AIoT 智慧紅綠燈

梁友誠　　廖子齊　　葉怡君　　王友群
國立中山大學資訊工程學系
{t2725528, dick850529, eugenia017}@gmail.com; ycwang@cse.nsysu.edu.tw

摘要

塞車一直是嚴重的現代交通問題,塞車不僅會造成交通時間上的增長，也會增加燃油消耗，進而造成更多的空氣汙染，而交通號誌的週期正是影響車流量的因素之一。傳統交通號誌的時相週期為固定，無法即時依照車流狀況來調整時相週期，這容易造成已擁塞車道的汽車往往要等待車流量小甚至幾乎無車流的車道之紅燈結束才能往前進，從而更加惡化塞車問題，而近年來物聯網和人工智慧話題越來越熱門，兩者所結合而成的 AIoT 產業也蓬勃發展，因此，在本論文中，我們希望結合物聯網與深度學習物件辨識的方式，設計一套能夠即時辨別車流量之系統，並藉此動態改變交通號誌時相週期以舒緩塞車狀況,並將 AIoT 應用至智慧城市中。

關鍵詞:物聯網、深度學習、AIoT、物件辨識、交通號誌。

1. 前言

現行台灣的交通號誌系統主要是採用固定秒數模式，倘若遇到尖峰時段，原本交通號誌的固定秒數無法應對繁忙擁塞的路口，導致堵塞的狀況繼續惡性循環，無法自行消化車流量，因此需要交通警察到現場指揮來疏散堵塞的車輛，但現實生活的人力有限，無法在每一個路口分配交警來指揮交通，因此，能夠依車流量而自動化調控的交通號誌系統正是解決此問題的方法之一。

目前車流量的計算多半採聲納式、紅外線、環路線圈式等各式不同的車輛偵測器(Vehicle detector, VD)來測量，然而建構 VD 及後續維護成本較高，在有限的成本下無法廣泛建置成為一大缺點。近年來電腦硬體成本大幅下降以及深度學習在影像辨識方面日益成熟的情況下，以影像來計算車流量成為一可行的方法，成本也比傳統的 VD 更低也更方便設置。而透過在十字路口架設網路攝影機，我們可以因此獲得路口的即時影像，進而連接網路將影像上傳至伺服器以機器學習模型來分類出汽車類型，並計算道路上的車流量以及不同方向的車流數目，之後依運算結果來決定如何調整交通號誌的時相週期，再將調整時間回傳至路口的交通號誌來進行更動。

在本論文中，我們以 ESP32-cam 開發版作為影像伺服器，將網路攝影機的影像透過 WiFi 來上傳至伺服器，並交由另一組運算伺服器獲取影像伺服器中的即時道路影像,使用物件偵測模型 YOLO [1]來辨識即時影像中的車輛，再套用我們所設計之車流量計算演算法後獲得下一階段紅綠燈時相，把更改時間方案透過網路回傳至已設置紅綠燈及七段顯示器之 ESP32，最後使用微觀車流模擬軟體 SUMO [2]來驗證我們所設計之下一週期紅綠燈秒數是否能改善當前路況；為達成此目的，其研究方向與欲解決的問題有以下二者:

一、物件偵測:能對影像中不同的物件辨識出各式的汽車類型(例如:小客車、卡車、公車等)，並且能夠計算影像中有幾輛汽車，其目的是計算車道中的即時車流量。

二、交通號誌變更時相方案:如何依據車流量來調整時相週期，使其成為高效率的系統以便舒緩車流，讓車輛堵塞的道路擁有較長的綠燈秒數，而車流順暢的道路則擁有較短的綠燈秒數。

本論文的章節架構如下:第二章介紹本系統所使用的相關技術。第三章詳細說明研究方法，系統測試則在第四章進行討論，最後，我們將於第五章提出結論與未來工作。

2. 相關技術

2.1. 卷積神經網路

卷積神經網路(Convolutional neural networks, CNN)通常用於將圖片分類，主要由卷積層、池化層及全連接層所組成，其原理為藉由將圖片切割成較小的特徵圖，並比較該圖片之特徵與訓練用圖片的相似度以達成分類之目的。

