中華民國第 59 屆中小學科學展覽會

作品說明書

國中組　生活與應用科學(一)科

佳作

032805

火車行駛安全辨識系統

學校名稱：雲林縣私立揚子高級中學(附設國中)

作者：	指導老師：
國二 張宸珣	陳尚民
國二 王閔溱	王彥鈞
國二 吳尚原	

關鍵詞：影像辨識、OpenCV、YOLO

摘要

本實驗為實現運用於火車的安全辨識系統，製作一套即時影像辨識，且使用了OpenCV(Open Source Computer Vision Library)以及YOLO(You only look once)等開源程式碼，並運用 Python 進行編譯和整合的工作，再進一步使用 Tensorfolw 和 Darknet 的類神經網路來訓練機器，增強系統精確性，最後，將偵測到的資訊文字化後輸出，供給訊息給其他功能使用。

壹、研究動機

「火車撞人事件，三天兩頭就會發生。根據台鐵統計，近五年來，火車事故造成的死傷多達 1055 人，其中，有六成是因為民眾穿越軌道或闖越平交道。」

若是當局能有一種提前警示系統，來更加有效的預警駕駛和控制中心可能發生的事故，是不是就能降低事故的發生率呢？於是便與老師以這個題目為出發點，設計一套系統來達成此目標。

貳、研究目的

為了有效地顯示危險狀況，省去人員一個一個檢視螢幕的麻煩，我們打算運用影像辨識技術來快速地找出異狀，提高其在監視器上顯眼程度，並藉由機器學習來提高辨識的精確度，區分出物體是什麼？構不構成威脅？記錄下事態異常的時間點，及時找出任何可能的破壞或危險，在第一時間排除異常，提高整體效率。

1. 為系統設置好安裝地點的影像資料，訓練監視器的靈敏度和精確性
2. 利用影像辨識來篩選可能構成威脅的物體
3. 規劃出安全區、危險區，相對於月台和鐵道，判斷物體在哪個位置來區分危險性
4. 設置警示系統，如警示燈、蜂鳴器
5. 以不同的危險程度發出相對應的警示

圖 1、在鐵軌和道路上區分出危險區和安全區

圖 2、在月台的黃線後分出危險區

參、實驗設備

一、使用設備

表(一)設備

名稱	數量	備註
筆記型電腦	一台	撰寫程式、紀錄實驗
平板電腦	一台	拍攝照片、錄影
Arduino 控制板	一張	控制警示器 LED 燈
紅、藍 LED 燈	各一顆	警示器的警示燈
220 歐姆電阻	兩顆	降低警示器電路電流
麵包板	一個	串接警示器線路基底
杜邦線	數條	串接警示器線路

二、使用軟體

表(二)軟體

Python 3.6	OpenCV	Darkflow	Tensorflow
Darknet	YOLO v2	Arduino UNO	Firmata

肆、研究過程

一、研究流程

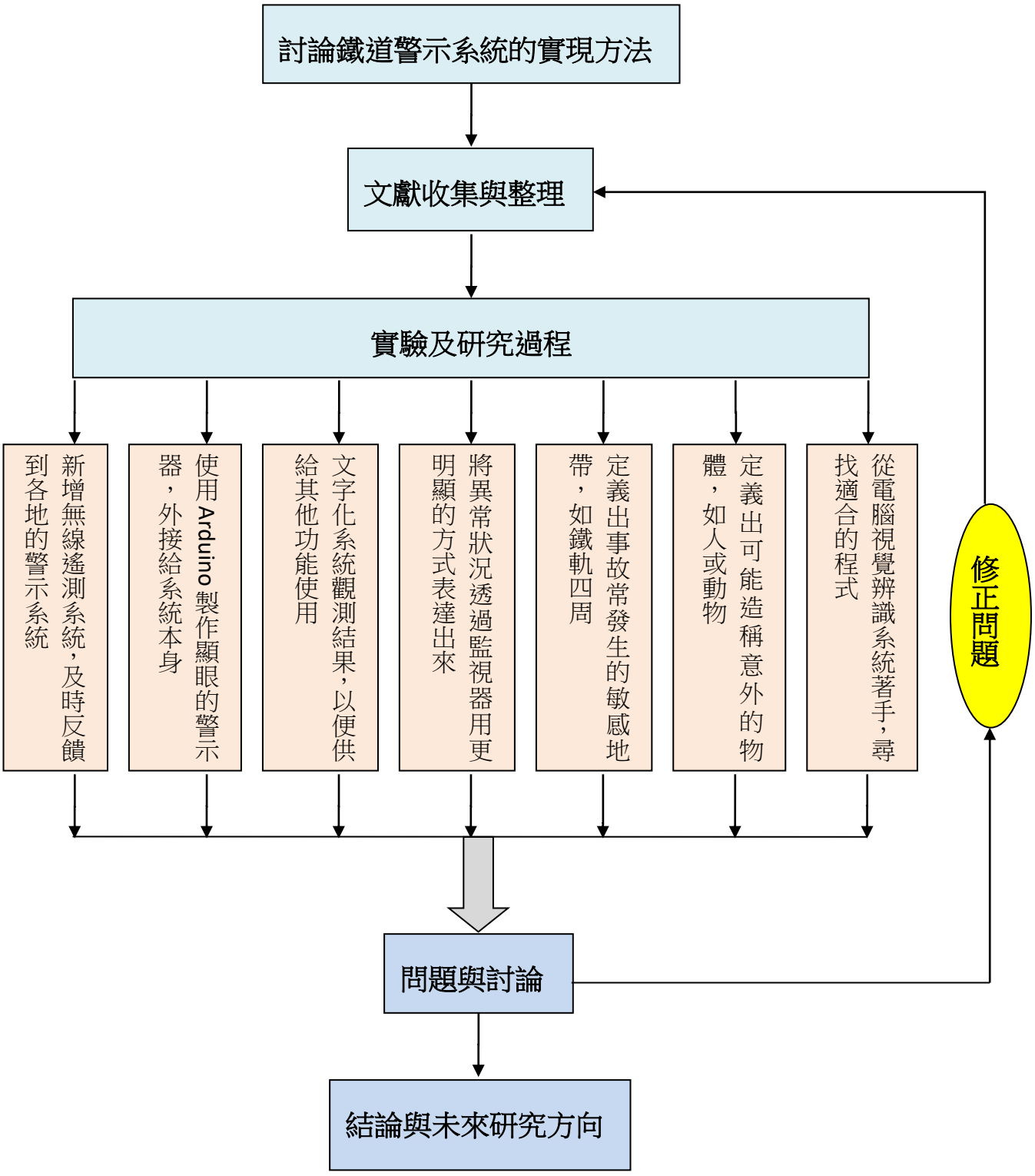


圖 3、研究流程圖

二、參考文獻

(一)OpenCV

由 Intel 所資助開發的 OpenCV 函式庫（Open Source Computer Vision Library），這套工具主要是基於 C 和 C++語言來建構出大約 300 個實用的電腦視覺中高階層 API（ Application Programming Interface ）。由於 OpenCV 的獨立性，在使用這套 API 的時候，完全不需要強制搭配額外的函式庫，使用者只需要利用自己平常使用的 C 語言或 C++語言 IDE（Integrated Development Environment），即可快速地開發影像處理相關的程式。同時，OpenCV 無論是在非商業或者是商業的使用，都是完全免費的，這更是 OpenCV 能夠普遍的被大多數開發者所廣泛使用的重要原因。

基於上述的優點，OpenCV 成為在影像分析及辨識的領域中熱門的工具之一，在 OpenCV 的函式庫中，有提供一組關於目標檢測方法的函式。主要是利用大量的樣本來進行分類器的訓練，訓練後可得到該目標所對應的分類器。用來訓練的樣本分為 positive examples 和 negative examples，其中 positive examples 就是該訓練器將需要辨識的目標，例如：人、飛機、汽車……等；negative examples 則是除了該目標外的所有任意物件。同時所有的樣本將被重新計算為相同的尺寸大小，例如：20x20，以便進行後續的影像比對程序。

分類器訓練完成後，就可以利用該分類器進行分析目標物件的程序。由於待測的原始圖片像素一定會遠大於分類器中的樣本像素，所以為了檢測整幅原始圖片，該演算法利用移動搜尋位置的方式，比對每一個位置來搜尋可能的目標；同時為了搜尋不同大小之目標，該演算法也會利用改變分類器所提供之樣本資訊的圖像大小來進行多次的掃描比對，以利找出所有可能的目標。當比對完整張待測的原始圖片後，該目標檢測的函式將會以結構的方式回傳所有可能的資訊，這時候使用者即可利用這些資訊來標記所有可能的區域，例如：畫圓圈來涵蓋該區域。利用 OpenCV 所提供的人臉特

徵分類器，使用者可以相當容易地建立起一套人臉辨識系統，同時也能設定該系統的影像來源為辨識已定義路徑之圖片或者是即時地由視訊串流來分析。

圖 4、OpenCV 人臉辨識 (此人非作者群)

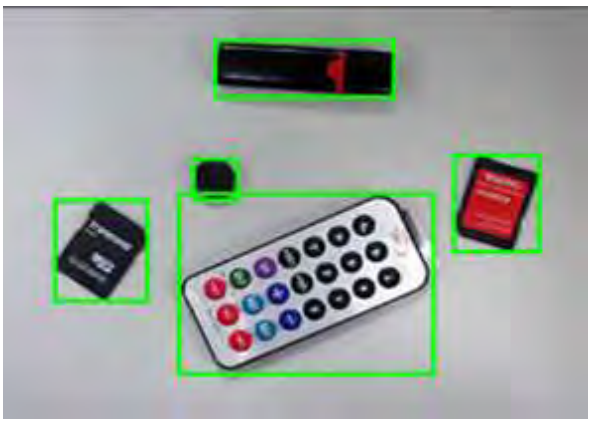


圖 5、OpenCV 多重物體辨識

(二)YOLO—You Only Look Once

YOLO 是一個完全開源的程式碼，採用單個卷積神經網絡來預測多個 bounding boxes 和類別概率。本方法相對於傳統方法有如下有優點：

1、非常快，YOLO 預測流程簡單，速度很快。基礎版在 Titan X GPU 上可以達到 45 幀/s；快速版可以達到 150 幀/s。因此，YOLO 可以實現實時檢測。

2、YOLO 採用全圖信息來進行預測，與滑動窗口方法和 region proposal-based 方法不同，YOLO 在訓練和預測過程中可以利用全圖信息。Fast R-CNN 檢測方法會錯誤的將背景中的斑塊檢測為目標，原因在於 Fast R-CNN 在檢測中無法看到全局圖像。相對於 Fast R-CNN，YOLO 背景預測錯誤率低一半。

3、YOLO 可以學習到目標的概括信息（generalizable representation），具有一定普適性。我們採用自然圖片訓練 YOLO，然後採用藝術圖像來預測。YOLO 比其它目標檢測方法（DPM 和 R-CNN）準確率高很多。

YOLO 將原本分散的 object detection 步驟融合成一個 single neural network，透過整張影像的 features 來預測每一個 bounding box，並且同時計算每個 bounding box 對於每一個 class 的機率，YOLO 不僅是從整張影像來偵測物體並且 end-to-end 訓練與運算以及可以即時運算仍維持著高精準度。

每個影像切成 SxS 的格子(grid)，如果格子中間有物體則該格子會負責去偵測該物體。每個格子又會預測 B 個 bounding boxes 與其 confidence scores，其中 conf. scores 表對應 b-box 含有物體的信心程度以及該 b-box 中的物體的精準度。

conf. score =Pr(Object) * IOU (groundtruth)

如果該 b-box 的原生 grid cell 不含有物體，則理想 conf. score 應為 0，否則理想 conf. score 的分數應和 IOU 相同。(最佳的情況是不含有物體則 Pr(Object) = 0；含有物體則 Pr(Object) = 1)。

每個 b-box 都有五個預測參數，x, y, w, h, confidence，(x, y)表示 box 中心相對於 grid cell 的位移，而 w, h 為 b-box 長寬，confidence 即為 IOU (between predicted box and any ground truth box)。

另外會將每一個 grid cell 對每個類別去計算該類別出現的機率(conditional class probabilities)，測試時是將 conditional class probabilities 乘上每個 b-box 的 conf. predictions。

