以 TK1 實作廣域視角之物件偵測系統

專題編號：105-CSIE-S009
執行期限：104 年第 1 學期至 105 年第 1 學期
指導教授：陳彥霖
專題參與人員： 102590451 陳修志
102590037 呂孟穎

一、摘要

本研究根據目前市場上，對於影像監控之需求，以 TK1 實作廣視角之物件偵測系統。主要解決兩個問題。第一，單一攝影機盲點區域可由對向攝影機協助解決。第二，以成本低廉但運算能力高的 TK1 進行影像分析。根據結果顯示，本研究確實完成廣域視角之物件偵測，也驗證完成 TK1 的開發環境與周邊功能。未來，根據不同的需求，僅只替換核心演算法即可。

關鍵詞：廣域視角、TK1、影像分析

二、緣由與目的

常駐型的攝影機隨處可見，被架設在各個角落做長期的俯視錄影。有些攝影機甚至配合著監控系統監視著某個區域。然而，攝影機有著「視角越大、焦距越短」的缺點，如圖一與二所示。當監視區域較大時，會有盲點區域的產生，監控系統的效果會降低，有時甚至造成誤報結果。本研究將透過兩台攝影機的對視架設互相彌補，再配合 TK1[1][2]使用 CUDA 技術讓原本的影像辨識運算達到加速的效果，完成無死角的俯視行人偵測系統。

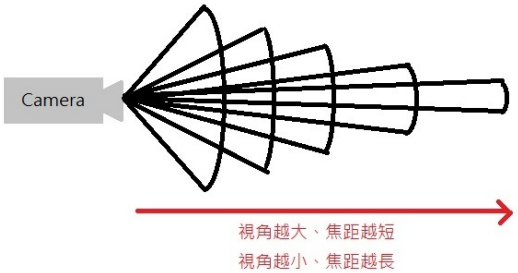


圖 1 攝影機視角與焦距的相對關係

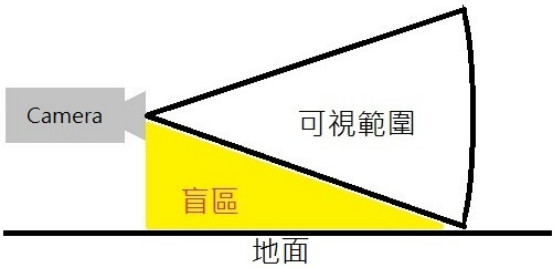


圖 2 盲點區域的產生

三、研究報告內容

整個系統設計如下圖所示。圖三為監控點之系統架構；圖四為監控室之系統架構圖。

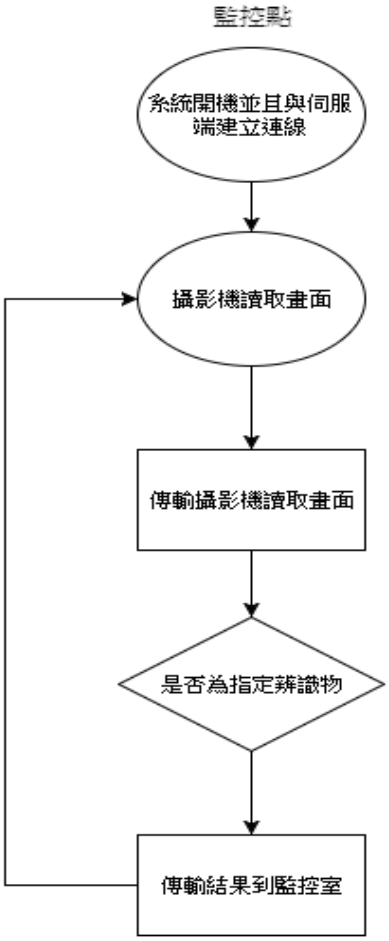


圖 3 監控點之系統架構圖

其中，在監控點部分，本研究採用 TK1 硬體，TK1 是一塊以視覺運算應用為出發點的嵌入式開發板，提供開發者許多元件與 I/O。對於我們系統而言，最大的特點為 CUDA 技術。在 GPU 的平行運算之下，能快速地執行影像辨識，達到 Real Time 的需求。為了將 TK1 的 CUDA 效能能有效發揮，我們在此採用 CUDA 中的 Zero Copy 的功能，由於 TK1 的 Device 記憶體和 Host 記憶體是共用的，執行 CUDA 時不需要再複製記憶體到 Device，執行結束也不用再複製到 Host，一般的 CUDA 化在複製記憶體的過程中會花掉大部分的時間，但在 Zero Copy 下不用複製，因此大幅提升執行上的效能。