卷積層負責計算整張圖片中有多少相符的特徵，藉由將原圖和 Kernel 上的像素相乘並取平均值後得到的值作為該位置的相似度，越接近原圖所得到的分數越接近 1，和原圖差異越大則越接近-1。

池化層則將圖片壓縮並保留重要資訊，通常以 2x2 或 3x3 作為一區塊，接著取每個區塊中的最大值以組成新的圖片，如此更可專注於圖片是否存在相符特徵，並忽略不重要的像素以減少運算量。

圖片經過多個特徵卷積及池化後，產出含有多個特徵的資訊，全連接層則會給予相對應的權重，最後將它們進行彙整，而圖片與最高分者則會判斷為同一類別。

2.2. 深度學習物件偵測

最早將 CNN 引用至電腦視覺物件偵測領域是由 Ross Girshick 等人所提出的 R-CNN [4]，作者提到先將原始圖片裡篩選出約 2000 個候選區域(Region proposal)，再將每個區域丟入預先訓練好的 CNN 模型以獲得特徵，然後再以 SVM (Support Vector Machine)來區分該區域屬於哪一類型，最後使用線性回歸模型來校正物件框(Bounding box)位置，以上作法雖然比傳統物件偵測方法來的有效精準，但此方法必須先找出 Region proposal 再進行物件辨識，造成運算量太大而導致速度緩慢。而後由 Redmon 等人所提出的 YOLO [1]則採用 One-stage 方式，即影像從輸入到輸出預測結果只靠一個 CNN 就可同時偵測物件位置及辨識物件，這樣的作法整體辨識度雖然不比 R-CNN 來的高，但可大幅提升整體的運算速度。而為了能夠即時偵測網路攝影機所擷取道路上的車輛之影像，我們在本論文中選擇以 YOLOv3-tiny [3]當作我們的物件偵測模型，因為它是目前所有物件偵測模型裡運算速度最快，且仍有較高準確度的模型。

2.3. YOLO (YOU ONLY LOOK ONCE)

在文獻[1]中，作者提出 YOLO 的第一個版本(YOLOv1)，它將輸入影像切分成 SxS 個網格(grid)，每個網格會以自己作為中心點去預測 B 個 Bounding box 與其信心分數(Confidence score)，以及屬於哪個物件類別的機率，其中，信心分數為代表該 Bounding box 含有物體的信心程度，而在實作上，作者只讓每個網格去預測 2 個 Bounding box，這使得 YOLOv1 對於小物體或是兩個相鄰且中心點非常相近的物體在偵測上有效果不佳的缺點，圖 1 為 YOLOv1 運作之示意圖。

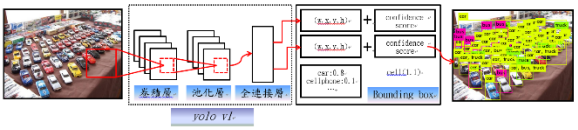

圖 1 YOLOv1 運作之示意圖

而文獻[5]則以 YOLOv1 為基礎，對模型進行修改並維持其偵測速度，首先 YOLOv2 移除原本的連接層，改用卷積層以提高物體的分辨率，並引入 Faster R-CNN 所提出的 Anchor box 來預測物體所在的邊界框，其利用不同大小及長寬比的 Anchor box 使網路輸出的邊界框能夠更符合實際的物體大小，再來是加入一層 Passthrough 層，將原本 26x26x512 的特徵圖轉換成 13x13x2048 的特徵圖，如此一來，便可提升偵測小物體的準確度。

