第十六屆旺宏科學獎

成果報告書

參賽編號：SA16-596
作品名稱：喀擦一下!停車導引
姓名：鍾安
關鍵字：影像辨識、車格計數、停車路線導引

目錄

壹、研究動機

現在的停車場大都在入口處提供一個寫著剩餘車格數的看版給使用者，卻未告訴哪裡還有停車格，而影像辨識現在大都實用於各種物件辨識，例如人臉辨識、車牌辨識，並未運用於停車場資訊中，因此希望以影像方式擷取停車場資訊，進而給予停車導引。

貳、研究目的

一、了解影像檔案資訊。

二、影像辨識導入停車系統。

三、以影像辨識處理停車場之停車相關資訊。

四、客製化停車導引。

參、研究設備與器材

一、電腦 (Windows7、Windows10)

二、網路攝影機

三、自製停車場模型

四、停車場模型平面設計圖(電子檔)

五、Code::Blocks 16.01

六、OpenCV 2.3.1

肆、研究過程與方法

一、研究流程:

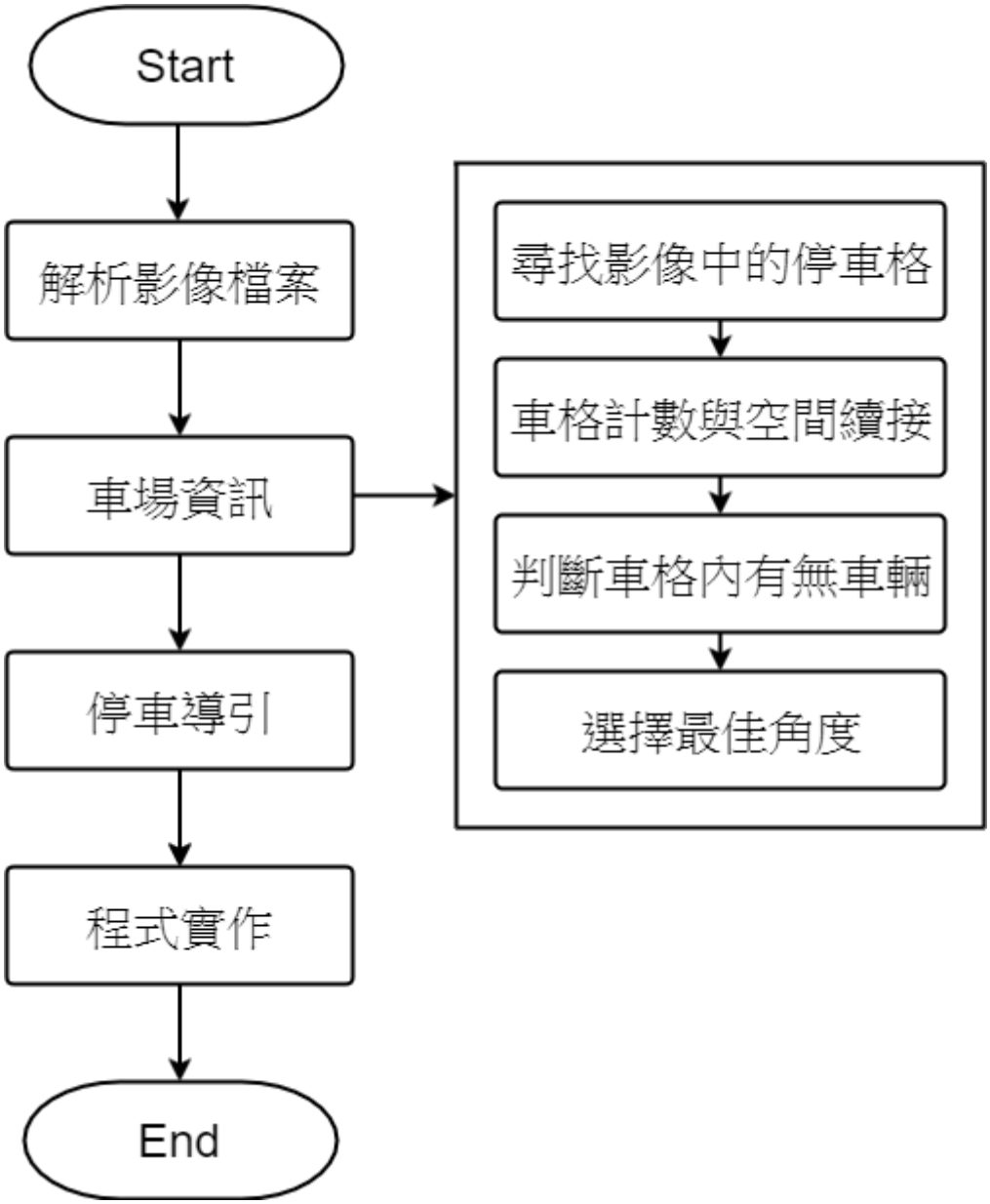


圖 1 研究流程圖

二、影像資訊解析

現今停車場管理系統大都以感測器的方式管理，每個停車位有一感測器模組，但無法完整得知停車場資訊，包含停車格數量及其位置分布，以及停車格是否有車。若對停車場進行攝像，則影像檔案包含以上資訊，所以解析影像資訊有可能獲得車場資訊。車場資訊是指停車場內的停車格數量、停車格的分布位置、出入口的位置以及停車格是否有車。本研究將從影像中取得車場資訊，並將這些資訊整合。現實停車場影像如下：

圖 2 停車場影像 1

圖 3 停車場影像 2

(一)影像檔案類型

常見的影像儲存類型有 JPEG、PNG 以及 GIF：

1. GIF (Graphics Interchange Format)圖形交換格式

   GIF 檔可以儲存透明背景及動畫效果，是透過減色的方式減少檔案大小，只有 256 色，不適用於全彩或漸層豐富的細膩影像，但十分適合存取色塊、顏色簡單的圖形。

2. JPEG(**J**oint **P**hotographic **E**xperts **G**roup)靜態影像格式

   JPEG 檔能儲存 24 位元的全彩影像但不支援透明及動畫，可將影像資料壓縮至原本的十分之一，但壓縮比越高也會越失真。屬於破壞性壓縮，不宜重複壓縮，適用於照片的存取。

3. PNG(Portable Network Graphics)可攜式網路圖像格式

   採用無失真壓縮，可存取 48 位元的彩色影像，有 256 種透明度選擇，改善 GIF 只有透明、半透明兩種選擇及描邊不佳的問題。無失真壓縮，因此檔案通常比 JPEG 大。

由於 GIF 只有 256 色，想要重現真實色彩則略顯不足，在 JPEG 與 PNG 的之間，由於不需重複存取同一張影像，且攝影機影像多以 JPEG 儲存，故本研究選用 JPEG 檔案類型作

為影像儲存的類型。

(二)OpenCV 影像解讀檔案內容

OpenCV 全稱為 Open Source Computer Vision Library，是一個跨平台的電腦視覺庫，可用於開發圖像處理、電腦視覺等程式。

OpenCV 讀入圖片時，將影像文件進行識別，判讀出影像內容後存入 class Matlab（簡稱 Mat）。Class matrix 是 OpenCV 中的特定格式，將影像內容的像素值以 RGB 色彩模型存取成矩陣。