Pr(Class-i|Object) * Pr(Object) * IOU = Pr(Class-i) * IOU

這樣對每個 b-box 皆會求出 class-specific conf. score。

影像切成 SxS 個 grid cell，每個 grid cell 預測 B 個 b-box 與 C class 的機率，最後的 tensor dimension 為:

tensor dinsion: S x S x (B * 5 + c) (B*5 因為 B 有 5 維)me

YOLO 在 PASCAL VOC 上使用的參數 S=7, B=2, C=20, 故 VOC final predition 為 7x7x30 tensor。

4、為提高物體定位精準性和召回率，YOLO 作者提出了 YOLO9000，提高訓練圖像的分辨率，引入了 faster rcnn 中 anchor box 的思想，對各網絡結構及各層的設計進行了改進，輸出層使用卷積層替代 YOLO 的全連接層，聯合使用 coco 物體檢測標註數據和 imagenet 物體分類標註數據訓練物體檢測模型。相比 YOLO，YOLO9000 在識別種類、精度、速度、和定位準確性等方面都有大大提升。

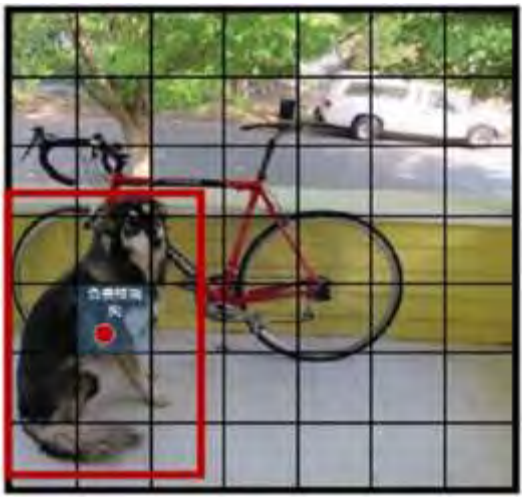


圖 6、負責辨識物體的 bounding box

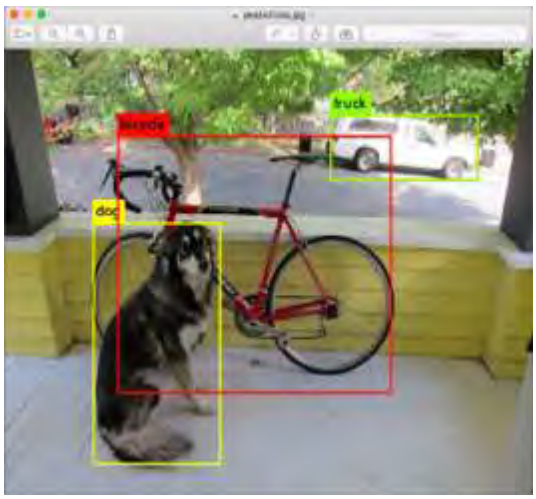


圖 7、及時辨識多重目標

三、實驗過程

(一)第一代系統

一開始，我們決定使用 Python 的 OpenCV 機器視覺擴充套件來進行即時人體辨識，為了讓 OpenCV 能夠辨識複雜的人體，我們導入了 Haar 分類器(Haar Feature-based Cascade Classifier)，它利用了幾百張圖片訓練機器辨識相同的物體，並藉由不斷的縮放、搜尋圖片，來辨識大小不一、距離不同的目標，又因為其有一層層的分類器，來一一濾掉不合條件的影像，被稱作「階層式(Cascade)」。

撰寫程式，讓電腦上的網路攝影機和程式連線，並從網路上下載已經訓練好的 Haar 分類器檔案(辨識人體)匯入撰寫好的. py 檔中，偵測到的目標則用藍框標記下來。下圖為我們當初設計的程式碼。

55.py - E:\geek\cv\55.py (3.6.5)

File Edit Format Run Options Window Help

```
import numpy as np   #匯入numpy函式庫
import cv2           #匯入opencv模組

body_cascade = cv2.CascadeClassifier('haarcascade_upperbody.xml')   #導入Haar分類器檔案(辨識身體)


cap = cv2.VideoCapture(1)   #使用攝影機鏡頭

while 1:
    ret, img = cap.read()
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    bodies = body_cascade.detectMultiScale(gray, 1.3, 5)   #使用detectMultiScale內建函數來分類出影像特徵

    for (x,y,w,h) in bodies:
        cv2.rectangle(img,(x,y),(x+w,y+h),(255,0,0),2)
        roi_gray = gray[y:y+h, x:x+w]
        roi_color = img[y:y+h, x:x+w]   #在已偵測到的人體上畫出藍色方框


    cv2.imshow('img',img)   #顯示出攝影機視窗
    k = cv2.waitKey(30) & 0xff
    if k == 27:
        break

cap.release()
cv2.destroyAllWindows()
```

圖 8、將 hear 分類器檔案匯入撰寫好的. Py

圖 9、無人在偵測範圍

圖 10、在偵測範圍內有人

在鏡頭範圍內規劃出危險區和安全區，若是人體在安全區則表示安全無恙，但如果人進入了危險區規定的座標內便發出警告，如(圖 11)。

圖 11、將鏡頭範圍內規劃出危險區和安全區

```
    for (x,y,w,h) in bodies:

        cv2.rectangle(img,(x,y),(x+w,y+h),(255,0,0),2)
        roi_gray = gray[y:y+h, x:x+w]
        roi_color = img[y:y+h, x:x+w]    #在已偵測到的人體上畫出藍色方框

        if x < 335:

            print("危險")    #當物體經過警戒線時，在Console打出危險

    cv2.imshow('img',img)    #顯示出攝影機視窗
    k = cv2.waitKey(30) & 0xff
    if k == 27:
        break

cap.release()
cv2.destroyAllWindows()
```

圖 12、在程式中定義出警戒線，當 X 座標小於 335 時，發出(危險)訊息

圖 13-1、人在安全區內

圖 13-2、Console 不顯示警告

圖 14-1、人在危險區內

圖 14-2、Console 不斷打出(危險)

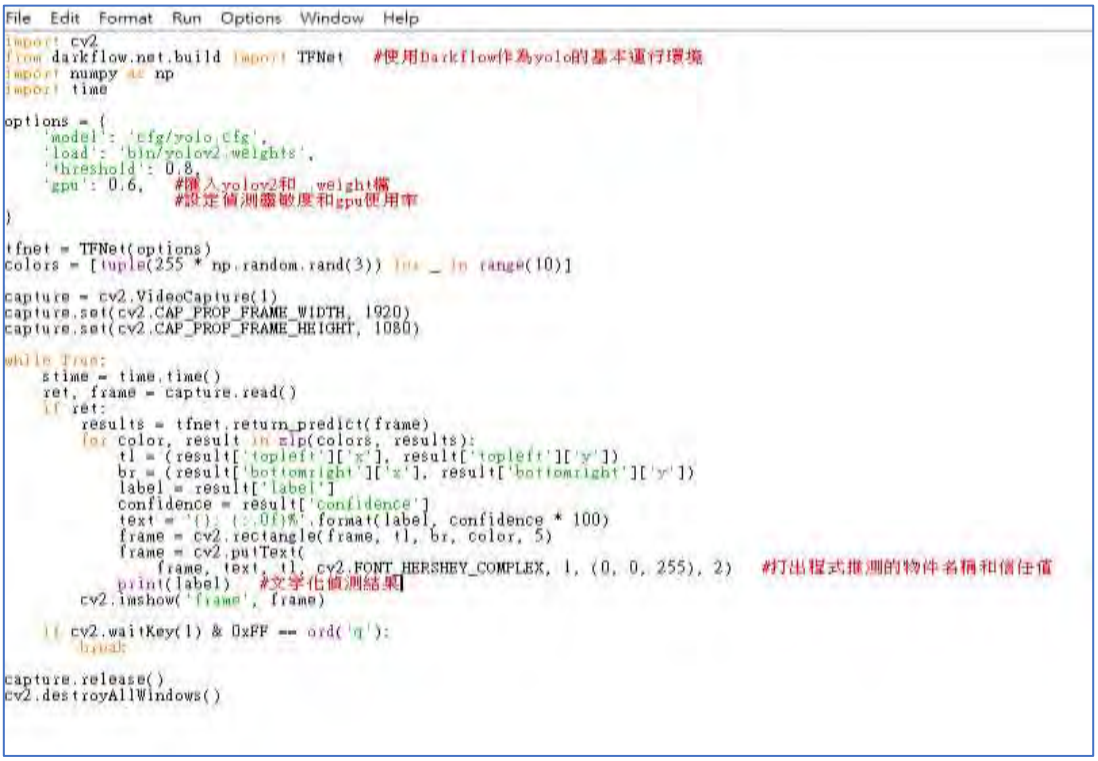
在經過多次測試後，我們得出了一個結論，雖然 OpenCV 的辨識速度和每秒張數都很高，但因為分類器的訓練樣本不足(約 400 到 500 張照片)和其本身功能較為簡單的原因，十分容易發生誤判的情況，常常將會動或者是有陰影的東西看成人，對於快速行動的目標也不夠靈敏，對於人來人往的平交道、月台，這樣的辨識能力是不合格的，因此我們需要一個能夠同時辨識多重目標，且更加準確的系統來代替 OpenCV。

(二)第二代系統

為了讓程式能有更精確和辨識多樣物體的能力，我們決定使用由 Darknet 類神經系統架構的 YOLO 即時影像辨識系統。

1.程式撰寫

由於 Darknet 只相容於 Linux 系統，使許多不支援 Linux 的攝影機無法使用，因此我們使用了 Python 的 Darkflow 模組，使 Darknet 能在 Windows 系統下運行。



```
File  Edit  Format  Run  Options  Window  Help
import cv2
from darkflow.net.build import TFNet   #使用Darkflow作為yolo的基本運行環境
import numpy as np
import time

options = {
    'model': 'cfg/yolo.cfg',
    'load': 'bin/yolov2.weights',
    'threshold': 0.8,
    'gpu': 0.6,    #導入yolov2和 weight檔
                   #設定偵測靈敏度和gpu使用率
}

tfnet = TFNet(options)
colors = [tuple(255 * np.random.rand(3)) for _ in range(10)]

capture = cv2.VideoCapture(1)
capture.set(cv2.CAP_PROP_FRAME_WIDTH, 1920)
capture.set(cv2.CAP_PROP_FRAME_HEIGHT, 1080)

while True:
    stime = time.time()
    ret, frame = capture.read()
    if ret:
        results = tfnet.return_predict(frame)
        for color, result in zip(colors, results):
            tl = (result['topleft']['x'], result['topleft']['y'])
            br = (result['bottomright']['x'], result['bottomright']['y'])
            label = result['label']
            confidence = result['confidence']
            text = '{}: {:.0f}%'.format(label, confidence * 100)
            frame = cv2.rectangle(frame, tl, br, color, 5)
            frame = cv2.putText(
                frame, text, tl, cv2.FONT_HERSHEY_COMPLEX, 1, (0, 0, 255), 2)   #打出程式推測的物件名稱和信任值
            print(label)   #文字化偵測結果
        cv2.imshow('frame', frame)

    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

capture.release()
cv2.destroyAllWindows()
```

圖 15、將 Darkflow 模組匯入 Python

將 Darkflow 模組匯入 Python 中，用 OpenCV 中原本的物體標記系統標出目標物，打出程式推測的物件名稱和信賴值，並在 Console 中打出文字化後的結果，如(圖 15)。

運用 YOLO 自帶的學習功能來訓練背景的出現的異物，將背景、火車訓練為安全正常的物體，將人、動物等等視為可能造成威脅的物體，再把軌道和月台的的間隔分為安全區和警示區，當物體和分隔好的區域重疊時，再出現不同級別的警示。

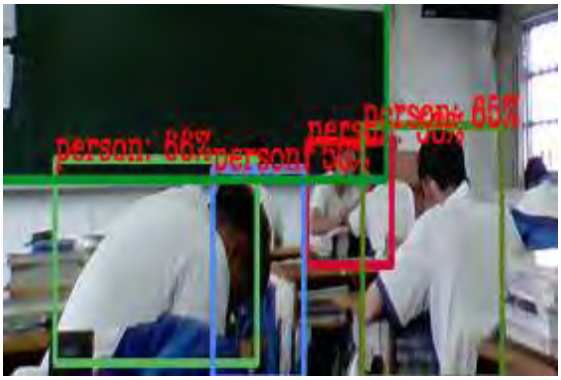


圖 16、更加靈敏的多重辨識，並準確的認出物件的種類

圖 17、即使在一段距離外也能辨識出物體(信賴值 80%以上)

四、實際應用

對自製的火車月台模型拍攝大量照片，蒐集至少 1000 張後，用 python 的 matplotlib 網格標記程式把人、火車和其他物體的方格座標標記起來，將資料存成 .xml 檔，將檔案訓練成一個 .weight 檔(類似於 Haar 分類器檔案)。