而 YOLOv3 [6]主要改進有兩點，第一是使用網路層數更深的 Darknet53 來擷取輸入圖片更多的特徵，其特色為加入 ResNet 來解決神經網路加深時所遭遇到的退化(Degradation)問題，第二則是使用 Feature pyramid networks 來提升小物體的預測能力，將原本的單層 13x13 特徵層改成多層的 13x13、26x26 和 52x52 特徵層，可分別偵測大、中、小的物體，並讓每層輸出 3 個 Bounding box，如此一來，

它可比 YOLOv2 所輸出的 5 個 bounding box 能更加地定位物體的位置，並提高對於小物體的預測。

在本論文中，我們一開始使用 YOLOv3 來作為偵測車輛的模型，然而經過實驗發現 YOLOv3 在我們所使用的設備 NVIDIA GTX-960M GPU 中的 FPS(Frames per second)嚴重不足，而無法達成即時辨識的效果，因此我們改用[3]中所提出的 YOLOv3-tiny 作為我們的模型，此模型為 YOLOv3 的輕量版，縮減了特徵擷取網路中卷積層的數量，雖然這會使得整體的辨識量下降，但經過我們測試 YOLOv3-tiny 仍能在網路攝影機所拍攝道的車輛有不錯的辨識能力，因此最後我們使用 YOLOv3-tiny 作為我們的辨識模型。圖 2 為 YOLOv3-tiny 的架構圖。

```
   layer   filters  size/strd(dil)      input                output
   0 conv     16       3 x 3/ 1    416 x 416 x   3 ->  416 x 416 x  16
   1 max                2x 2/ 2    416 x 416 x  16 ->  208 x 208 x  16
   2 conv     32       3 x 3/ 1    208 x 208 x  16 ->  208 x 208 x  32
   3 max                2x 2/ 2    208 x 208 x  32 ->  104 x 104 x  32
   4 conv     64       3 x 3/ 1    104 x 104 x  32 ->  104 x 104 x  64
   5 max                2x 2/ 2    104 x 104 x  64 ->   52 x  52 x  64
   6 conv    128       3 x 3/ 1     52 x  52 x  64 ->   52 x  52 x 128
   7 max                2x 2/ 2     52 x  52 x 128 ->   26 x  26 x 128
   8 conv    256       3 x 3/ 1     26 x  26 x 128 ->   26 x  26 x 256
   9 max                2x 2/ 2     26 x  26 x 256 ->   13 x  13 x 256
  10 conv    512       3 x 3/ 1     13 x  13 x 256 ->   13 x  13 x 512
  11 max                2x 2/ 1     13 x  13 x 512 ->   13 x  13 x 512
  12 conv   1024       3 x 3/ 1     13 x  13 x 512 ->   13 x  13 x1024
  13 conv    256       1 x 1/ 1     13 x  13 x1024 ->   13 x  13 x 256
  14 conv    512       3 x 3/ 1     13 x  13 x 256 ->   13 x  13 x 512
  15 conv    255       1 x 1/ 1     13 x  13 x 512 ->   13 x  13 x 255
  16 yolo
  17 route  13                                     ->   13 x  13 x 256
  18 conv    128       1 x 1/ 1     13 x  13 x 256 ->   13 x  13 x 128
  19 upsample                 2x    13 x  13 x 128 ->   26 x  26 x 128
  20 route  19 8                                   ->   26 x  26 x 384
  21 conv    256       3 x 3/ 1     26 x  26 x 384 ->   26 x  26 x 256
  22 conv    255       1 x 1/ 1     26 x  26 x 256 ->   26 x  26 x 255
  23 yolo
```

圖 2 YOLOv3-tiny 的架構圖

3. 研究方法

3.1. 系統架構

我們的系統架構如圖 3 所示，在此系統中，我們利用 WiFi 將攝影機所獲取的道路影像上傳至網頁影像伺服器中，並且讓運算伺服器擷取網頁影像伺服器的道路影像，經由 YOLO 辨識影像中的車輛數，再使用車流量演算法計算車流量，之後將此結果套用至以事先在 SUMO 模擬好的參數來獲得下一週期的紅綠燈時間，並將時間傳送到搭載 TCP server 的紅綠燈上(ESP32)，使其可以參考此時間來實際變換硬體上的 LED 燈。