如圖 4 為一張像素 6x6 的影像，使用 OpenCV 讀取後可解讀成圖 5 之 Mat 格式，其第 1 個像素之 RGB 值為 255,255,255，即白色。

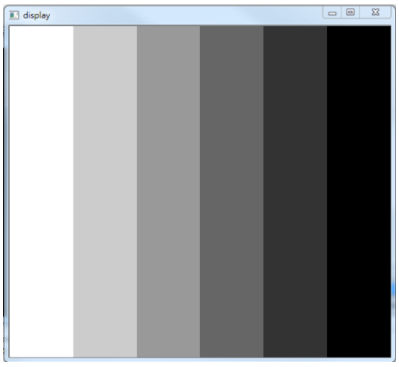


圖 4 像素 6*6 漸層圖

```
[255, 255, 255, 204, 204, 204, 153, 153, 153, 102, 102, 102, 51, 51, 51, 0, 0, 0;
 255, 255, 255, 204, 204, 204, 153, 153, 153, 102, 102, 102, 51, 51, 51, 0, 0, 0;
 255, 255, 255, 204, 204, 204, 153, 153, 153, 102, 102, 102, 51, 51, 51, 0, 0, 0;
 255, 255, 255, 204, 204, 204, 153, 153, 153, 102, 102, 102, 51, 51, 51, 0, 0, 0;
 255, 255, 255, 204, 204, 204, 153, 153, 153, 102, 102, 102, 51, 51, 51, 0, 0, 0;
 255, 255, 255, 204, 204, 204, 153, 153, 153, 102, 102, 102, 51, 51, 51, 0, 0, 0]
```

圖 5 圖 4 的 RGB 值

三、擷取車場資訊

(一)尋找影像中的停車格

觀察停車場影像後發現，車格由 4 條白色直線組成，若能找出這 4 條白色直線，就能找出車格所在的位置，再藉由這 4 條線的交點找出車格的 4 個頂點，我們將車格位置紀錄成 4 個頂點座標，即代表一格車格。

方法 1：找尋影像中的白色直線

1.影像中的直線

直線是同一方向不彎曲的線，而線是點的移動軌跡，具有位置及長度，而無寬度和厚度，在電腦的影像中，以一個像素為點，朝著同一方向移動的不間斷軌跡，位於軌跡上的像素格的 RGB 值都相同，而且有 RGB 值明顯與線上的 RGB 值不同的像

素在兩旁的可判斷為線。

2.簡化圖片

我國法律規定紅線與黃線劃記的地方禁止停車，因此車格線多以白色線為主。而為了更容易辨識影像，我們將影像進行二階化。由於部分顏亮色在二階化時會有顯示成白色之情形，故先將亮色轉成黑色後再進行二階化處理。

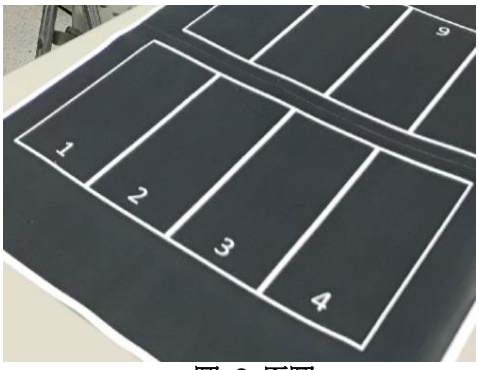


圖 6 原圖

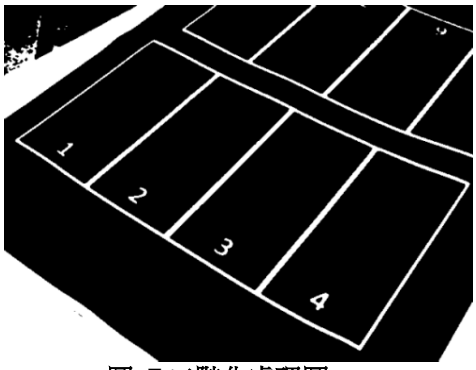


圖 7 二階化處理圖

3.白線辨識方法

從影像左上方的像素開始，判別是白線上的點的設定如下：

(1) 此像素點的 RGB 值為(255,255,255)，即白色。

(2) 此像素在垂直於線(移動方向上)兩旁的點，其中之一的像素的 RGB 值為(0,0,0)，即黑色。

當辨識出一像素為白線上的點，則繼續向一方向上移動，確認同一方向上的點是否位於白線上，直到遇到非白線上的點停止，相近之白線視為同一條白線，將線合併成一條。

4.找線步驟一

從影像左上角的像素開始，以水平線向上 45 度開始辨識白線，完成後以 0.4 度為一間距，持續辨識，辨識到水平線時，改從左下方的像素開始，繼續辨識，直到辨識至水平線而下 45 度為止。由上述方法，可找出影像中部分白線，稱作母線，為了確實判別出車格，先將所有母線依其同方向延伸。

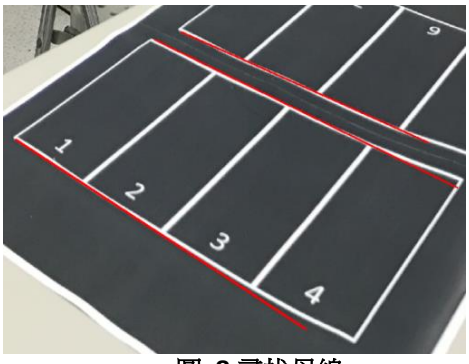


圖 8 尋找母線

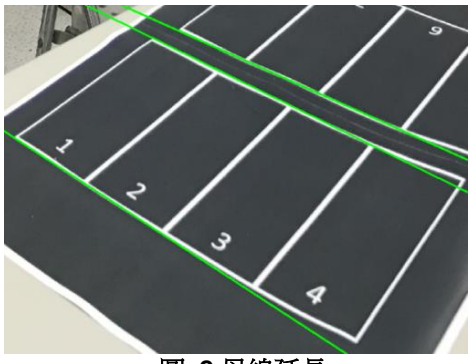


圖 9 母線延長

5.找線步驟二

從母線最左端的像素開始，以母線的垂直線向左 45 度開始辨識白線，完成後以 45 度為一間距，持續辨識至母線的垂直線向右 45 度為止。由上述方法可找出影像中另一方向的白線稱為子線。

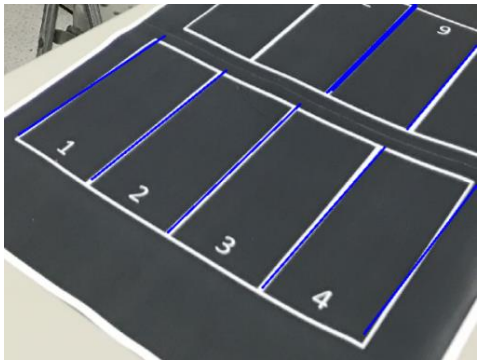


圖 10 尋找子線

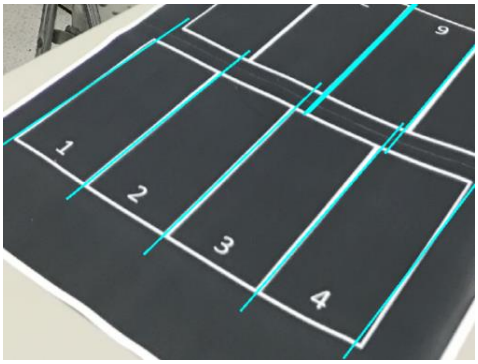


圖 11 子線延長

6.車格的四個位置點為母線與子線的交集處，而為了不要將車道誤判為車格，從母線開始，將子線延長，遇到母線則停止(這時候子線會與母線相交)。兩條平行相鄰的子線，若與相同兩條母線相交，則可判斷這兩條子線與母線之間為一個車格。因此四個相交的點即為車格的位置點。

