀取路徑位於 inputPath 的 input.mp4 作為輸入影片，影片格式為 MP4，**長寬不固定（勿將長寬寫死）**，影片長度 3 秒，幀率 30 FPS，總計 90 幀。影片中包含三種物件，分別是車輛、車道及車道線，如圖 4。車道及車道線的位置和顏色皆固定。車輛形狀統一為矩形，車輛顏色的 RGB 三個值皆介於 100 至 255。車輛的顏色在單一影片中不會重複，車輛之間也不會重疊。不需考慮車輛暫時消失後又出現的情況。

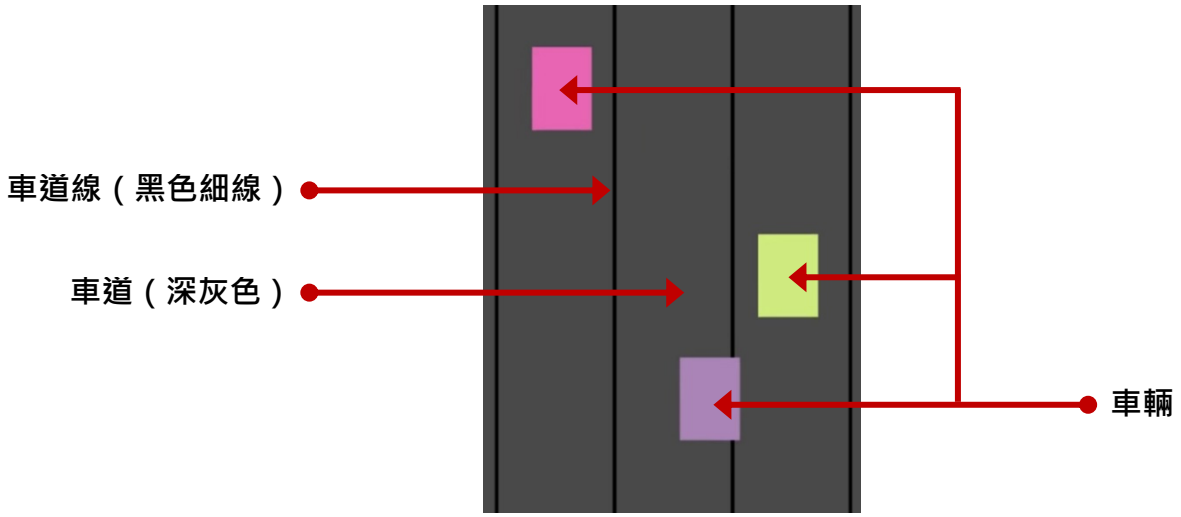


圖 4 影片中會出現的三種物件

請計算每一台車在可視範圍內的移動路徑長。路徑長以像素為單位，若計算結果是小數，請四捨五入至小數點後二位。然後按指定格式在 outputPath 的 output.txt 輸出車輛顏色及路徑長。輸出的每一行代表一輛車，在一行中請先輸出車輛顏色的十六進位色碼（最前端需加字元 '#'），接著間格一個空白後輸出路徑長，然後換行。車輛的輸出順序是根據其十六進位色碼的 ASCII 值來決定的，

請由左到右比較每個字元的 ASCII 值，值較小的色碼所對應的車輛優先輸出。以 #907DAC、#F08476 跟 #A984B5 為例，順序應為 #907DAC、#A984B5 再來是 #F08476。實際輸出範例如圖 5。**車輛顏色及路徑長皆設有容錯範圍**。

※ 十六進位色碼：十六進位色碼是一種常用於網頁設計的顏色代碼格式，它將 RGB 三個顏色的值轉換成十六進位數字，然後串接在一起。這種格式通常以一個井號（#）開頭，後面跟著六個十六進位數字，前兩個代表紅色，接下來兩個代表綠色，最後兩個代表藍色。

※ 顏色誤差：將影片儲存為 MP4 格式時，其壓縮方法會使得顏色有些許誤差。同一車輛在不同幀中，**RGB 的值可能會出現 ±3 以內的顏色誤差**，擇一輸出即可，批改系統都會認為是對的。

```
#907DAC 759.20
#A984B5 243.00
#F08476 660.64
```