圖 18、車站月台模型圖

19、簡易火車模型

圖 20-1 遠距離火車影像

圖 20-2 近距離火車影像

圖 20-2 火車左側影像

圖 20-3 火車右側影像

圖 20、蒐集各個不同角度的火車照片

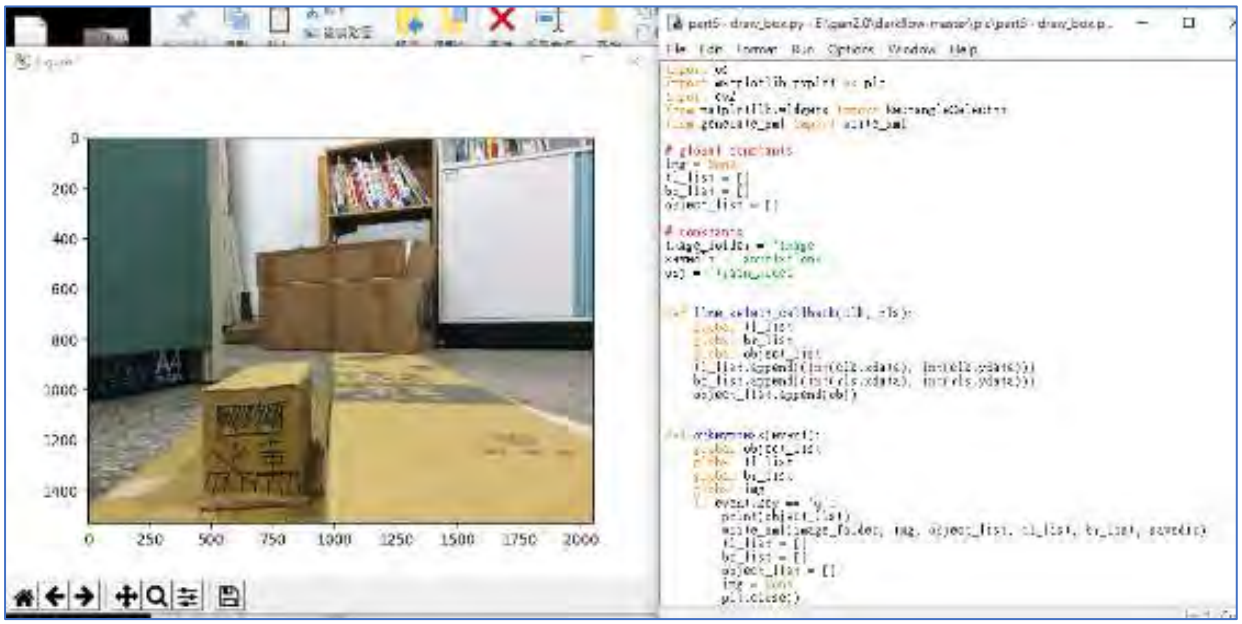


圖 21、將多張圖片匯入網格標記程式中

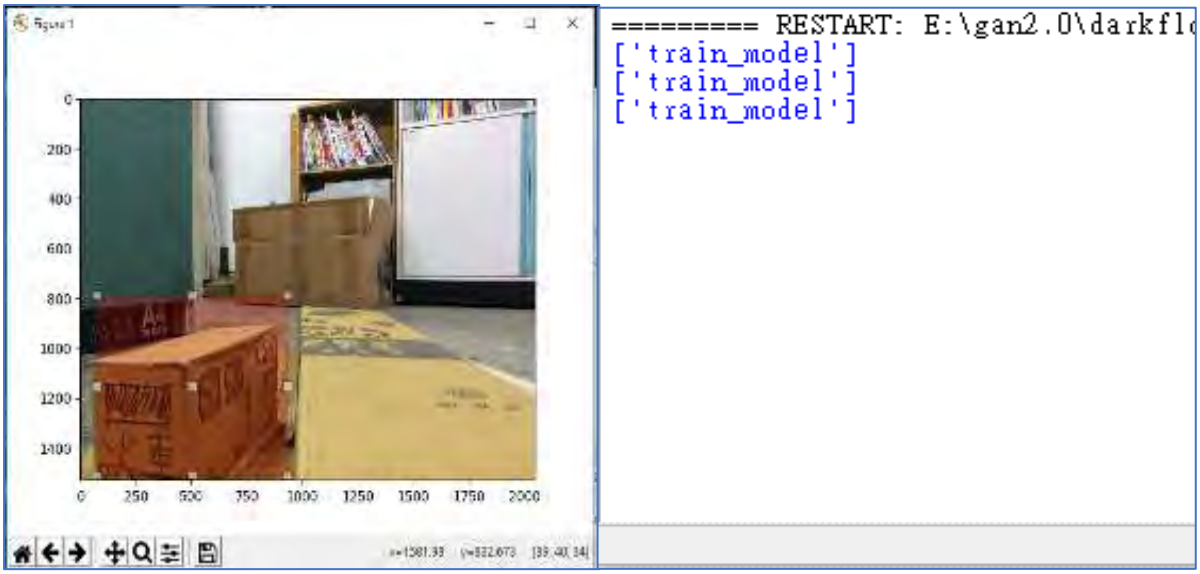


圖 22、將火車模型劃記起來後，程式記下座標(左圖)，並記下物體種類(右圖)

訓練完 .weight 檔後，將所見到的目標以鮮豔的方框標記起來，並在方框上寫出可能的物件的推測和信賴值，定義出能造成威脅的物體，例如人、動物、垃圾等，和正常的物體，如火車，將偵測到的結果文字化後輸出，在資料庫中加入經過物體的物體類別、座標、信賴值。

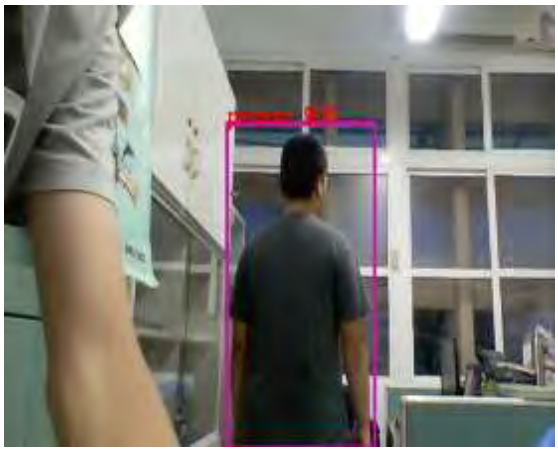


圖 23、系統偵測到的人，並在頭上顯示名稱和信賴值

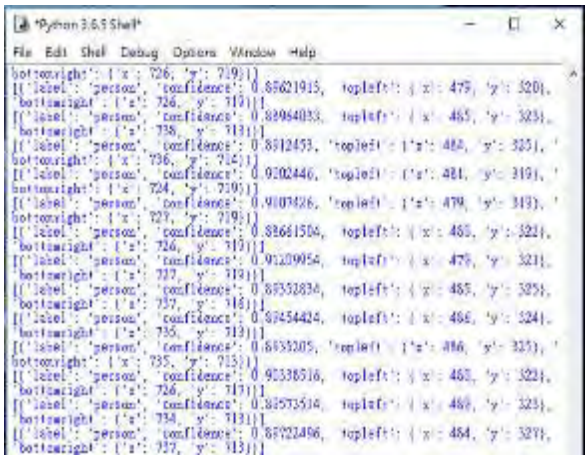


圖 24、Console 上文字化後的結果

在程式碼定義出危險區和安全區，若是已經由定義過的危險物體進入危險區，則系統發出警示，而火車進入危險區時則顯示正常，已經由設定過的物體若是出現在安全區內，不會出現任何警訊。

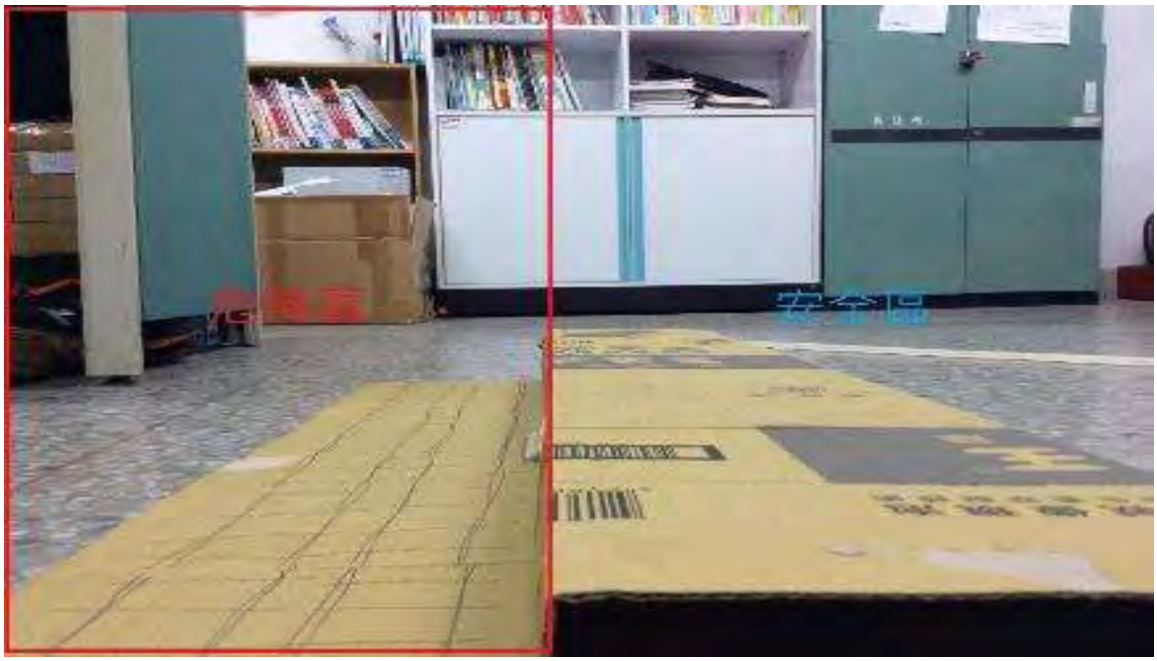


圖 25、對月台模型的危險區定義

之後加入用 Arduino 控制板製作的警示器，由紅、藍 LED 燈泡所組成，當系統偵測到危險時便開啟警示系統。

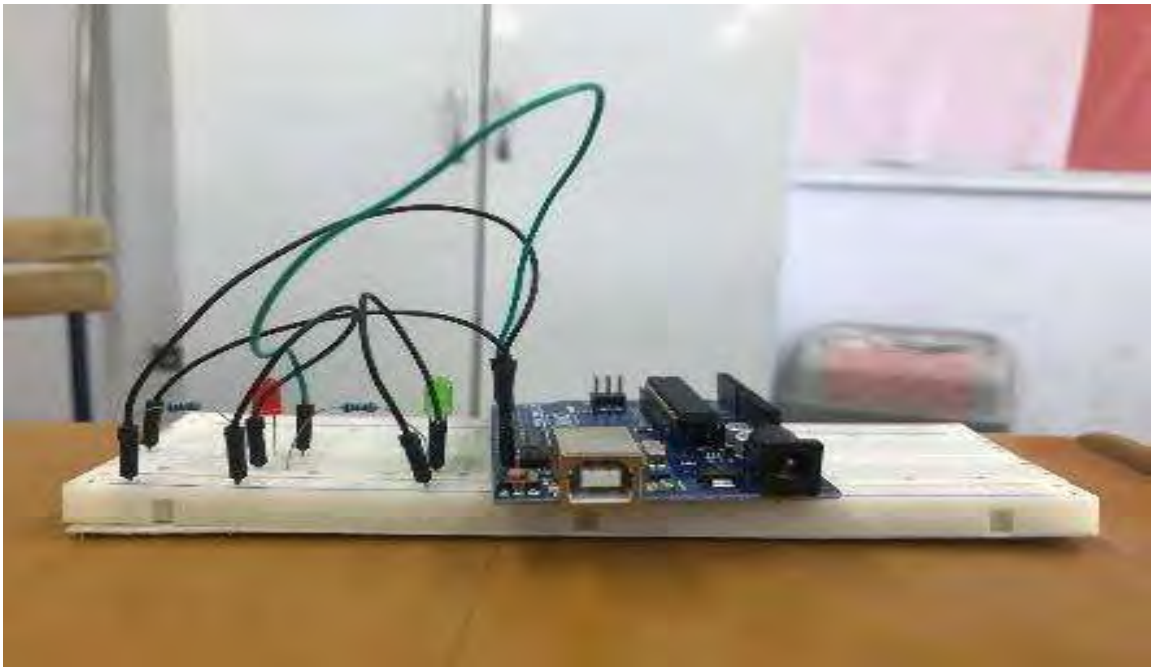


圖 26、包含了紅燈和綠燈的 Arduino 警示器

(二)第三代系統

為了讓本實驗能夠更切合實際應用，故新增 wifi 模組、普通攝影機模組和 IR Camera(紅外線攝影機)。

1. 監控節點初步處理

使用 raspberry pi 的 wifi 模組，加上一個普通攝影機模組和 IR Camera(紅外線攝影機)，補足在夜間光線不足的問題，在各個需要監控的地點設立足夠的監視器數量，將這些地點視為各個節點，而每個節點再裝設一個 Local 端處理電腦，使用較為輕便快速的模型(tiny yolo)進行初步影像辨識(辨識較為顯著且重要的目標，如人、車)，以便快速同步處理各個影像，即時反應到本地的警示系統。