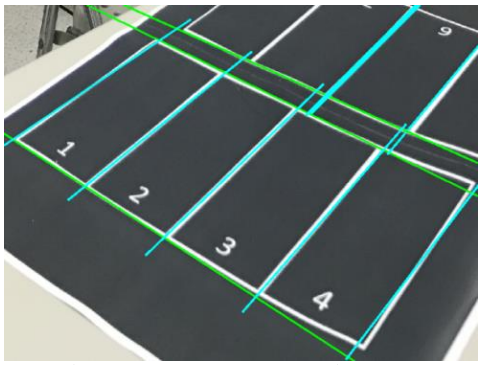


圖 12 母線、子線相交圖

經過測試後，發現方法一找線的運算時間長，應該可以再進一步的改善，因此尋找到 OpenCV 中有一找線方法稱作霍夫找線，可以提升速度與準確率，因此提出方法二來改善。

在實際停車場中，發現因為燈光照射的亮度不同，使得地面亮度不均勻，因此不同區塊的地面，像素強度不同，而不同的停車場影像，地面顏色強度也不同，使得在車格辨識時會有許多干擾，因此提出區域閾值來解決此問題。

方法 2：霍夫轉換找線

1.霍夫轉換

霍夫轉換是一種特徵檢測的方法，廣泛運用在圖像分析、電腦視覺與影像處理，可以用來辨別找出物件中的特徵，例如線條。

霍夫轉換法用於直線偵測的原理，是由於在影像上的任一點會有數條直線經過，一個點會有數條直線通過，將這些線以 r, θ 表示，並映射到 r-θ 關係圖上，會形成一條曲線，如下圖(圖 13)。

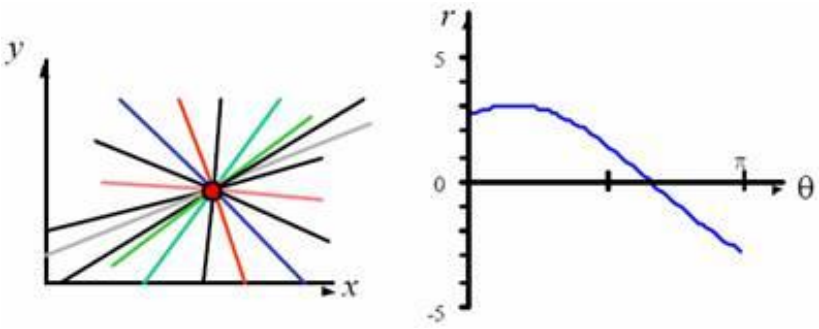


圖 13 霍夫轉換

假如有一條線通過三點，將所有通過那三點的線映射到 r-θ 關係圖上，會產生三條曲線，且三條曲線交於一點，而那個點就是三點所共的線，如下圖(圖 14)。

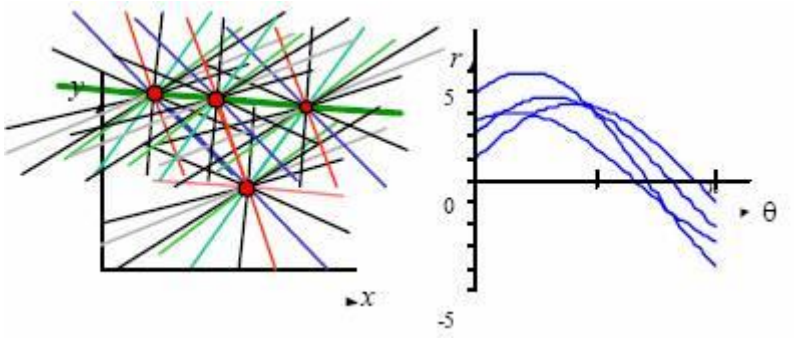


圖 14 霍夫轉換

若在 r-θ 關係圖中，有 N 條曲線交於一點，那點的票數為 N，那只要找到 N 大於定值的點，且該點所代表的線長度夠長，就會是條找出來的線。

2.影像前處理

與方法 1 相同，由於部分亮色在二階化時會有顯示成白色之情形，故先將亮色轉成黑色。但因實際停車場影像中，由於燈光照射的亮度不同，使得地面亮度不均勻，因此不同區塊的地面，顏色強度不同。因此本研究以座標(x,y)為中心，周圍一個正方形區域，計算區域中全部像素強度的算術平均數 μ 、標準差 σ ，將座標(x,y)的像素閥值定為 T(x,y) = μ + C(constant) * σ ，也就是讓影像中每一個像素以所在的區域顏色強度，計算出各自的閥值，稱作區域二值化。

接著，對區域二值化後的影像分別做以下處理：

(1) 膨脹：建立一個 3*3 矩形當作結構元素。在影像中，以結構元素中心沿著白色像素上移動，將被結構元素覆蓋的像素記錄下來，當結構元素中心走過所有白色

像素後，將被記錄的所有像素顯示呈白色，其餘為黑色，就會使影像中白色部分膨脹，如圖 15(以藍色區塊顯示膨脹的部分)。

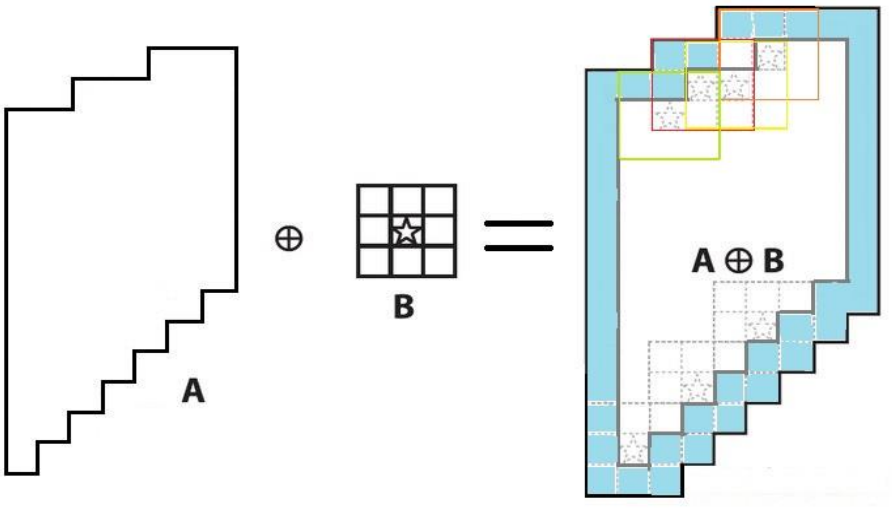


圖 15 膨脹說明圖

(2) 侵蝕：建立 1 個 3*3 結構元素，在影像中，結構元素沿著白色像素內移動，且結構元素完全在白色像素內。將結構元素中心經過的像素記錄，當結構元素走完所有白色像素後，將被記錄的像素顯示呈白色，其餘為黑色，就會使影像中白色部分縮小，如圖 16。