圖 5 輸出範例 紅字為車輛的色碼，藍字為車輛的路徑長

為了達到作業的目的，請分為以下四步驟撰寫程式：

(1) 重採樣（Resampling）

在影像處理領域，Resampling 是一項重要的技術，尤其在需要調整影片幀率以減少計算需求時。本次作業讀取的影片長度為 3 秒，幀率為 30 FPS，代表影片總共有 90 幀。為了降低計算量，我們先對影片進行 Resampling，**只留下奇數幀**，將總幀數從 90 減至 45。以下是範例程式碼：

```cpp
#include <opencv2/opencv.hpp>
#include <vector>
using namespace std;
using namespace cv;

vector<Mat> videoResample(const vector<Mat>& clip) {
    vector<Mat> resampledClip;                  // 用來儲存 Resampling 後的影片
    for (int i = 0; i < clip.size(); i += 2)    // 遍歷 clip，每兩幀取一幀
        resampledClip.push_back(clip[i]);
    return resampledClip;
}
```

程式碼連結

(2) 定位車輛

由於車輛必定是淺色的，我們可以直接對影片進行二值化以過濾出車輛，接著再尋找矩形以進行定位，如圖 6。**當車輛的矩形面積大於 1000 個像素，才判定車輛為出現**，開始進行追蹤及提取顏色。