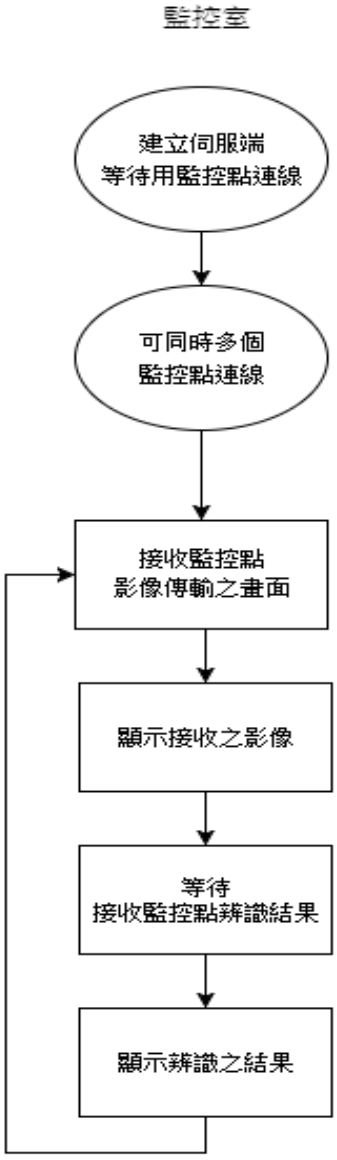


圖 4 監控室之系統架構圖

整個系統藉由無線方式傳輸，監控區域有多個監控點，能夠同時讓各個監控點傳輸即時狀況到監控室，同時透過 AP 當作傳輸媒介達到遠端的效果，除此之外無線的設備簡單輕巧，不像有線傳輸有實體的線材來影響架設時的限制。

（一）、系統架設部分

將對視的兩支攝影機架設於監控區域的邊界上，達到入口把關的效果，如下圖五，並架設第三台 TK1 接顯示器或是電腦做為終端，接收兩台 TK1 所發出的資料，並決定出最後輸出結果顯示在螢幕上。

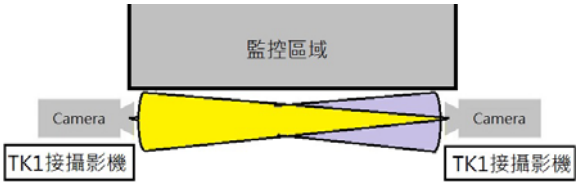


圖 5 監控區域與攝影機架設之相對位置

（二）、影像辨識流程圖

執行影像辨識時，同時參考到當前影像與前一張影像。先對兩張影像做灰階化、平滑法作為前處理，並將兩張影像相減，以取得空間變化區域。若相減差值在閥值內，則視為無變動區域，並將像素值設定為 0；反之，則為變動區域，並將像素值設定為 255，藉此得到一張二值化影像。變動區域以輪廓最為明顯，因此對此二值化影像做連通區域，即可得到完整的變化區，便可得到 ROI 區域。對 ROI 區域進行比例縮放，並計算 HOG 特徵[3]。將特徵值送入事先訓練完的 SVM 分類器進行分類[4][5]，若「是」行人，則監控端會回傳訊號 1 給終端；反之，則監控端會回傳訊號 0 給終端。終端會透過兩邊監控端的結果決策出最後的輸出。