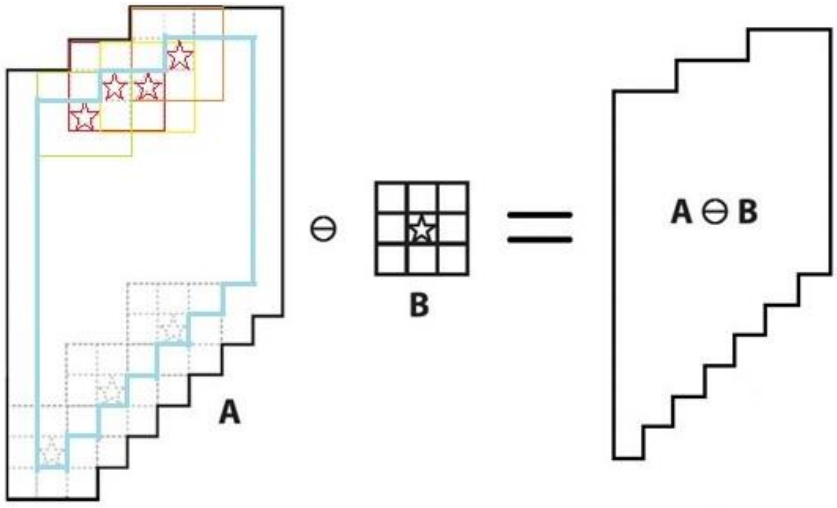


圖 16 侵蝕說明圖

將膨脹後影像減去侵蝕後的影像，可以得到車格四直線的邊緣，我們將此影像稱為 dst。為了減少影像中的雜訊，我們對影像 dst 做開運算，而後再做閉運算，來減少直線邊緣凹凸不平的情形。

3.找線

對影像 dst 使用 OpenCV 中的霍夫轉換直線偵測法，可以在影像中直線的邊緣上找出多條線段，我們將線延長並將鄰近的線段進行合併，最後得到多條直線。將這些

直線依據角度的不同，區分為橫線與縱線。

4.找車格步驟一

先找到一條離影像最下方最近的橫線，稱此線為 M 線。在 M 線上由左至右找，找到一條與 M 線相交的縱線，稱此線為 C 線。從此 C 線由下往上找，若找到一條與子線相交並且離 M 線最近的橫線，則稱此線為 R 線，並將 M 線與 C 線交點、C 線與 R 線的交點記錄下來。而後繼續尋找下一條 C 線，若在 C 線上沒找到 R 線，就跳過此 C 線，繼續尋找下一條 C 線。

5.找車格步驟二

車格是由四個相交的點所組成，在紀錄中相鄰的四個交點可形成一車格，若此車格合乎比例，此四交點即為車格的位置點。

(二)車格計數與空間接續

攝影機的可視範圍之廣角限制、車場的大小及形狀，皆會造成無法由單一的攝影機觀看到停車場全部，因此需要多部攝影機進行拍攝，並把每部攝影機拍攝的空間連接起來。先對平面圖進行車格計數，找出所有的停車格並編號。藉由攝影機廣角、俯角、高度、方向計算出可視範圍，再將攝影機影像中的停車格作車格計數，並把可視範圍內的車格號碼由左而右對照到影像中的車格，就完成了空間續接。