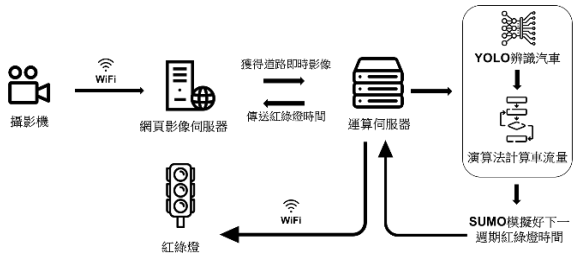

圖 3 系統架構圖

3.2. 模型訓練

我們採用 COCO 資料集[7]當作訓練我們的 YOLOv3-tiny 模型，該資料集為 Microsoft 所提供的開源影像資料庫，它包含大量以標記的圖片可供模型進行訓練，其資料庫含有 80 種物件類別，包括我們在此論文中需要偵測的物件(例如:小客車、卡車、公車等)，而文獻[3]中提到使用 COCO 資料集訓練 YOLOv3-tiny 可達到 220FPS 的速率，這符合本論文需要對輸入影像進行即時運算之要求。

3.3. 車流量計算演算法

圖 4 為本論文所使用的車流量計算演算法。此其目的為讓某路口等待紅燈之車輛盡可能在下一週期之綠燈內全數通過，此演算法一開始會先針對某個道路之車流進行計算，在該道路紅燈開始倒數計時，就會將即時車流影片中的畫面經由 YOLO 運算，而影片是由一張張連續的影像所組合成連續動畫，因此 YOLO 運算的單位也是以影片中每張連續影像來運算，並將原影像加入物件偵測後所產生之 Bounding box 作為結果，再將這些結果組合起來成為物件偵測的影片，因此，當即時車流影片作為 YOLO 的輸入時，我們可以得到構成此影片之每張影像經過物件偵測後所得到之結果，而每個結果都含有該影像所有偵測到物件之 Bounding box，而我們將這些 Bounding box 取出並統計含有小客車、卡車、公車之 Bounding box 總數，因為卡車及公車的體積與車輛長度比小客車來的大，所以我們將所獲得的卡車與公車的 Bounding box 總數乘以 2，以符合實際情況，並與小客車之 Bounding box 總數相加

得到此影像之分數，之後，會以此路口紅燈結束前 5 秒擷取這期間所統計到最大分數，再利用此分數將路況分為 5 種道路擁塞之情況，並搭配已事先再 SUMO 模擬好之參數，來決定此路口下一週期之綠燈秒數為何，以便讓等待紅燈之車輛能全數通過，最後將此綠燈秒數傳給 ESP32，切換至偵測另一方向之路口車流，並重新開始演算法。