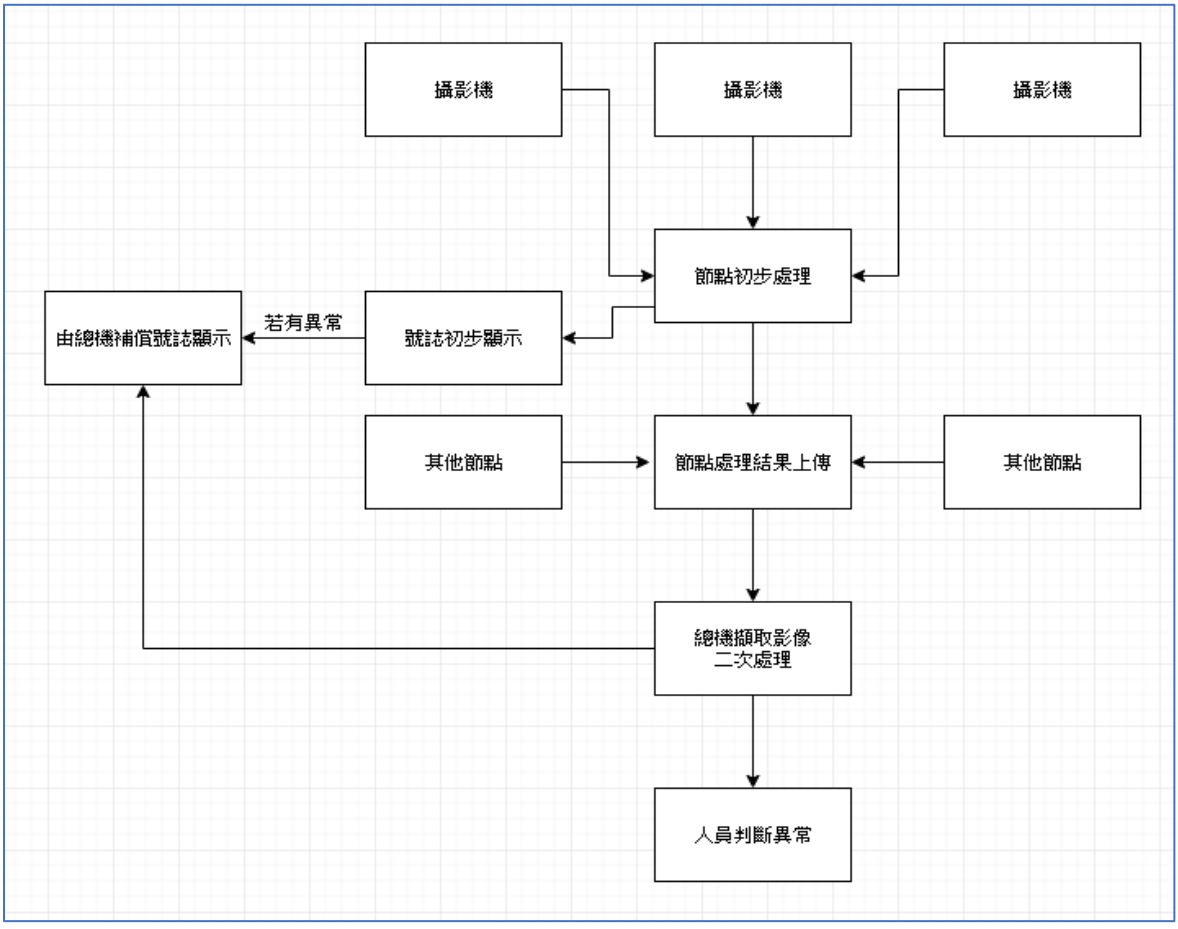


圖 27、包含了 wifi 模組、攝影機模組和 IR Camera 的 Arduino 警示器

2. 雲端二次處理

在進行初步處理的同時，將初步處理過的影像數據運用 rtsp 串流至雲端，在彙整各節點的影像後，使用總機擷取雲端的影像第二次處理，使用標準的影像辨識模型(yolov2)執行更為細緻的檢測(檢測較為次要或者是未辨識到的物體)，處理後在主控中心的螢幕顯示，讓人員更加快速地發現異常，同時將資訊回傳至節點處，若是與初步辨識的結果有落差，便讓號誌做出補償反應(如原本是安全，但實際是有異常，便升級至警戒狀態)

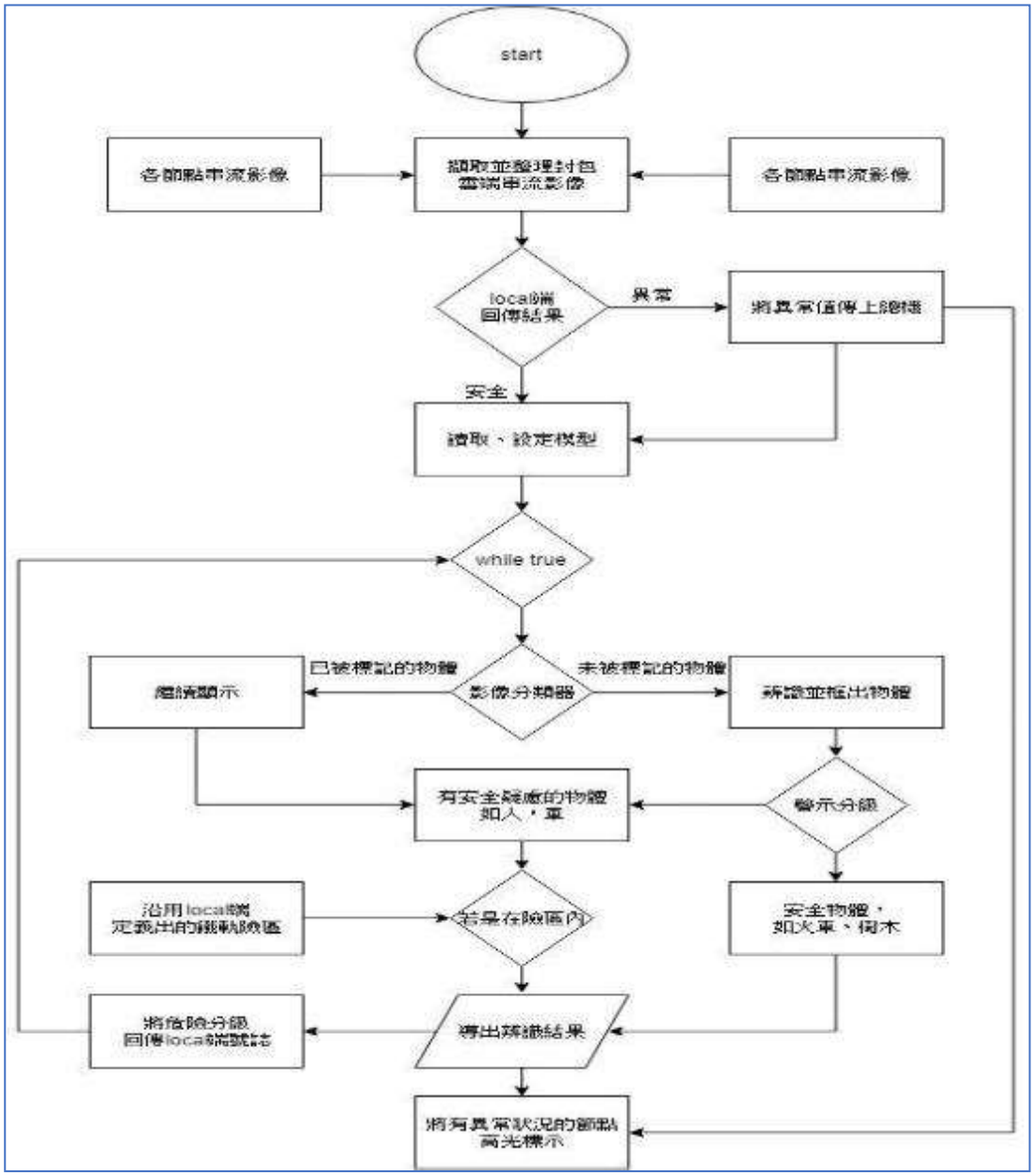


圖 28、無線遙測系統作用示意圖

3. 動態捕捉險區定義

將鐵軌的影像數據訓練至使用的模型中，使攝影機可以動態定義危險區等分界，能夠更快速方便的安裝攝影機，不須再次定義出分界，且若是因外力讓攝影機偏離定義完成的範圍，能更快速準確的校正。

圖 29、攝影機動態定義危險區與安全區

伍、研究分析結果

本研究的實驗環境為 ASUS 筆記型電腦，程式開發環境為下表:

表(三)、硬體規格

項目	規格
型號	ASUS X542U
CPU	i5-8250U
硬碟	Toshiba 128G (SSD)；日立 1TB (HDD、5400rpm)
記憶體	DDR3-1066 4GB
顯示卡	NVIDIA GeForce 940MX

表(四) 軟體開發環境

項目	規格
作業系統	Windows 10 64bit
開發平台	Python 3.6
開發語言	
Open CV	

本研究測試資料為模擬6組不同位置的攝影畫面，影片解析度為1600*900像素，影片速度平均為24FPS，長度20秒。準確率計算方式如下，其中TP代表信賴值80%以上，FP代表信賴值低於80%，FN代表遺失偵測的目標。

$$\text{Accuracy} = \frac{TP}{TP+FP+FN} \quad \text{False alarm} = \frac{FP}{TP+FP} \quad \text{Miss} = \frac{FN}{TP+FN}$$

使用Python的Pyfirmata擴充套件，與Arduino連動，使我們所製作之警示器模擬真實情況，並讀取已經有設定過的資料庫與檔案，當 YOLO 偵測系統的偵測物在安全區內時處在安全狀態，且綠燈亮起，當偵測到危險物體進入危險區時，綠燈熄滅，亮起紅燈，危險狀態解除後再亮起綠燈，熄滅紅燈，且偵測到之物體也會由 Python 系統紀錄並匯出，藉以得知在過程中有何物體進入危險區。

結合監控節點、雲端和動態險區定義的特點，這項系統不只可以運用在火車鐵軌上，也可以結合其他功能用在各個區域，如高速公路上，做為道路監控或是在行車紀錄器上監測行車死角等。

圖 28、人在安全範圍內，亮綠燈

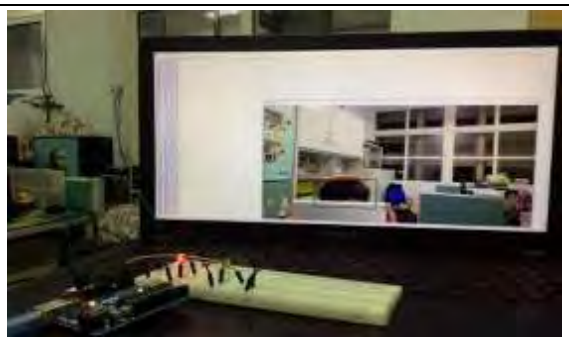


圖 29、人在危險區，綠燈熄滅，亮紅燈

(一) 流明與準確度分析結果

圖 30-1: 1500 流明準確度

圖 30-2: 1200 流明準確度

圖 30-3: 900 流明準確度

圖 30-4: 600 流明準確度

圖 30-5: 300 流明準確度

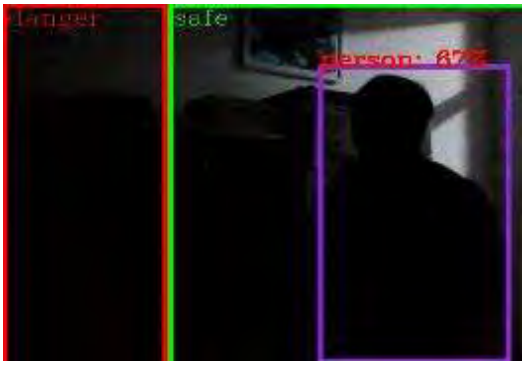


圖 30-6: 100 流明準確度

(二) 第一代在六個測試環境中的實驗結果

表(五) 第一代測試數據

	Frames	True Pos.	False Pos.	False Neg.	Miss rate	False Alarm	Accuracy
Test1	480	1640	80	104	5.96%	4.65%	89.91%
Test2	480	805	127	103	11.34%	13.63%	77.78%
Test3	480	682	101	96	12.34%	12.90%	77.59%
Test4	480	702	110	102	12.69%	13.55%	76.81%
Test5	480	1020	106	100	8.93%	9.41%	83.20%
Test6	480	1011	104	106	9.49%	9.33%	82.80%