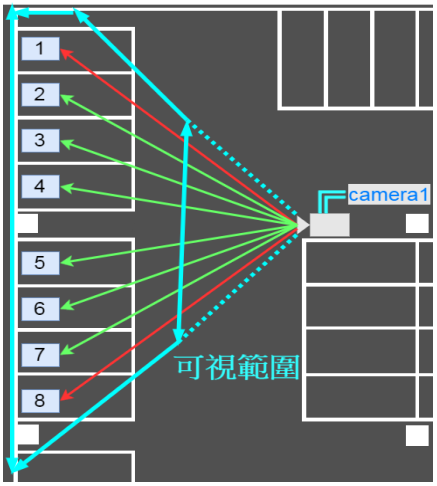


圖 17 平面圖上攝影機可視範圍

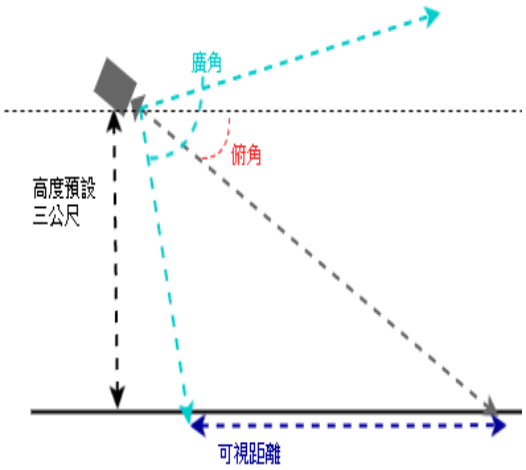


圖 18 計算攝影機可視範圍

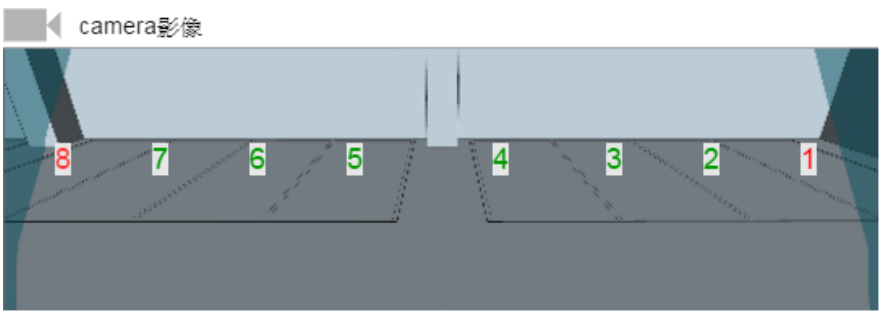


圖 19 攝影機畫面

當有多個攝影機時則分別進行判斷。用同樣的方法一一的算出個別攝影機的可視範圍，再判斷個別攝影機所看到的車格並紀錄之。

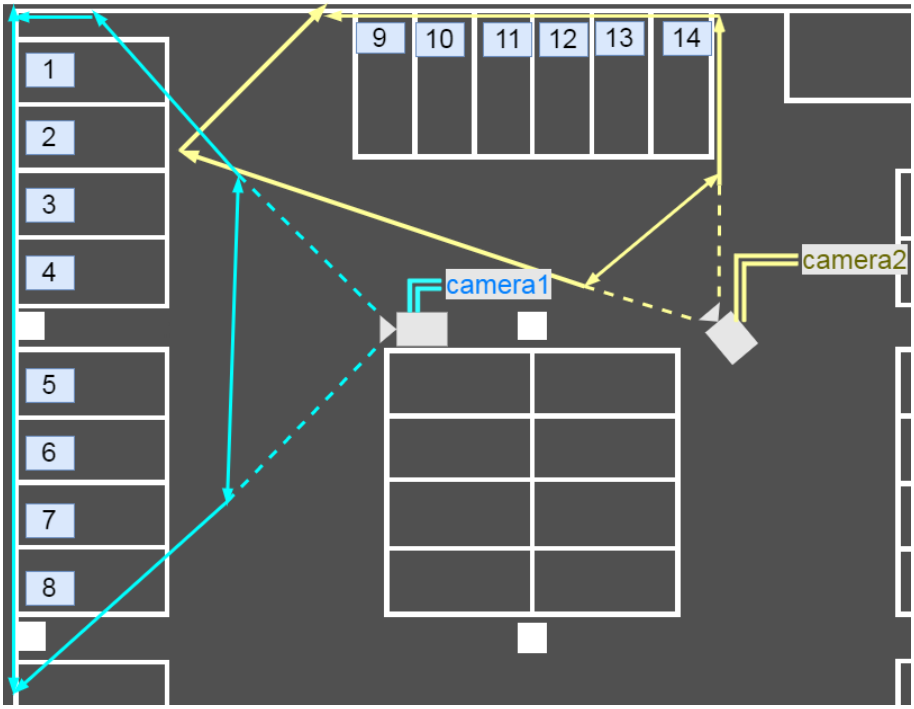


圖 20 多個攝影機在平面圖上

(三)判斷車格內有無車輛

為了判斷出車格內是否有車，將攝影機對著同一停車格拍照，一張有車、一張沒車，比較這兩張的不同處。

圖 21 未停車車格部分位置標示圖

圖 22 已停車車格部分位置標示圖

057	062	065	057	065	067	049	054	057	064	069	073	050	055	058	060	065	068	062	067	070	059	064	067	058	063	066
064	069	072	050	058	060	056	061	064	060	068	071	062	067	071	054	062	064	058	064	064	067	072	075	063	068	072
062	067	070	063	068	071	055	060	063	065	073	076	053	061	064	056	064	066	052	060	062	060	069	068	059	067	069
056	064	066	056	064	066	060	065	068	063	071	074	056	064	067	058	066	068	061	066	069	060	065	068	061	066	069
062	070	072	060	065	069	058	063	067	053	058	062	056	061	065	061	066	069	058	063	066	060	065	068	056	061	064
057	065	067	058	066	069	055	063	065	055	060	063	056	064	067	052	060	062	054	062	064	053	061	063	056	064	066
057	065	067	070	073	078	062	070	072	054	062	064	059	064	068	060	065	068	059	064	067	059	064	067	060	065	068
054	059	062	058	063	067	056	064	067	049	054	057	063	068	071	059	064	067	059	067	069	064	069	072	053	058	061
056	061	064	057	065	068	055	063	066	056	061	065	051	056	060	053	058	062	054	062	065	074	079	083	055	060	064

圖 23　圖 21 紅線框出部分的 RGB 值

049	055	051	040	044	047	101	054	034	017	007	008	193	192	187	196	195	191	194	195	190	194	194	192	193	193	191
048	054	052	042	046	047	157	118	101	030	000	000	041	021	022	060	040	041	061	040	037	050	026	024	047	022	015
049	055	055	049	055	055	206	177	161	203	115	075	215	123	076	225	130	082	229	134	086	218	122	074	219	123	075
059	065	065	051	057	057	160	132	121	206	123	083	210	118	071	222	127	081	221	126	080	220	123	078	216	120	072
064	068	071	059	060	062	102	072	062	197	114	070	219	127	080	215	122	078	216	120	070	213	119	067	216	118	071
059	063	064	061	065	066	090	065	058	093	043	020	224	133	086	214	126	080	215	128	083	202	121	076	201	123	084
056	061	064	060	068	070	051	053	050	079	065	062	028	022	024	025	021	020	013	012	017	013	011	012	016	016	016
059	064	067	059	064	067	058	063	066	053	057	058	060	066	066	054	062	064	055	063	065	047	053	053	058	062	063
071	076	079	054	059	062	054	059	062	062	067	070	057	063	063	046	050	051	054	059	062	055	060	063	060	064	065

圖 24　圖 22 紅線框出部分的 RGB 值

在 RGB 值圖中，紅線框出了色差大的部分，兩張圖相同位置色差大的部分就是車。

攝影機對著同一位置拍攝，在沒有車時先拍一張，稱為無車圖，接著定時取一張攝影機現況圖，將在現況圖與無車圖中位置相同的像素格的 RGB 值分別相減，若大於色差閥值，代表此像素格是車的一部份。經過調查最小規格的汽車在法律規定的車格中佔的比例約為 39%，因此將所有代表車的像素格數除以車格全部像素格數，若此值大於 35%，就代表這格有車。