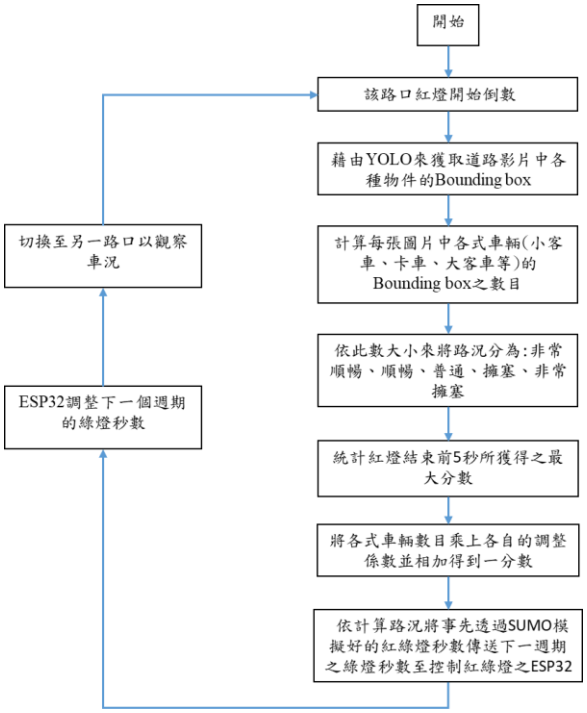

圖 4 車流量計算之演算法

3.4. 紅綠燈時間制定

為了能讓等待紅燈的車輛能在下一週期綠燈內儘可能全數通過，我們需要為每種不同路況制定相對的紅燈及綠燈時間，圖 5 為我們使用 SUMO 所設計的含有兩個十字路口的常見路網，此路網為雙向三線道，分別為左轉專用、直行專用及右轉和直行皆可通行的之車道。

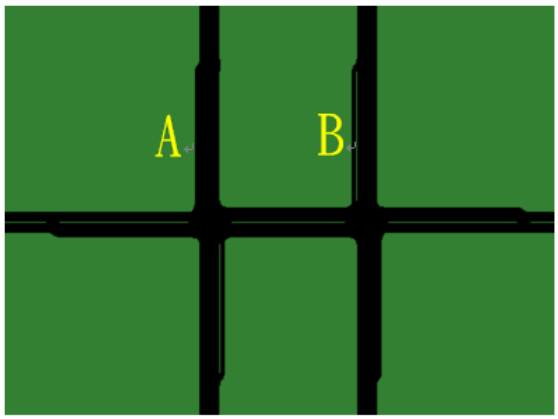

圖 5 本論文利用 SUMO 所設計之路網圖

以台灣來說，一般道路速限多半為 50 到 60 時速公里，因此我們將車流的最高車速定為 50 公里/小時，在此情況下的黃燈秒數為 3 秒、全紅燈秒數為 2 秒[8]，接著，我們假設兩路口的距離最多能塞下 25 輛車子，故將這些車輛數平均分成 5 種程度範圍，而紅綠燈的時相變化會如圖 6 所示，再者，為了盡量使累積的車輛數控制在 25 輛以內，我們必須制定一個累積車輛時間的上界，根據高雄市 109 年 5 月份主要道路平均流量及速率統計表[9]，在只有 2 車道的路段中，最高尖峰流量為 1,699 輛/小時，因此取 1,700 輛/小時為最高流量去計算產生 25 輛車需幾秒，最後得出累積車輛時間需盡量少於 52 秒。根據圖 6 得知，我們只需決定全部車輛皆可以通行的秒數(即 A 與 a)以及只限左轉車輛可通行的秒數(即 B 與 b)即可完成整個週期，經過我們使用 SUMO 所模擬出的結果，平均一輛直行車通過路口約需 2 秒、右轉車需 2.5 秒、左轉車則需 3 秒，在左轉車的部分會先以前 3 輛車為基礎，後來每多一輛車就再多 3 秒去計算。

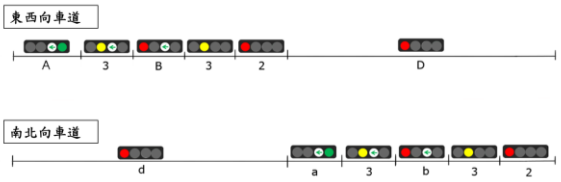

圖 6 紅綠燈時相圖

有了以上資訊，我們便可制定紅綠燈秒數，先將車輛範圍內的最高車輛數分成左轉、直行與右轉

的車輛，左轉車輛數為最高車輛數的三分之一並取其下界，其餘平均分配給直行及右轉車輛，在全部車輛皆可以通行的秒數的部份，由於直行及右轉車輛只能在此時通過，且右轉通過所需要的時間較長，故為主要參考的對象，因此可以得到全數右轉車和左轉車通過時間需至少幾秒，如圖 7 所示。