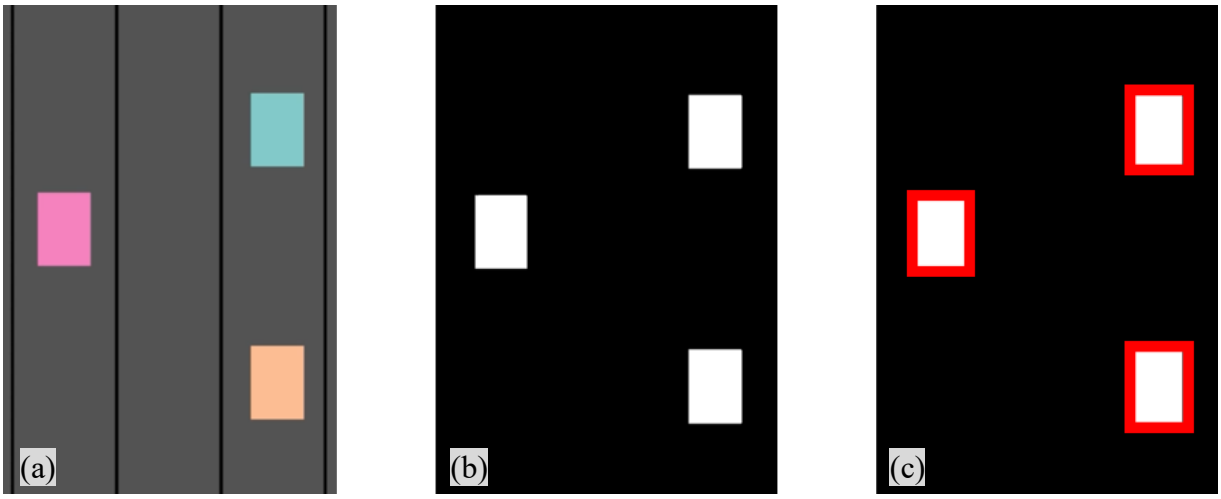


圖 6 定位車輛 (a) 原圖；(b) 透過閾值過濾出車輛；(c) 尋找出的矩形（紅框）

(3) 計算質心、顏色及車頭坐標

我們以上個步驟尋找出的矩形作為邊界框，求出邊界框的中心點作為質心。接著記錄下質心坐標，並提取此點的顏色作為車輛顏色，同時也記錄下車頭的坐標，如圖 7。

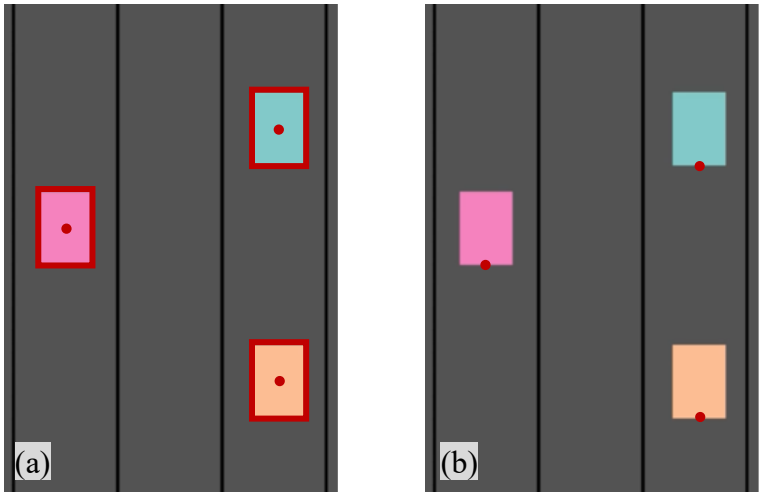


圖 7 計算質心、顏色及車頭坐標 (a) 以邊界框（紅框）的中心點作為質心（紅點），提取質心點的顏色作為車輛顏色；(b) 求出車頭位置（紅點）

(4) 進行物件追蹤，計算路徑

我們可以根據本題的情況，修改質心追蹤演算法讓其更準確（紅字為修改處），示意圖如圖 8：

a. 讀取一幀，透過以上步驟 (2) 得到車輛的邊界框(Bounding Box)，捨棄面積小於 1000 像素者。

b. 計算每輛車的質心、顏色及車頭坐標。

c. 將當前幀中的質心與前一幀中的質心進行配對，若**距離小於車的寬度**且 **RGB 三通道的差距都在 ±3 內**就視為是同一輛車。

d. 對於新的車輛 (前一幀不存在，這幀才存在)，**以色碼作為 ID**，加入追蹤名單並記錄下目前位置；對於舊的車輛 (前一幀存在，這幀也存在)，在追蹤名單中更新該 ID 目前的質心位置，並記錄車頭的軌跡；對於消失的物件 (前一幀存在，這幀不存在)，將該 ID 從追蹤名單剔除。

e. 回到步驟 (1) 繼續讀取下一幀。

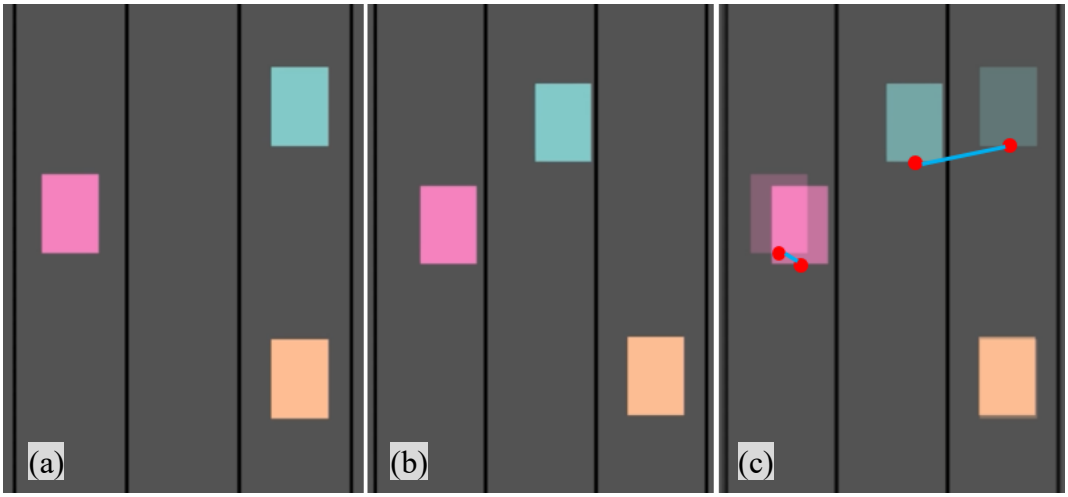


圖 8 對車輛進行物件追蹤 (a) Frame t；(b) Frame t+1；(c) 若距離夠小且顏色相近，就視為同一輛車

3. 輸入與輸出

程式執行一次只要處理一筆測資，需要接收兩個命令列引數。本次作業的輸出檔案之行數會依出現的車輛數而定，請先輸出車輛的顏色，隨後用一個空白隔開，再輸出路徑長並換行。車輛輸出的順序依照十六進位色碼及前述規則決定。