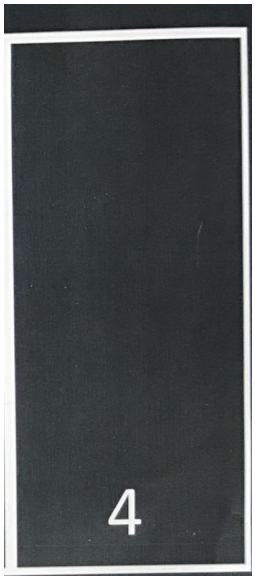


圖 25 無車圖

圖 26 現況圖

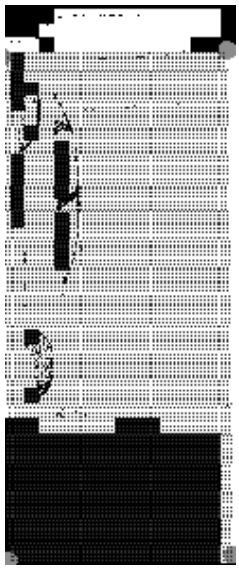


圖 27 判斷圖

(四)選擇最佳角度

為了方便說明，我們對於角度做一個簡單的定義：

1. 俯角：以水平線為 0 度，垂直於鏡頭的鉛直線與水平線之間的夾角如圖 28 所示。

2.斜角：以車格的中心點為主，畫一條平行於長邊車格線的平行線，L1 鏡頭中心點連接車格中心點之直線 L2，L1 與 L2 之間的夾角，如圖 29 所示。

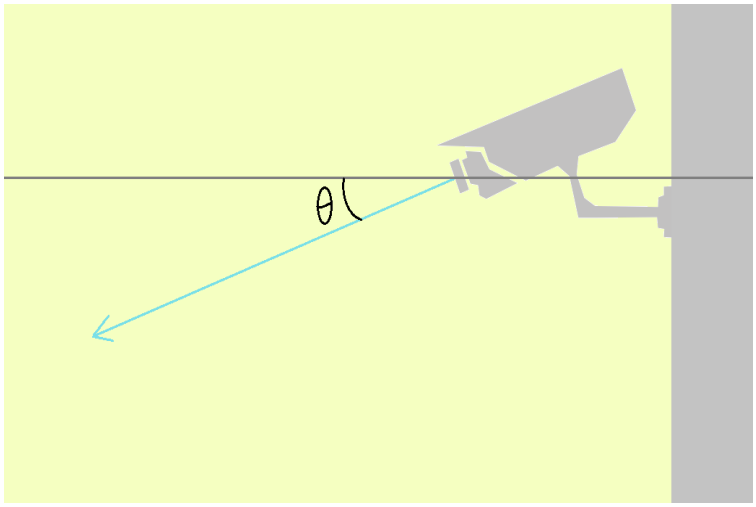


圖 28 俯角示意圖

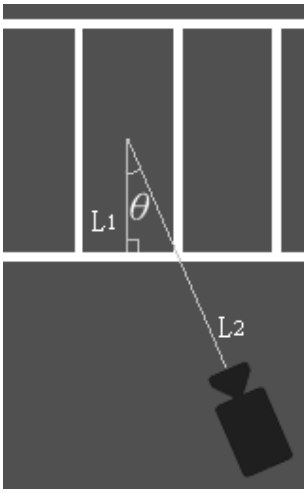


圖 29 斜角示意圖

攝影機架設時，需考量攝影機的俯角與斜角並選擇最佳的角度，有以下三個原因：

1.阻擋：如同人的視角，在有物體的情形下，攝影機同樣有阻擋視線的問題。而依據俯角角度的不同，阻擋的情形也會有所不同。如圖 30 與圖 31 所示，俯角角度為 30 度時，1 號車格中的物體阻擋到 2 號車格的面積約 2 號車格的 51.4%。當俯角 60 度時，1 號車格中的物體阻擋到 2 號車格的面積約 2 號車格的 26.3%。而斜角角度的不同，也會影響阻擋的情形，如圖 32 和圖 33。

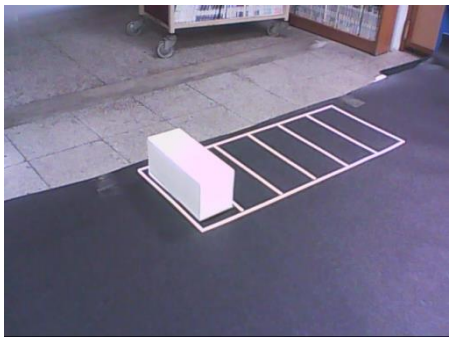


圖 30 俯角 30° 斜角 40°

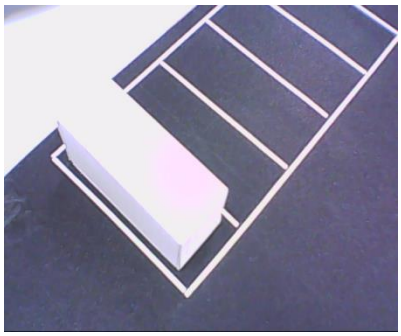


圖 31 俯角 60° 斜角 40°

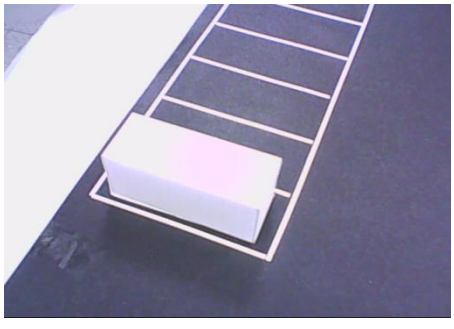


圖 32 俯角 50° 斜角 70°

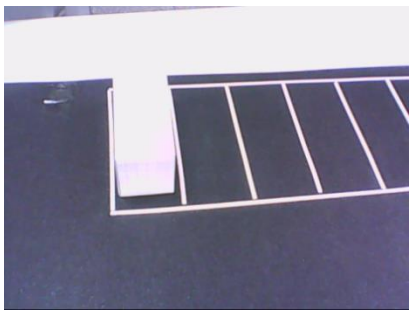


圖 33 俯角 50° 斜角 10°

2.扭曲（歪斜）：空間的遠近會影響人類的視覺，同樣的，攝影機拍攝出的線條也會呈現一定的扭曲歪斜。而依據俯角與斜角的角度不同，扭曲歪斜的程度也會有所不同，因此車格在影像中的大小也會改變。如圖 34。

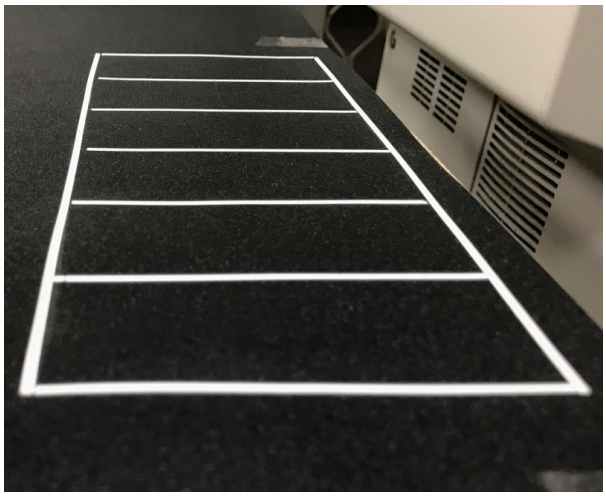


圖 34 車格扭曲歪斜

3.成本：在我們的研究中，有別於現今使用感測器偵測停車位的方式，採用影像辨識的方式來判別停車位，因此在硬體上最大的支出成本是攝影機的架設。為了盡量節省成本，在可以判讀車格的情形下，我們採取增加攝影機的可見範圍，來減少攝影機的數量。除了攝影機的架設位置外，攝影機架設時的俯角和斜角皆會影響攝影機的可見範圍。

由於各種俯角與斜角的搭配組合數量眾多，因此我們使用數學方法，先計算出因空間造成歪斜的車格可以正確地被判讀出車格的俯角和斜角範圍。接著，在這些俯角和斜角範圍中，計算出車格中的物體會阻擋到後方車格的面積比例（如圖 35），在系統可以準確的判斷出後方車格是否有車的情形下，選擇出最佳角度。