車輛數	左轉車輛數	全部左轉車通過至少需要的時間(單位：秒)	1/2左轉車通過至少需要的時間(單位：秒)	右轉車輛數	全部右轉車通過至少需要的時間(單位：秒)
25	8	18	6	9	23
20	6	12	3	7	18
15	5	9	3	5	13
10	3	3	小於3	4	10
5	1	小於3	小於3	2	5

圖 7 右轉車及左轉車通過所需之時間

在實驗一開始我們想要讓等待的車輛都能通過，但它會使累積車輛時間超過 52 秒，因此我們會優先考慮讓等待直行和右轉的車輛全數通過，這是因為這段時間主要通過的車輛數佔了三分之二，此外，在[10]文獻中還有提到，可以通過間隙理論計算穿越衝突流的最大左轉流量，因此在這段時間仍有少數左轉車是有機會通過的，而在只限左轉車輛可通行秒數的部份，以讓 50%比例的左轉車通過的秒數作為基準。在累積車輛時間不超過 52 秒的情況下，我們會以能全數通過為目標的在只限左轉車輛可通行的時相增加秒數，而在全部車輛皆可以通行的時相會多給 2 至 5 秒的時間，讓後來抵達的直行及右轉車子能多通過一些，在兩者互相協調下最終定出圖 8 來當作我們最終採取的方案。

由於我們的場景設定為兩個十字路口(如圖 5 所示)，而兩路口所執行的紅綠燈秒數皆一樣，倘若 A 和 B 兩路口所對應到的擁塞程度不同時，或是一條路左右兩向程度不同時，我們會選擇較擁塞的程度去執行。

程度	車輛範圍	 (單位：秒)	(單位：秒)	左轉車輛數	可過車輛數
非常壅塞	大於21輛	23~28	6~9	8	4~6
壅塞	16~20	18~20	6~9	6	4~6
普通	11~15	13~18	6~9	5	4~6
順暢	6~10	10~15	3~6	3	3~5
非常順暢	0~5	5~10	3~6	1	3~5

圖 8 各種道路擁塞程度所需之綠燈秒數

4. 系統測試

我們使用 ESP32-cam 來拍攝螢幕上所撥出的一段實際車流影片，以測試 YOLOv3-tiny 對於偵測車輛的準確度如何，圖 9 與圖 10 分別為原始的影像以及經由 YOLOv3-tiny 偵測後所輸出的影像，在輸入影像解析度為 640x480 的情況下，可獲得 30FPS 的影像輸出，且可發現對於較近的車輛能有良好的辨識率，然而對於較遠方的車輛則較辨識不出來，而在車輛總數部份，經過 YOLOv3-tiny 所偵測出的車輛總數為 8 輛，而若實際計算該路口之車輛數為 10 輛，因此準確率並未達到百分之百，然而我們所決定更改紅綠燈時間的方案是取決於車輛數範圍，而非準確的車輛數目，也因此，此誤差算是在本實驗中可接受範圍內，最後我們也實作出一套搭載七段顯示器以及兩組紅、黃、綠 LED 燈之硬體來模擬實際生活的場景(如圖 11 所示)，並且以此硬體來測試紅綠燈是否有接收到運算伺服器所傳送的下一週期綠燈秒數。