(三) 第二代在六個測試環境中的實驗結果

表(六) 第二代測試數據

	Frames	True Pos.	False Pos.	False Neg.	Miss rate	False Alarm	Accuracy
Test1	480	1120	51	113	9.16%	4.36%	87.23%
Test2	480	677	101	96	12.42%	12.98%	77.46%
Test3	480	1011	96	83	7.59%	8.67%	84.96%
Test4	480	1300	106	103	7.34%	7.54%	86.15%
Test5	480	1020	106	100	8.93%	9.41%	83.20%
Test6	480	1011	104	106	9.49%	9.33%	82.80%

(四) 第二代在六個測試環境中的實驗結果

表(七) 第三代測試數據

	Frames	True Pos.	False Pos.	False Neg.	Miss rate	False Alarm	Accuracy
Test1	480	1220	49	92	7.01%	3.86%	89.64%
Test2	480	1077	60	101	8.57%	5.28%	87.00%
Test3	480	1061	86	90	7.82%	7.50%	85.77%
Test4	480	1210	91	93	7.14%	6.99%	86.80%
Test5	480	1001	102	94	8.58%	9.25%	83.63%
Test6	480	962	93	102	9.59%	8.82%	83.15%

陸、問題與討論

本研究設計的構思與演變與問題如下:

(一)起初剛想要運用辨識系統時，我們剛開始本來打算使用現成的辨識系統，並利用圖片辨識物體的方式來偵測某一範圍內的物品種類，但由於我們是利用照片來分析，所以對於我們所想辨識的方式有所差異，所以我們就想將其改良成能辨識影像的系統。

(二)我們最先找到的程式為 OpenCV 是因為其為時下十分熱門的辨識程式之一，它只需要具備 C 語言或 C++語言的能力即可加以編輯，也因為 OpenCV 的獨立性，讓使用者在使用時不需要插入函式庫,我們所使用的是利用 Python 並外加 OpenCV 模組進行辨識，但經過多次實驗後，我們發現雖然它具有十分高的辨識速度，不過它的程式結構較為簡單，時常發生辨識錯誤的情況，不符合本研究要求，所以繼續尋找更合適的程式。

(三)在找 OpenCV 時我們也有看到另一個十分熱門的辨識系統叫作「YOLO」，他是一款由 Darknet 所開發的程式系統，因為他在網路上已有許多人應用，所以對於我們來說也十分方便，且其系統主要為運用目前較新興的 Python 系統外加 Darkflow 模組和其他部分模組即可編輯，而我們也利用 Python 的程式將其加入了信賴值的功能，使我們可以大略知道其準確率為多少，以便降低準確率，其中，我們是利用 OpenCV 的影像讀取與畫圖結合 YOLO 的系統進行辨識，大幅提升其功能性與方便性。

(四)當我們在討論要使用其系統於火車上時，想將其架設在月台與平交道，但因兩者的辨識模式不大相同，前者是判斷在某區域內進行判定，並將訊號傳送至火車上的控制室與中央控制室使人員得知消息，並即時做出反應，以降低危險，後者則是在平交道的軌道附近架設鏡頭，並辨識在火車行徑範圍內是否有危險物，並在平交道設置燈號，如果有，就會亮起紅燈並發出警示，讓警衛能迅速解除危險，若有火車即將行經，則也會向列車控制室提出警示以降低車速，以免發生危險。

表(八) 兩系統之比較

系統	OpenCV	YOLO 與 OpenCV 整合
編輯程式	Python	Python
辨識目標速度	高	高
準確率	較低	較高
靈敏度	較不穩定	穩定
使用方便性	使用較為方便	須自行編寫部分程式碼
模組	OpenCV 機器視覺擴充套件、Haar 分類器	Darkflow 模組、OpenCV 模組與其他輔助模組
樣本數量範圍	400 到 500	1000 到 1500

(五)最後在考慮到實際應用時，若是光線不足會影響系統判讀，故加入 IR Camera(紅外線攝影機)，補足在夜間光線不足的問題，以及將每個需要監控的地點視為節點，將資料初步處理後及時回饋到本地的警示系統，將會更符合實際應用。

柒、結論

綜合以上研究，本團隊歸納出以下論點:

一、此辨識系統可運用在各種需辨識物體之地方，且其所要辨識之物體可經由設定來決定，其危險程度也可再進一步的設置，若要增加辨識區域與準確度，需搭配望遠鏡頭與高畫質感光元件使用。

二、此辨識方式比起人類自己肉眼辨識準確許多，可以有效輔助駕駛人員突發事件處理，因此可大幅度降低因失誤而造成之影響。

三、本系統所設定之危險範圍可藉由調整程式內的數值來改變，可適應各種不同的需求方式，使用起來十分便利。

四、本系統有設定信賴值，可藉由其所顯示的數值來得知其準確率，以減少其誤差，由實驗結果顯示，系統會因流明度差異而影響準確率，建議火車與軌道照明設備盡可能充足與加強，這樣對駕駛員與系統皆能提高辨識效果。

五、本系統運用於火車月台與平交道辨識時，可藉其所判斷之物體來警告列車長減速，並通知中控室以幫助列車長做出行動，以避免因人為疏失或反應時間不足所造成之意外。

六、此辨識系統結合 wifi 模組、攝影機模組和 IR Camera 的 Arduino 警示器，將會提高夜間辨識能力，與即時資料回饋，讓監控者在最短時間內做出正確的判斷。

捌、參考資料來源

1. 官峰任(2018)臺灣警政單位導入即時影像人臉辨識系統關鍵成功因素之研究。國防大學資訊管理系碩士論文
2. 林俊傑，林群倫，應用 OpenCV 完成影像辨識功能，安全管理與工程技術國際研討會，民國 102 年
3. https://opencv.org/ -OpenCV 官網
4. http://monkeycoding.com/?page_id=12 -OpenCV 阿洲的程式教學
5. https://pythonprogramming.net/haar-cascade-face-eye-detection-python-opencv-tutorial/ -OpenCVHaar cascade 及時辨識教學
6. https://pjreddie.com/darknet/yolo/ -YOLO 官網
7. https://blog.csdn.net/tangwei2014/article/details/50915317 -YOLO 論文閱讀筆記
8. https://github.com/thtrieu/darkflow -Darkflow 模組來源
9. https://matplotlib.org/ -Matplotlib 模組官網
10. https://github.com/markjay4k/YOLO-series/blob/master/part6%20-%20draw_box.py -Matplotlib 網格標記程式來源
11. https://github.com/markjay4k/YOLO-series/blob/master/part4_video.py -YOLO 攝影機連動程式來源
12. https://www.arduino.cc/ -Arduino 官網
13. https://www.arduino.cc/en/Reference/Firmata -Firmata 官網
14. http://yehnan.blogspot.tw/2016/01/arduinowindowspythonfirmataarduino.html -Firmata Arduino 連動教學

【評語】032805

1. 本作品利用 OpenCV 及 Yolo 開發即時影像辨識系統，可用於火車月台及軌道附近人員進入危險區之安全警示，可改善火車行駛之安全性，有很好的應用價值，值得嘉許。
2. 作者利用目前資訊科技的技術，也使用現有的軟體及硬體套件，讓影像判讀的準確性有明顯的提升。
3. 惟外在環境因素、圖片訓練數量等對影像判讀正確性的影響，可以更深入探討。

作品海報

火車行駛安全辨識系統

摘 要

本實驗為實現運用於火車的安全辨識系統，製作一套即時影像辨識，且使用了OpenCV(Open Source Computer Vision Library)以及YOLO(You only look once)等開源程式碼，並運用Python進行編譯和整合的工作，再進一步使用Tensorfolw和Darknet的類神經網路來訓練機器，增強系統精確性，最後，將偵測到的資訊文字化後輸出，供給訊息給其他功能使用。

壹、研究動機

「火車撞人事件，三天兩頭就會發生。根據台鐵統計，近五年來，火車事故造成的死傷多達1055人，其中，有六成是因為民眾穿越軌道或闖越平交道。」
若是當局能有一種提前警示系統，來更加有效的預警駕駛和控制中心可能發生的事故，是不是就能降低事故的發生率呢？於是便與老師以這個題目為出發點，設計一套系統來達成此目標。

貳、研究目的

為了有效地顯示危險狀況，省去人員一個一個檢視螢幕的麻煩，我們打算運用影像辨識技術來快速地找出異狀，提高其在監視器上顯眼程度，並藉由機器學習來提高辨識的精確度，區分出物體是什麼？構不構成威脅？記錄下事態異常的時間點，及時找出任何可能的破壞或危險，在第一時間排除異常，提高整體效率。

1. 為系統設置好安裝地點的影像資料，訓練監視器的靈敏度和精確性
2. 利用影像辨識來篩選可能構成威脅的物體
3. 規劃出安全區、危險區，相對於月台和鐵道，判斷物體在哪個位置來區分危險性
4. 設置警示系統，如警示燈、蜂鳴器
5. 以不同的危險程度發出相對應的警示

圖1、在鐵軌和道路上區分出危險區和安全區

圖2、在月台的黃線後分出危險區

參、實驗設備

一、使用設備

筆記型電腦	一台	撰寫程式、紀錄實驗
平板電腦	一台	拍攝照片、錄影
Arduino控制板	一張	控制警示器LED燈
紅、藍LED燈	各一顆	警示器的警示燈
220歐姆電阻	兩顆	降低警示器電路電流
麵包板	一個	串接警示器線路基底
杜邦線	數條	串接警示器線路

二、使用軟體

Python 3.6	OpenCV	Darkflow	Tensorflow
Darknet	YOLO v2	Arduino UNO	Firmata

肆、研究過程

一、研究流程

圖3、研究流程圖

二、參考文獻

(一)OpenCV

由 Intel所資助開發的OpenCV函式庫（Open Source Computer Vision Library），這套工具主要是基於C和C++語言來建構出大約300個實用的電腦視覺中高階層 API（ Application Programming Interface ）。由於OpenCV的獨立性，在使用這套 API 的時候，完全不需要強制搭配額外的函式庫，使用者只需要利用自己平常使用的 C 語言或C++語言IDE（Integrated Development Environment），即可快速地開發影像處理相關的程式。同時，OpenCV無論是在非商業或者是商業的使用，都是完全免費的，這更是OpenCV能夠普遍的被大多數開發者所廣泛使用的重要原因。

基於上述的優點，OpenCV成為在影像分析及辨識的領域中熱門的工具之一，在OpenCV的函式庫中，有提供一組關於目標檢測方法的函式。主要是利用大量的樣本來進行分類器的訓練，訓練後可得到該目標所對應的分類器。用來訓練的樣本分為 positive examples 和 negative examples，其中 positive examples 就是該訓練器將需要辨識的目標，例如：人、飛機、汽車.....等；negative examples 則是除了該目標外的所有任意物件。同時所有的樣本將被重新計算為相同的尺寸大小，例如：20x20，以便進行後續的影像比對程序。

分類器訓練完成後，就可以利用該分類器進行分析目標物件的程序。由於待測的原始圖片像素一定會遠大於分類器中的樣本像素，所以為了檢測整幅原始圖片，該演算法利用移動搜尋位置的方式，比對每一個位置來搜尋可能的目標；同時為了搜尋不同大小之目標，該演算法也會利用改變分類器所提供之樣本資訊的圖像大小來進行多次的掃描比對，以利找出所有可能的目標。當比對完整張待測的原始圖片後，該目標檢測的函式將會以結構的方式回傳所有可能的資訊，這時候使用者即可利用這些資訊來標記所有可能的區域，例如：畫圓圈來涵蓋該區域。利用OpenCV所提供的人臉特徵分類器，使用者可以相當容易地建立起一套人臉辨識系統，同時也能設定該系統的影像來源為辨識已定義路徑之圖片或者是即時地由視訊串流來分析。

圖4、OpenCV人臉辨識 (非相關作者)