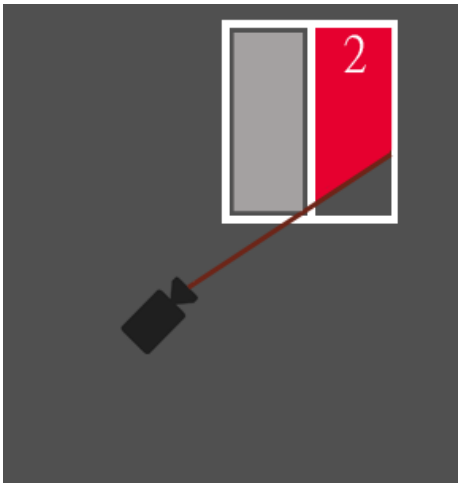


圖 35 受障礙物遮掩

四、停車導引

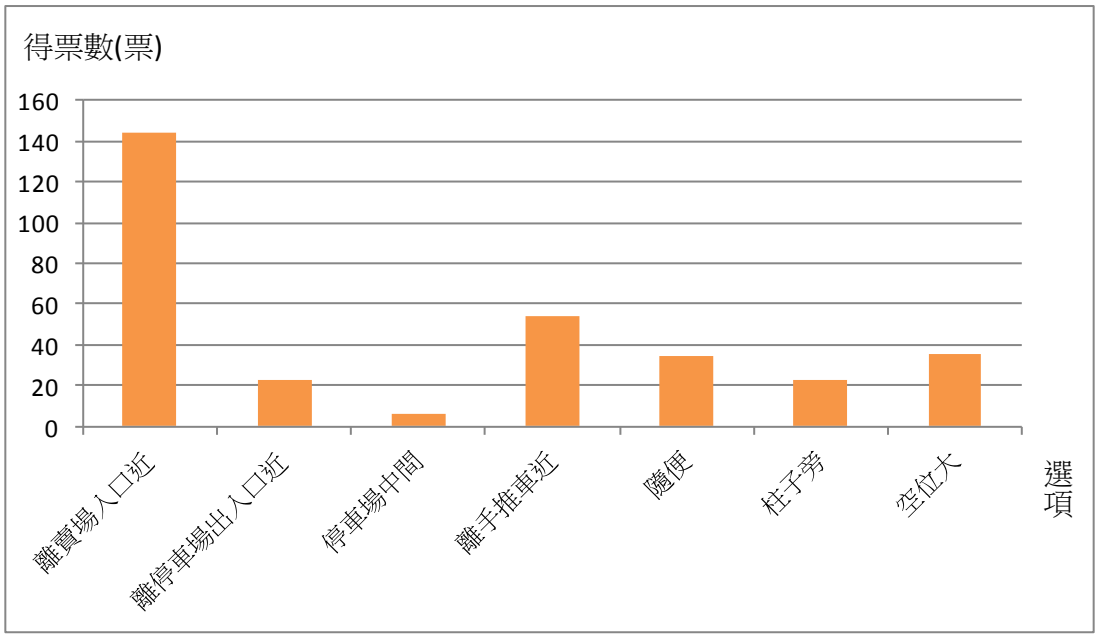
為了瞭解顧客會需要什麼樣的停車導引，在網路上進行問卷調查，問卷內容如下：

為科展研究，請幫忙填寫以下問卷。
在停車場希望能夠快速找到車位的情形下，最希望停車場提供的停車導引是?
1. → □ 離賣場入口近
2. → □ 離停車場出入口近
3. → □ 停車場中間
4. → □ 離手推車近
5. → □ 柱子旁
6. → □ 空位大
7. → □ 隨便

圖 36 問卷內容

經調查後的結果如下：

表 1 停車導引需求票數調查結果票數



調查結果顯示前 4 種顧客想要的停車導引為「離賣場入口近、離手推車近、有空位就停、車格空位大」，但在賣場內的停車場中，除了特別的停車格外(例如殘障車格)，每一格車格的大小都依相同的規格繪製，因此不提供車格空位大的導引，改成提供離賣場入口近、離手推車近、有空位就停、離停車場出入口近的停車導引。

想知道離目標最近的車格是哪一格，需要計算目標離每一格車格的最短路徑，計算最短路徑時，要考慮到障礙物的問題(不能穿越停車格、柱子等)，我們使用 flood fill 演算法，將影像中的車道進行注水。根據不同的需求，設定目標點，以「離賣場入口近」之需求為例，我們使用 flood fill 演算法，以目標點為起始節點，每個車格靠車道之短邊車格線中心點視為那個車格位置示意點，當注水到車格位置示意點時，記錄下此車格編號及距離。最後全部注完水後，可得到所有起始節點與車格之距離，將距離進行排序，再判斷車格有車的情況，將有車的車格排除，即可得到距離最短之車格。

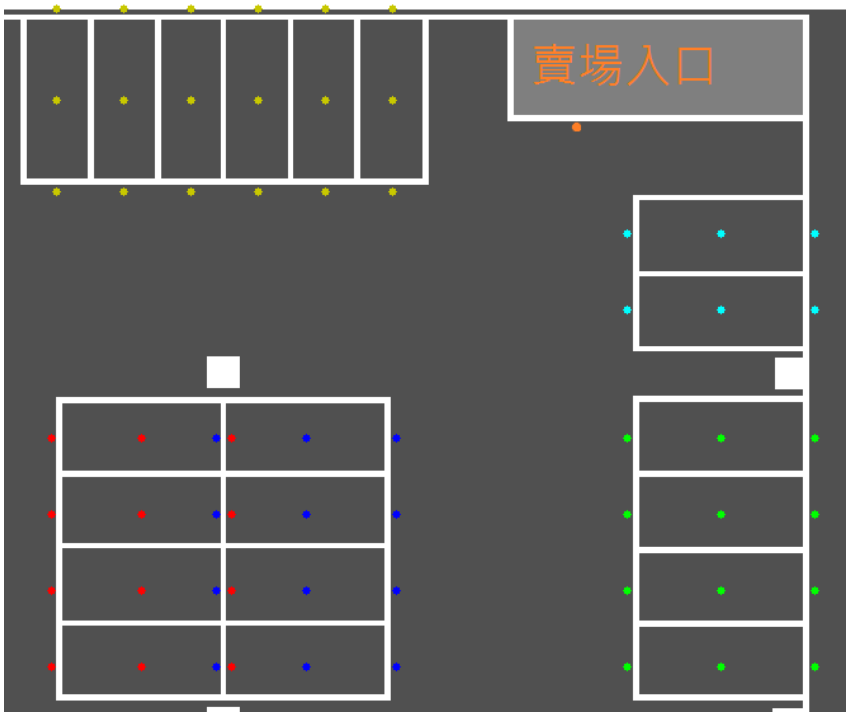


圖 37 目標點、車格位置示意點之示意圖

在程式實作導引系統過程中，原先使用廣度優先搜尋法(Breadth-First-Search，BFS)，在將導引路線連線時，發現線貼著車格的情形，使得路線顯示不清楚。為了解決這種情況，我們另外建了一張與原圖相同的影像 a，將 a 的所有停車格格內都填滿白色後，再以「1px/一個車格寬 px」的倍率進行壓縮，壓縮完再使用 BFS 將導引路線重新連線，之後將路徑轉折處資訊回傳給原圖，再將起點、轉折點、終點連線，完成的路線不太會貼著車格，顯示較為清楚。

透過每隔一段時間擷取車場資訊，辨識停車狀況後，將 4 種需求之導引，以分開顯示的方式，輸出至螢幕上同時提供給使用者參考。使用者可以在停車場入口處，透過螢幕看到最近一次擷取的車場資訊，以及經辨識後所提供的導引。