圖 9 ESP32-cam 所拍攝之影像原圖

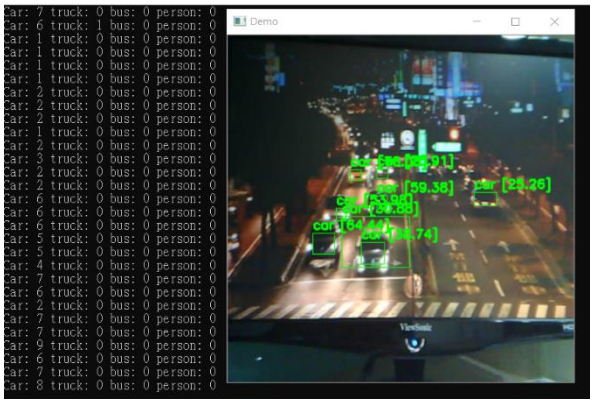

圖 10 經由 YOLOv3-tiny 運算之結果

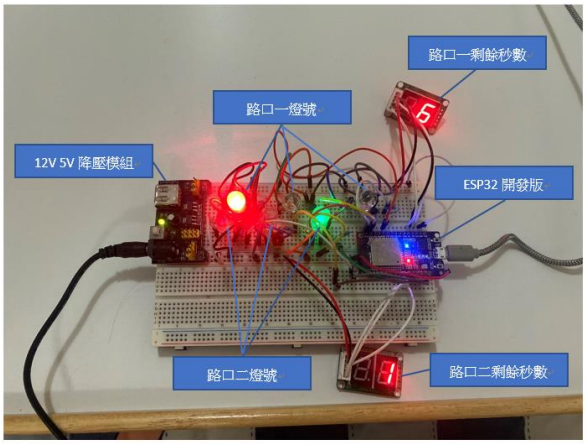

圖 11 模擬實際場景之硬體

5. 結論與未來工作

本論文利用 YOLO 辨識車輛，並藉由自動化調整紅綠燈時相的方式減少道路上指揮交通所需的人力，同時也可藉由攝影機回傳的影像來了解是否有道路的突發狀況發生，而事先使用 SUMO 模擬好的紅綠燈秒數也可以較精準地為該路段各種車流情形提供合適的紅綠燈秒數，以減少每台車輛的平均旅遊時間，從結果來看目前在車輛辨識率上仍有改進的空間，我們之後會嘗試使用較好的運算伺服器以及攝影機來改善硬體方面的問題，並改用辨識率更高的模型來提升整體辨識率，最後也會改善車流量計算的演算法，以因應更多不同的道路狀況。

參考文獻

[1] J. Redmon, S. Divvala, R. Girshick, and A.Farhadi, “You only look once: Unified, real-time object detection,” *IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 779-788.

[2] P. A. Lopez et al., "Microscopic Traffic Simulation using SUMO," *IEEE Intelligent Transportation Systems Conference*, 2018, pp. 2575-2582.

[3] J. Redmon, “Darknet: Open source neural networks in C”, [Online] Available: http://pjreddie.com/darknet/.

[4] R. Girshick, J. Donahue, T. Darrell, and J. Malik, “Rich feature hierarchies for accurate object detection and semantic segmentation,” *IEEE Conference on Computer Vision and Pattern Recognition*, 2014, pp. 580-587.

[5] J. Redmon and A. Farhadi, "YOLO9000: Better, faster, stronger," *IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 6517-6525.

[6] J. Redmon and A. Farhadi. arXiv, 2018. “YOLOv3: An incremental improvement,” [Online]. Available: https://arxiv.org/abs/1804.02767

[7] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P.Dollár, and C. L. Zitnick, "Microsoft coco: Common objects in context," *European Conference on Computer Vision*, 2014, pp. 740–755.

[8] 台北市交通管制工程處，”交通號誌黃燈秒數如何設定．”[Online]Available:https://www.bote.gov.taipei/News_Content.aspx?n=20A2BA930381C524&s=C796F047EFED8F61

[9] 高雄市政府交通局,”高雄市 109 年 5 月份主要道路平均流量及速率統計表(東西向)”. [Online] Available: https://www.tbkc.gov.tw/upload/WebGroupList/97/0f07442a-46f6-453b-883c-125c5f6797ea/AllFiles/109%E5%B9%B45%E6%9C%88%E5%88%86%E6%9E%90%E7%B6%B2%E7%AB%99%E7%94%A8.pdf

[10] J. Q. Leng, Y. Q. Feng, J. Zhai, L. Bao, and Y. He, “Travel time model of left-turning vehicles at signalized intersection,” *Mathematical Problems in Engineering*, 2012, pp. 1-16.