圖5、OpenCV多重物體辨識

(二)YOLO—You Only Look Once

YOLO是一個完全開源的程式碼，採用單個卷積神經網絡來預測多個bounding boxes和類別概率。本方法相對於傳統方法有如下有優點：

1、非常快，YOLO預測流程簡單，速度很快。基礎版在Titan X GPU上可以達到45幀/s；快速版可以達到150幀/s。因此，YOLO可以實現實時檢測。

2、YOLO採用全圖信息來進行預測，與滑動窗口方法和region proposal-based方法不同，YOLO在訓練和預測過程中可以利用全圖信息。Fast R-CNN檢測方法會錯誤的將背景中的斑塊檢測為目標，原因在於Fast R-CNN在檢測中無法看到全局圖像。相對於Fast R-CNN，YOLO背景預測錯誤率低一半。

3、YOLO可以學習到目標的概括信息（generalizable representation），具有一定普適性。我們採用自然圖片訓練YOLO，然後採用藝術圖像來預測。YOLO比其它目標檢測方法（DPM和R-CNN）準確率高很多。

YOLO將原本分散的object detection 步驟融合成一個single neural network，透過整張影像的features來預測每一個bounding box，並且同時計算每個bounding box對於每一個class的機率，YOLO不僅是從整張影像來偵測物體並且end-to-end訓練與運算以及可以即時運算仍維持著高精準度。

每個影像切成SxS的格子(grid)，如果格子中間有物體則該格子會負責去偵測該物體。每個格子又會預測B個bounding boxes與其confidence scores，其中conf. scores表對應b-box含有物體的信心程度以及該b-box中的物體的精準度。

conf. score =Pr(Object) * IOU (groundtruth)

如果該b-box的原生grid cell不含有物體，則理想conf. score應為0，否則理想conf. score的分數應和IOU相同。(最佳的情況是不含有物體則Pr(Object) = 0；含有物體則Pr(Object) = 1)。

每個b-box都有五個預測參數，x, y, w, h, confidence，(x, y)表示box中心相對於grid cell的位移，而w, h為b-box長寬，confidence即為IOU (between predicted box and any ground truth box)。

另外會將每一個grid cell對每個類別去計算該類別出現的機率(conditional class probabilities)，測試時是將conditional class probabilities乘上每個b-box的conf. predictions。

Pr(Class-i|Object) * Pr(Object) * IOU = Pr(Class-i) * IOU
這樣對每個b-box皆會求出class-specific conf. score。

影像切成SxS個grid cell，每個grid cell預測B個b-box與C class的機率，最後的tensor dimension為:

tensor dinsion: S x S x (B * 5 + c) (B*5因為B有5維)me

YOLO在PASCAL VOC上使用的參數 S=7, B=2, C=20, 故VOC final prediction為7x7x30 tensor。

4、為提高物體定位精準性和召回率，YOLO作者提出了YOLO9000，提高訓練圖像的分辨率，引入了faster rcnn中anchor box的思想，對各網絡結構及各層的設計進行了改進，輸出層使用卷積層替代YOLO的全連接層，聯合使用coco物體檢測標註數據和imagenet物體分類標註數據訓練物體檢測模型。相比YOLO，YOLO9000在識別種類、精度、速度、和定位準確性等方面都有大大提升。

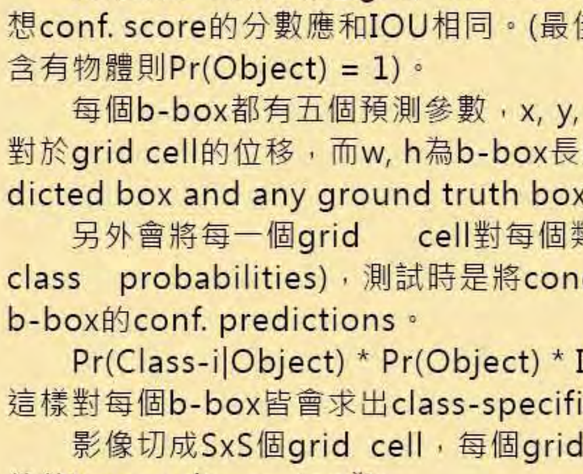


圖6、負責辨識物體的bounding box

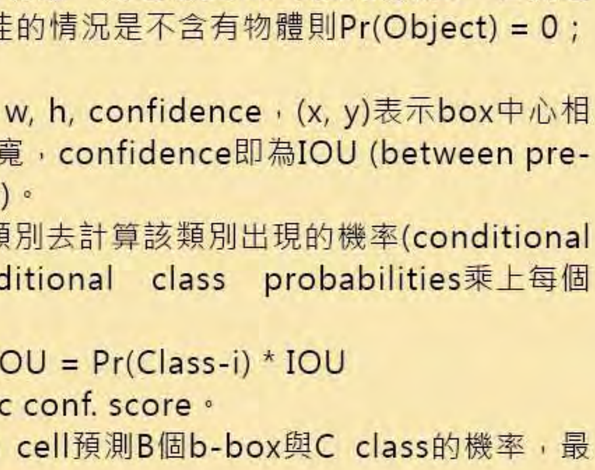


圖7、及時辨識多重目標

三、實驗過程

(一) 第一代系統

一開始，我們決定使用Python的OpenCV機器視覺擴充套件來進行即時人體辨識，為了讓OpenCV能夠辨識複雜的人體，我們導入了Haar分類器(Haar Feature-based Cascade Classifier)，它利用了幾百張圖片訓練機器辨識相同的物體，並藉由不斷的縮放、搜尋圖片，來辨識大小不一、距離不同的目標，又因為其有一層層的分類器，來一一濾掉不合條件的影像，被稱作「階層式(Cascade)」。

撰寫程式，讓電腦上的網路攝影機和程式連線，並從網路上下載已經訓練好的Haar分類器檔案(辨識人體)匯入撰寫好的. py檔中，偵測到的目標則用藍框標記下來。下圖為我們當初設計的程式碼。

55.py - E:\geek\cv\55.py (3.6.5)

File Edit Format Run Options Window Help

```
import numpy as np   #匯入numpy函式庫
import cv2           #匯入opencv模組

body_cascade = cv2.CascadeClassifier('haarcascade_upperbody.xml')   #導入Haar分類器檔案(辨識身體)


cap = cv2.VideoCapture(1)   #使用攝影機鏡頭

while 1:
    ret, img = cap.read()
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    bodies = body_cascade.detectMultiScale(gray, 1.3, 5)   #使用detectMultiScale內建函數來分類出影像特徵

    for (x,y,w,h) in bodies:
        cv2.rectangle(img,(x,y),(x+w,y+h),(255,0,0),2)
        roi_gray = gray[y:y+h, x:x+w]
        roi_color = img[y:y+h, x:x+w]   #在已偵測到的人體上畫出藍色方框


    cv2.imshow('img',img)   #顯示出攝影機視窗
    k = cv2.waitKey(30) & 0xff
    if k == 27:
        break

cap.release()
cv2.destroyAllWindows()
```

圖8、將hear分類器檔案匯入撰寫好的. Py

之後加入用Arduino控制板製作的警示器，由紅、藍LED燈泡所組成，當系統偵測到危險時便開啟警示系統。

圖26、包含了紅燈和綠燈的Arduino警示器

(二)第三代系統

為了讓本實驗能夠更切合實際應用，故新增wifi模組、普通攝影機模組和IR Camera(紅外線攝影機)。

1. 監控節點初步處理

使用raspberry pi 的wifi模組，加上一個普通攝影機模組和IR Camera(紅外線攝影機)，補足在夜間光線不足的問題，在各個需要監控的地點設立足夠的監視器數量，將這些地點視為各個節點，而每個節點再裝設一個Local端處理電腦，使用較為輕便快速的模型(tiny yolo)進行初步影像辨識(辨識較為顯著且重要的目標，如人、車)，以便快速同步處理各個影像，即時反應到本地的警示系統。

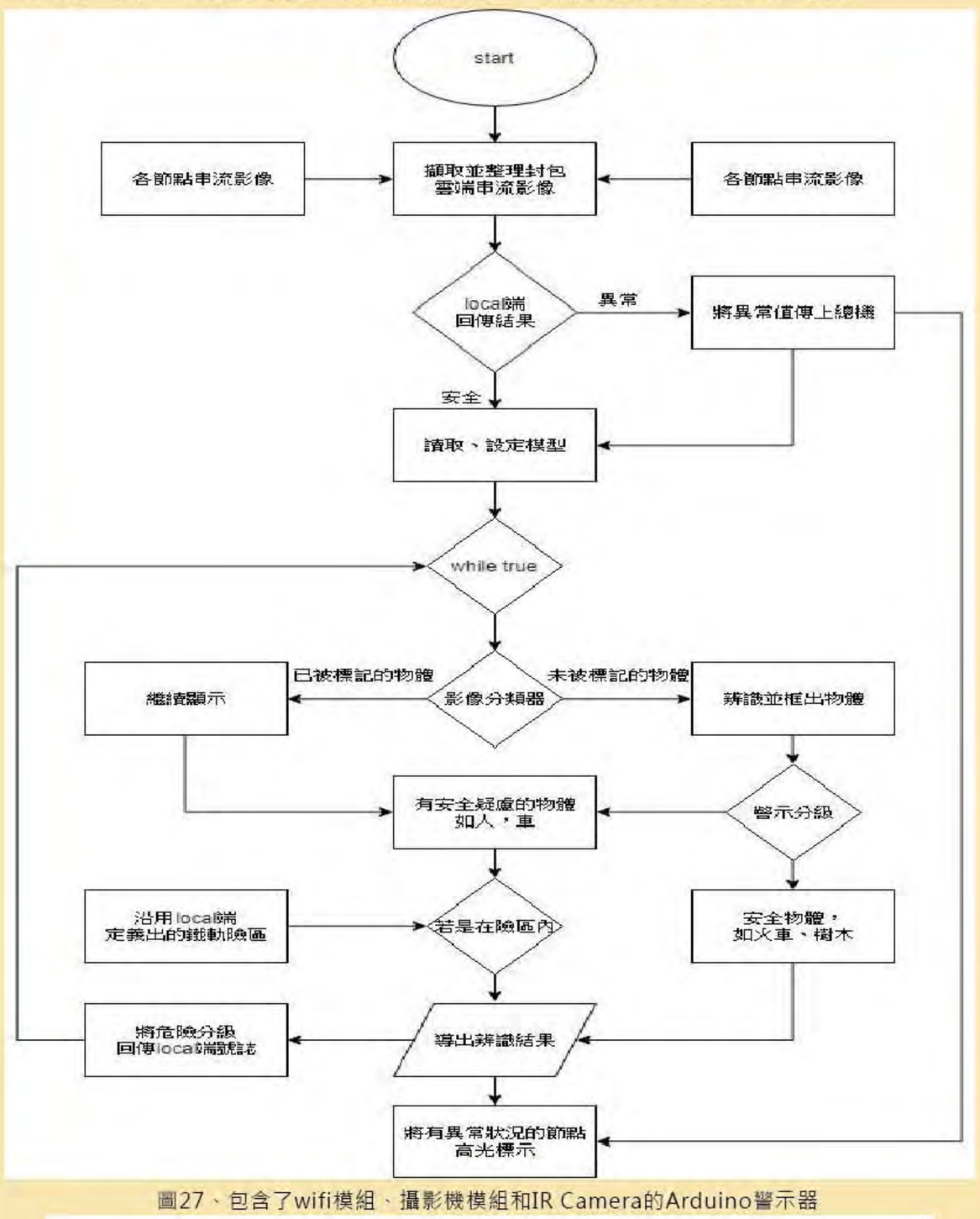


圖27、包含了wifi模組、攝影機模組和IR Camera的Arduino警示器

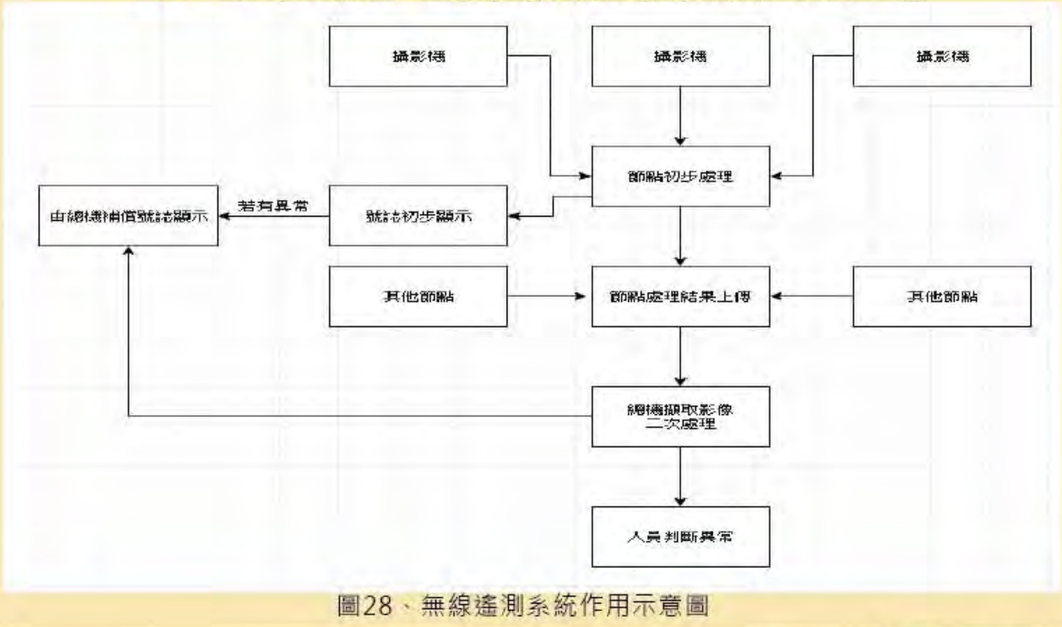


圖28、無線遙測系統作用示意圖

圖29、攝影機動態定義危險區與安全區

本研究的實驗環境為ASUS筆記型電腦，程式開發環境為下表:

表(三)、硬體規格

項目	規格
型號	ASUS X542U
CPU	i5-8250U
硬碟	Toshiba 128G (SSD)； 日立 1TB (HDD、5400rpm)
記憶體	DDR3-1066 4GB
顯示卡	NVIDIA GeForce 940MX

表(四) 軟體開發環境

項目	規格
作業系統	Windows 10 64bit
開發平台	Python 3.6
開發語言	
Open CV	

3. 動態捕捉險區定義

將鐵軌的影像數據訓練至使用的模型中，使攝影機可以動態定義危險區等分界，能夠更快速方便的安裝攝影機，不須再次定義出分界，且若是因外力讓攝影機偏離定義完成的範圍，能更快速準確的校正。

本研究測試資料為模擬6組不同位置的攝影畫面，影片解析度為1600*900像素，影片速度平均為24FPS，長度20秒。準確率計算方式如下，其中TP代表信賴值80%以上，FP代表信賴值低於80%，FN代表遺失偵測的目標。

Accuracy = TP/(TP+FP+FN) False alarm = FP/(TP+FP) Miss = FN/(TP+FN)

使用Python的Pyfirmata擴充套件，與Arduino連動，使我們所製作之警示器模擬真實情況，並讀取已經有設定過的資料庫與檔案，當YOLO偵測系統的偵測物在安全區內時處在安全狀態，且綠燈亮起，當偵測到危險物體進入危險區時，綠燈熄滅，亮起紅燈，危險狀態解除後再亮起綠燈，熄滅紅燈，且偵測到之物體也會由Python系統紀錄並匯出，藉以得知在過程中有何物體進入危險區。

結合監控節點、雲端和動態險區定義的特點，這項系統不只可以運用在火車鐵軌上，也可以結合其他功能用在各個區域，如高速公路上，做為道路監控或是在行車紀錄器上監測行車死角等。

圖28、人在安全範圍內，亮綠燈

圖29、人在危險區，綠燈熄滅，亮紅燈

(一)流明與準確度分析結果

圖30-1: 1500流明準確度

圖30-2: 1200流明準確度

圖30-3: 900流明準確度

圖30-4: 600流明準確度

圖30-5: 300流明準確度

圖30-6: 100流明準確度

(二)第一代在六個測試環境中的實驗結果

表(五) 第一代測試數據

	Frames	True Pos.	False Pos.	False Neg.	Miss rate	False Alarm	Accuracy
Test1	480	1640	80	104	5.96%	4.65%	89.91%
Test2	480	805	127	103	11.34%	13.63%	77.78%
Test3	480	682	101	96	12.34%	12.90%	77.59%
Test4	480	702	110	102	12.69%	13.55%	76.81%
Test5	480	1020	106	100	8.93%	9.41%	83.20%
Test6	480	1011	104	106	9.49%	9.33%	82.80%

(三)第二代在六個測試環境中的實驗結果

表(六) 第二代測試數據

	Frames	True Pos.	False Pos.	False Neg.	Miss rate	False Alarm	Accuracy
Test1	480	1120	51	113	9.16%	4.36%	87.23%
Test2	480	677	101	96	12.42%	12.98%	77.46%
Test3	480	1011	96	83	7.59%	8.67%	84.96%
Test4	480	1300	106	103	7.34%	7.54%	86.15%
Test5	480	1020	106	100	8.93%	9.41%	83.20%
Test6	480	1011	104	106	9.49%	9.33%	82.80%

(四)第二代在六個測試環境中的實驗結果

表(七) 第三代測試數據

	Frames	True Pos.	False Pos.	False Neg.	Miss rate	False Alarm	Accuracy
Test1	480	1220	49	92	7.01%	3.86%	89.64%
Test2	480	1077	60	101	8.57%	5.28%	87.00%
Test3	480	1061	86	90	7.82%	7.50%	85.77%
Test4	480	1210	91	93	7.14%	6.99%	86.80%
Test5	480	1001	102	94	8.58%	9.25%	83.63%
Test6	480	962	93	102	9.59%	8.82%	83.15%

陸、問題與討論

本研究設計的構思與演變與問題如下:

(一) 起初剛想要運用辨識系統時，我們剛開始本來打算使用現成的辨識系統，並利用圖片辨識物體的方式來偵測某一範圍內的物品種類，但由於我們是利用照片來分析，所以對於我們所想辨識的方式有所差異，所以我們就想將其改良成能辨識影像的系統。

(二) 我們最先找到的程式為OpenCV是因為其為時下十分熱門的辨識程式之一，它只需要具備C語言或C++語言的能力即可加以編輯，也因為OpenCV的獨立性，讓使用者在使用時不需要插入函式庫，我們所使用的是利用Python並外加OpenCV模組進行辨識，但經過多次實驗後，我們發現雖然它具有十分高的辨識速度，不過它的程式結構較為簡單，時常發生辨識錯誤的情況，不符合本研究要求，所以繼續尋找更合適的程式。

(三) 在找OpenCV時我們也有看到另一個十分熱門的辨識系統叫作「YOLO」，他是一款由Darknet所開發的程式系統，因為他在網路上已有許多人應用，所以對於我們來說也十分方便，且其系統主要為運用目前較新興的Python系統外加Dark-flow模組和其他部分模組即可編輯，而我們也利用Python的程式將其加入了信賴值的功能，使我們可以大略知道其準確率為多少，以便降低準確率，其中，我們是利用OpenCV的影像讀取與畫圖結合YOLO的系統進行辨識，大幅提升其功能性與方便性。

(四) 當我們在討論要使用其系統於火車上時，想將其架設在月台與平交道，但因兩者的辨識模式不大相同，前者是判斷在某區域內進行判定，並將訊號傳送至火車上的控制室與中央控制室使人員得知消息，並即時做出反應，以降低危險，後者則是在平交道的軌道附近架設鏡頭，並辨識在火車行徑範圍內是否有危險物，並在平交道設置燈號，如果有，就會亮起紅燈並發出警示，讓警衛能迅速解除危險，若有火車即將行經，則也會向列車控制室提出警示以降低車速，以免發生危險。

表(八) 兩系統之比較

系統	OpenCV	YOLO 與 OpenCV 整合
編輯程式	Python	Python
辨識目標速度	高	高
準確率	較低	較高
靈敏度	較不穩定	穩定
使用方便性	使用較為方便	須自行編寫部分程式碼
模組	OpenCV 機器視覺擴充套件、Haar 分類器	Darkflow 模組、OpenCV 模組與其他輔助模組
樣本數量範圍	400 到 500	1000 到 1500

(五) 最後在考慮到實際應用時，若是光線不足會影響系統判讀，故加入IR Camera(紅外線攝影機)，補足在夜間光線不足的問題，以及將每個需要監控的地點視為節點，將資料初步處理後及時回饋到本地的警示系統，將會更符合實際應用。

柒、結論

綜合以上研究，本團隊歸納出以下論點:

一、 此辨識系統可運用在各種需辨識物體之地方，且其所要辨識之物體可經由設定來決定，其危險程度也可再進一步的設置，若要增加辨識區域與準確度，需搭配望遠鏡頭與高畫質感光元件使用。

二、 此辨識方式比起人類自己肉眼辨識準確許多，可以有效輔助駕駛人員突發事件處理，因此可大幅度降低因失誤而造成之影響。

三、 本系統所設定之危險範圍可藉由調整程式內的數值來改變，可適應各種不同的需求方式，使用起來十分便利。

四、 本系統有設定信賴值，可藉由其所顯示的數值來得知其準確率，以減少其誤差，由實驗結果顯示，系統會因流明度差異而影響準確率，建議火車與軌道照明設備盡可能充足與加強，這樣對駕駛員與系統皆能提高辨識效果。

五、 本系統運用於火車月台與平交道辨識時，可藉其所判斷之物體來警告列車長減速，並通知中控室以幫助列車長做出行動，以避免因人為疏失或反應時間不足所造成之意外。

六、 此辨識系統結合wifi模組、攝影機模組和IR Camera的Arduino警示器，將會提高夜間辨識能力，與即時資料回饋，讓監控者在最短時間內做出正確的判斷。

捌、參考資料來源

1. 官鎮任(2018)臺灣警政單位導入即時影像人臉辨識系統關鍵成功因素之研究，國防大學資訊管理系碩士論文
2. 林俊傑，林群倫，應用OpenCV完成影像辨識功能，安全管理與工程技術國際研討會，民國102年
3. https://opencv.org/ -OpenCV官網
4. http://monkeycoding.com/?page_id=12 -OpenCV阿洲的程式教學
5. https://pythonprogramming.net/haar-cascade-face-eye-detection-python-opencv-tutorial/ -OpenCVHaar cascade及時辨識教學
6. https://pjreddie.com/darknet/yolo/ -YOLO官網
7. https://blog.csdn.net/tangwei2014/article/details/50915317 -YOLO論文閱讀筆記
8. https://github.com/thtrieu/darkflow -Darkflow模組來源
9. https://matplotlib.org/ -Matplotlib模組官網
10. https://github.com/markjay4k/YOLO-series/blob/master/part6%20-%20draw_box.py -Matplotlib網格標記程式來源
11. https://github.com/markjay4k/YOLO-series/blob/master/part4_video.py -YOLO攝影機連動程式來源
12. https://www.arduino.cc/ -Arduino官網
13. https://www.arduino.cc/en/Reference/Firmata -Firmata官網
14. http://yehnan.blogspot.tw/2016/01/arduinowindowspythonfirmataarduino.html -Firmata Arduino連動教學

圖9、無人在偵測範圍

圖10、在偵測範圍內有人

在鏡頭範圍內規劃出危險區和安全區，若是人體在安全區則表示安全無恙，但如果人進入了危險區規定的座標內便發出警告，如(圖11)。

圖11、將鏡頭範圍內規劃出危險區和安全區

```
    for (x,y,w,h) in bodies:

        cv2.rectangle(img,(x,y),(x+w,y+h),(255,0,0),2)
        roi_gray = gray[y:y+h, x:x+w]
        roi_color = img[y:y+h, x:x+w]    #在已偵測到的人體上畫出藍色方框

        if x < 335:

            print("危險")    #當物體經過警戒線時，在Console打出危險

    cv2.imshow('img',img)    #顯示出攝影機視窗
    k = cv2.waitKey(30) & 0xff
    if k == 27:
        break

cap.release()
cv2.destroyAllWindows()
```