4. 評分標準

本次作業共有 10 筆測資，皆為隱藏測資。每筆測資佔 10 分，滿分 100 分。色碼及路徑長皆設有容錯範圍。使用線上批改系統自動批改。

【BONUS】前 10% 繳交作業且在批改系統達 90 分者，最後會再額外加 10 分。

【BONUS】使用 YOLOv8 對車輛進行物件追蹤，可額外加 40 分，請參考本文件最後一點。

（本次作業沒有「攻擊測資加分」及「執行速度前 10% 的加分」）

作業程式碼將進行相似度比對，**對於相似度較高的程式，我們會現場 Demo 確保不是抄襲。若有抄襲或作弊的情況，一律視為 0 分。**

5. 線上批改系統與環境

請將程式碼上傳至老師的線上批改系統 http://dslab.csie.org/course/1121LA/。執行環境如下表。本作業限定使用 C++ 撰寫，且不提供 OpenCV 以外的第三方函式庫。

作業系統	**Ubuntu 22.04**
編譯器	**g++ 11.4.0**
OpenCV 版本	**opencv 4.5.3**
OpenCV contrib 版本	**opencv_contrib 4.5.3**

6. 繳交期限

2024/1/16 23:59。

7. 附註

若有成績的相關疑慮，請回報至 yzu1607a@gmail.com（標題：[線性代數] sXXXXXXX 作業四問題）或於 Discord 伺服器問答區發問。為了保持公平性，我們不協助 Debug，只會確認是否為批改系統系統錯誤。

8. 使用 YOLOv8 對車輛進行物件追蹤（加分）

8.1. 目的

當前，物件偵測和追蹤主要依賴深度學習技術，而 YOLOv8 是其中最受歡迎的物件偵測模型。它不僅功能齊全、使用簡單，而且在物件追蹤方面也提供了多種內建的追蹤器。請查閱網上的程式碼，

利用 YOLOv8 對本次作業中的影片進行車輛物件追蹤，運用深度學習技術來解決這一問題。

8.2. 實作細節

以下為實作的各方面規範：

(1) 標註：請以手動標註或寫程式自動標註的方式，產生 YOLOv8 要求格式的標註檔。

(2) 資料集：以 public1.mp4 中的影像作為訓練集（Training Dataset）；public2.mp4 中的影像作為驗證集（Validation Dataset）；public3.mp4 中的影像作為測試集（Test Dataset）。

(3) 環境：限定使用 Google Colab。

(4) 模型：YOLOv8n。

8.3. 上傳內容

在實作完成後，將以下四項上傳至 Portal，最高可獲得 40 分，若不完整會酌量扣分：

(1) 若是手動標註，請上傳整組資料集；若是透過程式自動標註，請上傳程式碼。

(2) 在 Colab 上訓練及預測完後，下載 ipynb 檔（裡面會有輸出紀錄），然後上傳至 portal。

(3) 上傳經 YOLOv8 物件追蹤後的 public3.mp4 影片。

(4) 上傳包含以下內容的書面報告：

a、 寫出在 Colab 訓練時使用的 GPU 型號，並附上截圖。

b、 解釋 YOLOv8 的標註格式，也就是標註檔中各個數字的意義。

c、 需寫出如何下訓練的指令，並說明指令中每個參數所代表的意義。

8.4. 提醒

(1) 標註檔會納入比對，請自己以手動或程式標註資料，不要與同學共用。

(2) 物件偵測及物件追蹤可以直接使用網路上的程式碼。

(3) 準確率不是重點，不會影響得分。