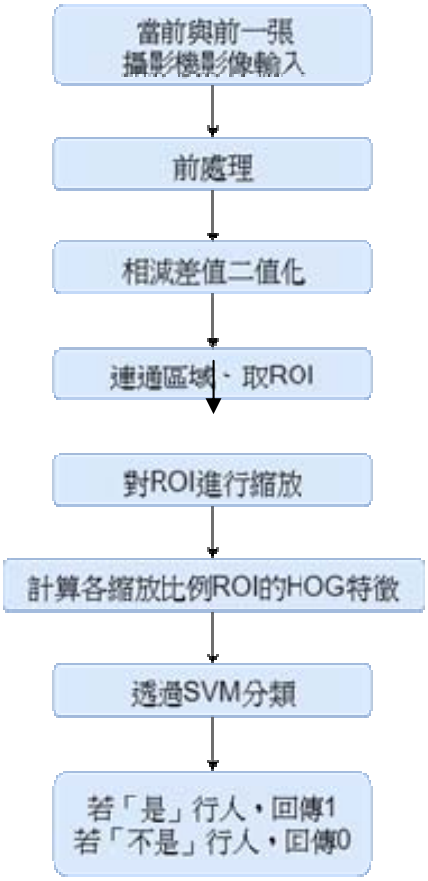


圖 6 影像辨識流程圖

（三）、終端決策 – Sliding Window 機制

為了降低誤報機率，因此增加 Sliding Window 機制，使得輸出結果不單單只是參考當下由監控端回傳的訊號，而同時參考了過去單位時間內的結果，來決策出最後的輸出。系統中，我們參考的過去結果為兩個單位，若過去兩個單位的結果加上當下的結果，有兩個（含）以上為 1，則最後決策判斷監控區域受到入侵；反之，則判定為雜訊或是誤報，最後決策判斷監控區域沒有受到入侵。

（四）、CUDA 技術加速運算

TK1 的 CUDA 核心和一般 GPU 不太相同，它的記憶體是和 CPU 共用的，因此可以使用 Zero Copy 的方式，免除複製記憶體(Host-To-Device 和 Device-To-Host)所花的時間，讓 CUDA 化的程式得到更多的加速。以下為 CUDA 化部分

1. RGB to Gray

每一個要灰階的點都沒有相依性，而且運算量有 640X480 個點運算量龐大非常適合 CUDA 化。下表為加速結果

表 1 加速測試數據

次數	CPU 計算	GPU 計算
1	2.04088sec/千次	1.88231sec/千次
2	2.22296sec/千次	1.73502sec/千次
3	2.17264sec/千次	1.78973sec/千次
4	2.11521sec/千次	1.87328sec/千次
5	2.10549sec/千次	1.76374sec/千次
平均提升	2.131436 (sec)	1.808816 (sec)
	提升 1.17 倍	

2. Sobel 與正規化

每一個像素點要做一次計算，且計算量龐大，每一個點之間沒有相依性，所以適合 CUDA 化。

表 2 加速測試數據

次數	CPU 計算	GPU 計算
1	104.982sec/千次	19.7935sec/千次
2	104.981sec/千次	20.021sec/千次
3	105.016sec/千次	20.3526sec/千次
4	105.013sec/千次	20.3636sec/千次
5	105.028sec/千次	19.5908sec/千次
平均提升	105.004 (sec)	20.0243 (sec)
	提升 5.24 倍	

3. 監控點與監控室傳輸

系統的主要傳輸，把每一個監控點的結果傳輸到監控室，但會有多個監控點，因此監控室需要使用多執行緒的方式[6]-[8]。以下為主要排程方法，分為 8 個 Step。

1st. 接收監控點的連線。

2nd. 配置編號給監控點。

3rd. 監控點收到編號後會回傳所取得之編號，跟監控室再次確認配置編號。

4th. 待所有監控點完成 Step2 或 Step7。個監控點開始運作，傳輸與偵測執行。

5th. 所有監控點皆完成 Step3，再開始執行。監控點傳輸影像之 Width(寬度)給監控室，監控室接收完成後回傳 Get。

6th. 所有監控點完成 Step4，再開始執行。監控點傳輸影像之 Height(高度)給監控室，監控室接收完成後回傳 Get。

7th. 所有監控點完成 Step5，再開始執行。監控點傳輸影像中每個像素值給監控室，監控室接收完成後回傳 Get。

8th. 所有監控點完成 Step6，再開始執行。監控點傳輸影像辨識結果給監控室，如果在監控點判別為人會傳輸座標給監控室，不為人則傳輸否，監控室接收完成後回傳 Get。之後回到 Step3。

除了 8 個 Step 以外，監控室的接收會由監控點 1→2→3 這樣的方式開始接收，透過 Time Sharing 技術，避免在傳輸的過程中，因為資源競爭，而導致傳輸錯誤。以上每個 Step 都有防止 Lost 的功能，每次傳輸都會先確認要接收多少 bytes，並且一定要接收到這麼多個 Byte，否則會回傳 ERROR，並且進行重傳。在 Kernel 最大傳輸約為 1500 bytes，因此我們將傳輸一筆資料之最大傳輸量設定為 1024bytes。

四、實驗結果

下圖 7 中，為實際運行之系統設置規劃。兩支攝影機分別接至 TK1 中，針對紅色區域進行監控，透過無線傳輸與終端電腦進行資訊交換，最後，將結果呈現於四個分割畫面。有紅色框表示監控端回傳 yes 給終端。預設終端目前記錄皆為非人，如圖 8。