圖12、在程式中定義出警戒線，當X座標小於335時，發出(危險)訊息

圖13-1、人在安全區內

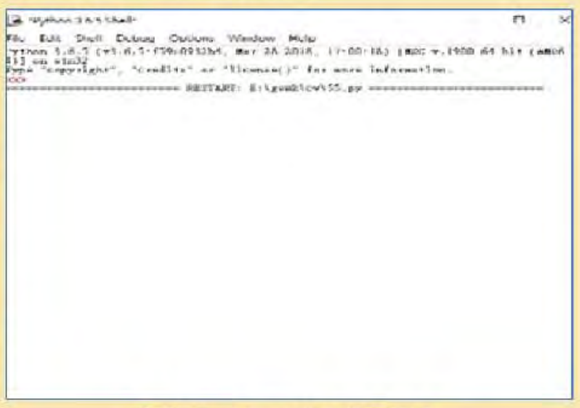


圖13-2、Console不顯示警告

圖14-1、人在危險區內

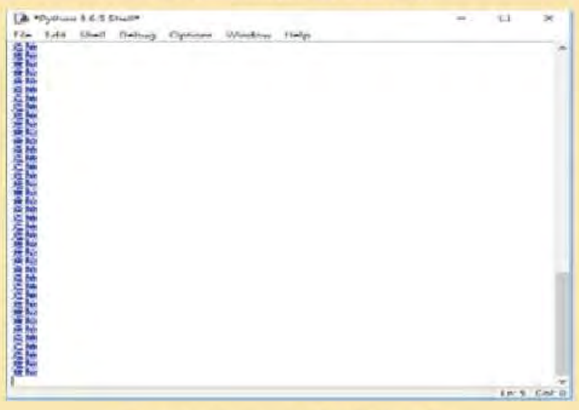


圖14-2、Console不斷打出(危險)

在經過多次測試後，我們得出了一個結論，雖然OpenCV的辨識速度和每秒張數都很高，但因為分類器的訓練樣本不足(約400到500張照片)和其本身功能較為簡單的原因，十分容易發生誤判的情況，常常將會動或者是有陰影的東西看成人，對於快速行動的目標也不夠靈敏，對於人來人往的平交道、月台，這樣的辨識能力是不合格的，因此我們需要一個能夠同時辨識多重目標，且更加準確的系統來代替OpenCV。

(二)第二代系統

為了讓程式能有更精確和辨識多樣物體的能力，我們決定使用由Darknet類神經系統架構的YOLO即時影像辨識系統。

1.程式撰寫

由於Darknet只相容於Linux系統，使許多不支援Linux的攝影機無法使用，因此我們使用了Python的Darkflow模組，使Darknet能在Windows系統下運行。

```
File  Edit  Format  Run  Options  Window  Help
import cv2
from darkflow.net.build import TFNet    #使用Darkflow作為yolo的基本運行環境
import numpy as np
import time

options = {
    'model': 'cfg/yolo.cfg',
    'load': 'bin/yolov2.weights',
    'threshold': 0.8,
    'gpu': 0.6,    #匯入yolov2和 .weight檔
                   #設定偵測靈敏度和gpu使用率
}

tfnet = TFNet(options)
colors = [tuple(255 * np.random.rand(3)) for _ in range(10)]

capture = cv2.VideoCapture(1)
capture.set(cv2.CAP_PROP_FRAME_WIDTH, 1920)
capture.set(cv2.CAP_PROP_FRAME_HEIGHT, 1080)

while True:
    stime = time.time()
    ret, frame = capture.read()
    if ret:
        results = tfnet.return_predict(frame)
        for color, result in zip(colors, results):
            tl = (result['topleft']['x'], result['topleft']['y'])
            br = (result['bottomright']['x'], result['bottomright']['y'])
            label = result['label']
            confidence = result['confidence']
            text = '{}: {:.0f}%'.format(label, confidence * 100)
            frame = cv2.rectangle(frame, tl, br, color, 5)
            frame = cv2.putText(
                frame, text, tl, cv2.FONT_HERSHEY_COMPLEX, 1, (0, 0, 255), 2)    #打出程式推測的物件名稱和信任值
            print(label)    #文字化偵測結果
        cv2.imshow('frame', frame)

    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

capture.release()
cv2.destroyAllWindows()
```

圖15、將Darkflow模組匯入Python

將Darkflow模組匯入Python中，用OpenCV中原本的物體標記系統標出目標物，打出程式推測的物件名稱和信賴值，並在Console中打出文字化後的結果，如(圖15)。

運用YOLO自帶的學習功能來訓練背景的出現的異物，將背景、火車訓練為安全正常的物體，將人、動物等等視為可能造成威脅的物體，再把軌道和月台的的間隔分為安全區和警示區，當物體和分隔好的區域重疊時，再出現不同級別的警示。

圖16、更加靈敏的多重辨識，並準確的認出物件的種類

圖17、即使一段距離外也能辨識出物體(信賴值80%以上)

四、實際應用

對自製的火車月台模型拍攝大量照片，蒐集至少1000張後，用python的matplotlib網格標記程式把人、火車和其他物體的方格座標標記起來，將資料存成 .xml檔，將檔案訓練成一個.weight檔(類似於Haar分類器檔案)。

圖18、車站月台模型圖

圖19、簡易火車模型

圖20、蒐集各個不同角度的火車照片

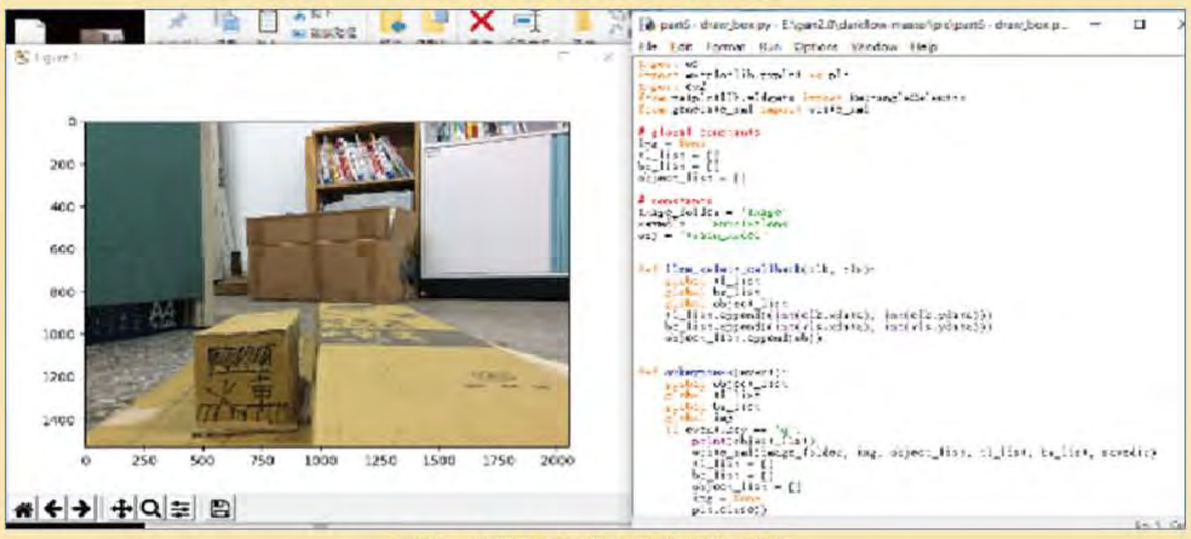


圖21、將多張圖片匯入網格標記程式中

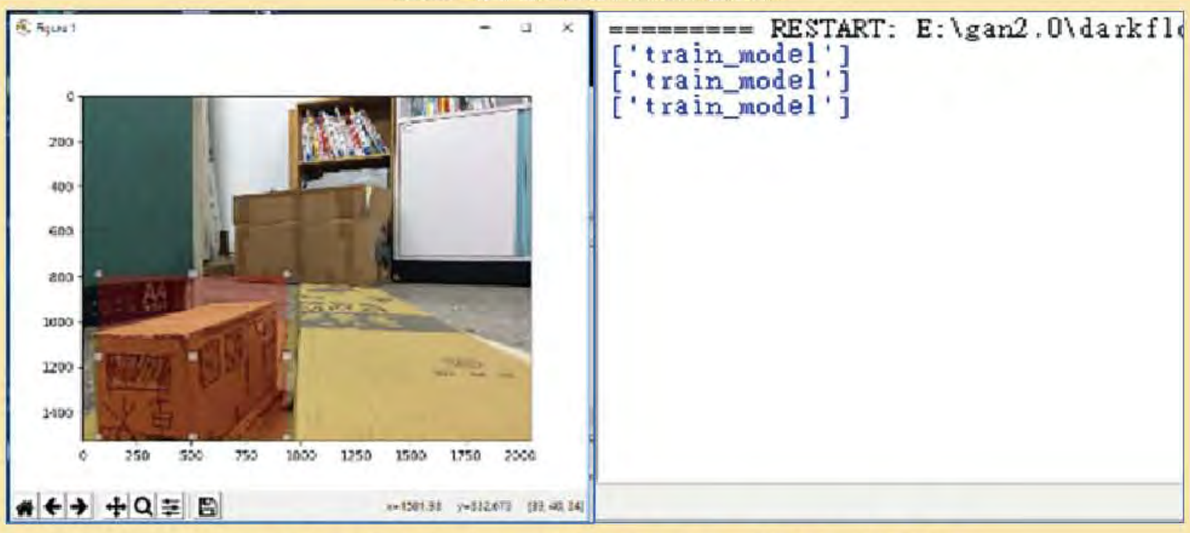


圖22、將火車模型劃記起來後，程式記下座標(左圖)，並記下物體種類(右圖)

訓練完 .weight檔後，將所見到的目標以鮮豔的方框標記起來，並在方框上寫出可能的物件的推測和信賴值，定義出能造成威脅的物體，例如人、動物、垃圾等，和正常的物體，如火車，將偵測到的結果文字化後輸出，在資料庫中加入經過物體的物體類別、座標、信賴值。

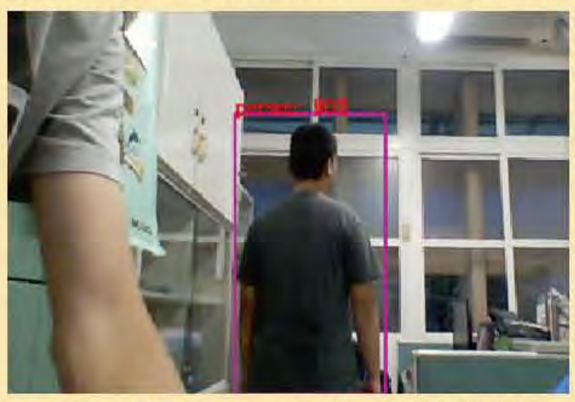


圖23、系統偵測到的人，並在頭上顯示名稱和信賴值

在程式碼定義出危險區和安全區，若是已經由定義過的危險物體進入危險區，則系統發出警示，而火車進入危險區時則顯示正常，已經由設定過的物體若是出現在安全區內，不會出現

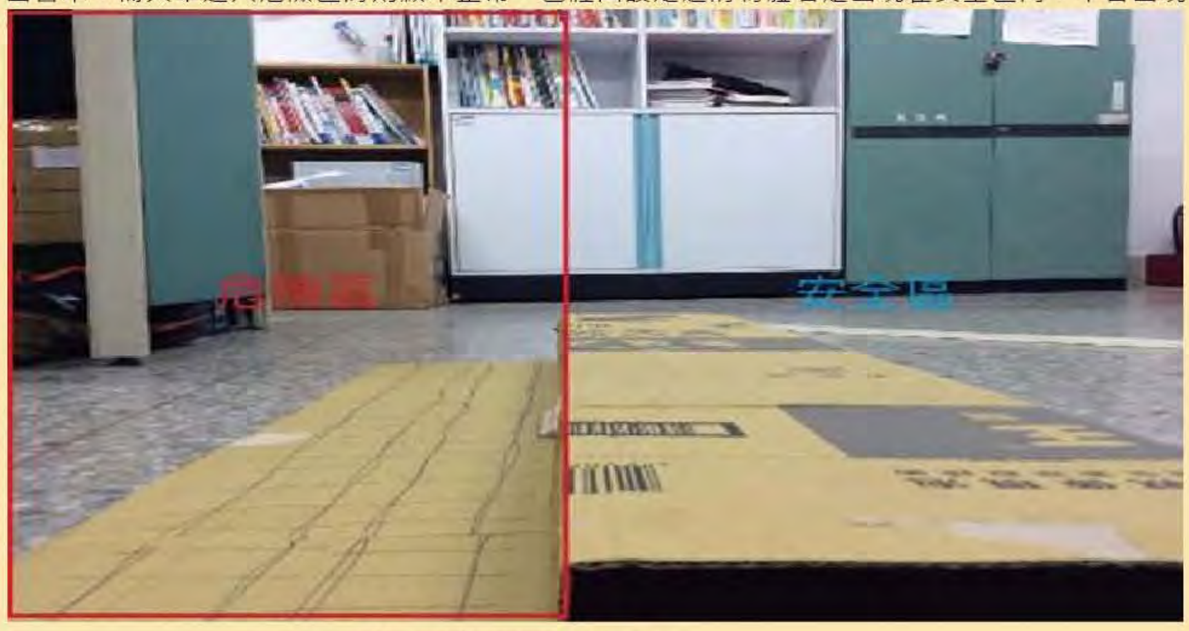


圖25、對月台模型的危險區定義