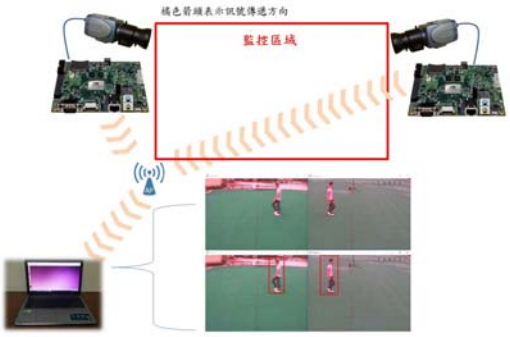


圖 7 系統運行畫面

五、結論

本研究已經成功的藉由 TK1 完成廣域視角之物件偵測系統。透過現有知識，包含影像辨識基礎理論與嵌入式系統實作，完整呈現一個實際之應用系統。

由此成果可知，對於 TK1 的陌生程度勢必下降，且也給予對此應用有期待的產業一份參考。更進一步的，對於有新穎創新的應用，便可以透過此一系統，替換核心演算法之後，即可完成。

參考文獻

[1] http://www.cool3c.com/article/83490, NVIDIA Jetson TK1 開發平台簡介

[2] http://arrayfire.com/zero-copy-on-tegra-k1/, zero copy on TK1

[3] Navneet Dalal and Bill Triggs, “Histograms of Oriented Gradients for human Detection”, in Proc. Of CVPR, pages 886-893, 2005.

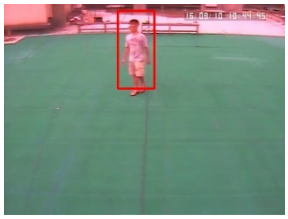
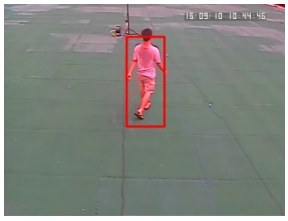
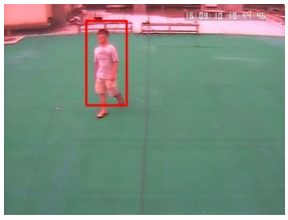
[4] http://opencv.org/, opencv 官方網站

[5] http://mathworld.wolfram.com/L2-Norm.html, Wolfram

[6] http://www.takobear.tw/2014/02/28/bearsocket/, Socket 教學

[7] https://kheresy.wordpress.com/2012/07/06/multi-thread-programming-in-c-thread-pl/, c++多執行緒

[8] http://horacio9573.no-ip.org/cuda/index/html, CUDA Library

	Camera 1	Camera 2	終端結果說明
↓↓↓影像串流↓↓↓	 Return yes	 Return no	Terminal Buffer：NN Camera 1 有偵測到行人，但終端前兩次結果紀錄皆為 0，因此判定無入侵。
	 Return yes	 Return no	Terminal Buffer：NY 有 Camera 偵測到行人，含終端紀錄共兩次入侵紀錄，因此判定受到入侵！
	 Return yes	 Return no	Terminal Buffer：YY 有 Camera 偵測到行人，含終端紀錄共三次入侵紀錄，因此判定受到入侵！
	 Return yes	 Return yes	Terminal Buffer：YY 有 Camera 偵測到行人，含終端紀錄共三次入侵紀錄，因此判定受到入侵！
	 Return yes	 Return no	Terminal Buffer：YY 有 Camera 偵測到行人，含終端紀錄共三次入侵紀錄，因此判定受到入侵！
	 Return yes	 Return yes	Terminal Buffer：YY 有 Camera 偵測到行人，含終端紀錄共三次入侵紀錄，因此判定受到入侵！

	 Return no	 Return no	Terminal Buffer：YY 無 Camera 偵測到行人，但終端紀錄共兩次入侵紀錄，因此判定受到入侵！
	 Return yes	 Return yes	Terminal Buffer：YN 有 Camera 偵測到行人，含終端紀錄共兩次入侵紀錄，因此判定受到入侵！
	 Return no	 Return yes	Terminal Buffer：NY 有 Camera 偵測到行人，含終端紀錄共三次入侵紀錄，因此判定受到入侵！
	 Return no	 Return no	Terminal Buffer：YY 無 Camera 偵測到行人，但終端紀錄共兩次入侵紀錄，因此判定受到入侵！

圖 8 物件偵測運行過程說明