五、系統流程

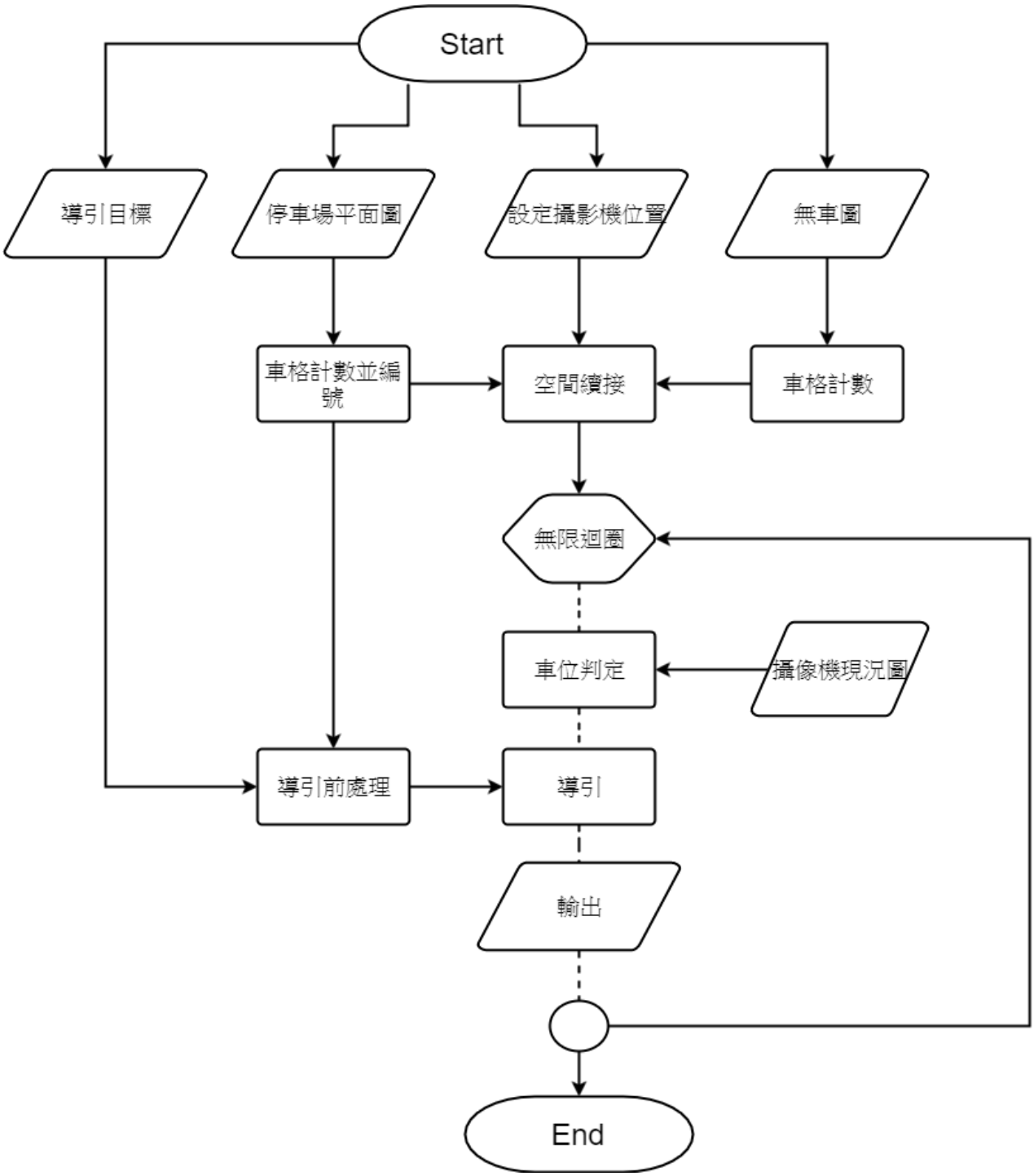


圖 38 系統流程圖

伍、研究結果

根據提出的研究方法，設計以下幾個實驗，實驗的結果如下。

一、色差閥值

(一)實驗目的

找出適當的色差閥值，讓系統在辨別有無車時有較高的正確率。

(二)實驗設計

在模型上的同一車格、俯角 90 度斜角 0 度的情形下，拍攝 200 張隨機擺放車子的圖，分別以不同的色差閥值執行辨別有無車的程式，並以自動核對的方式，找出適當的色差閥值。

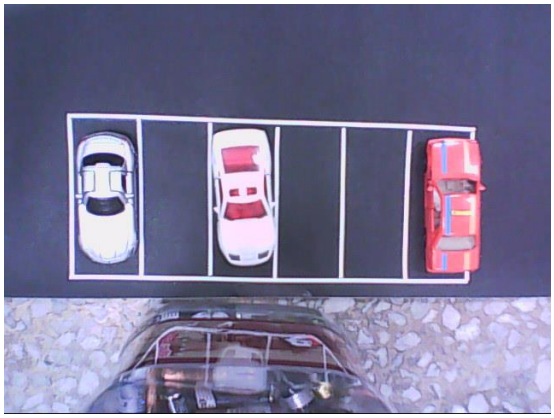


圖 39 實驗狀況

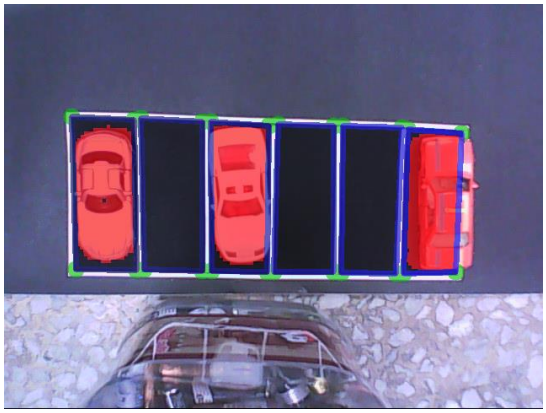


圖 40 車位判定

(三)實驗結果

先試著以 50 為區間辨別有無車。

表格 1 閥值以 50 為區間辨別有無車

色差閥值 / 結果	0	50	100
正確張數	10	200	140
錯誤張數	190	0	60
正確率	5%	100%	70%

表格 2 閥值以 10 為區間辨別有無車

色差閥值 / 結果	40	50	60
正確	192	200	198
錯誤	8	0	2
正確率	96%	100%	99%

從表格 1 發現，閥值為 50 時正確率為 100%，但閥值可能為一個範圍，再以 50 為中心加減 10 為區間，測試準確率。

從表格 2 發現，閥值下限應該在 40 到 50 之間，上限在 50 到 60 之間，再取 45、55 來試準確率。

表格 3 以閥值 45、55 辨別有無車

色差閥值 / 結果	45	55
正確張數	198	200
錯誤張數	2	0
正確率	99%	100%

從表格 3 能發現，下限應在 45 到 50 之間，上限應在 55 到 60 之間，以 1 度分別對這兩個範圍試準確率。

表格 4 以閥值 46 到 49 辨別有無車

色差閥值 / 結果	46	47	48	49
正確張數	198	200	200	200
錯誤張數	2	0	0	0
正確率	99%	100%	100%	100%

表格 5 以閥值 56 到 59 辨別有無車

色差閥值 / 結果	56	57	58	59
正確張數	200	200	200	200
錯誤張數	0	0	0	0
正確率	100%	100%	100%	100%

從表格 4 和表格 5 可看出，色差閥值範圍為 47 到 59 時，準確率為 100%。

二、攝影機的角度阻擋限制

(一)實驗目的

找出適當的俯角、斜角範圍，以進行攝影機的配置。

(二)實驗設計

先以攝影機對 2 號車格以俯角 $\theta 1^{\circ}$ 斜角 $\theta 2^{\circ}$ 拍攝一張 1 號車格沒有障礙物的圖，再以相同的角度拍攝一張 1 號車格有障礙物的圖，將兩者進行疊圖分析，計算圖片中 1 號車格障礙物遮掩到 2 號車格的面積與 2 號車格總面積的比值。$\theta 1$ 為 10,20,30,…,80，$\theta 2$ 為 0,10,20,…,80。

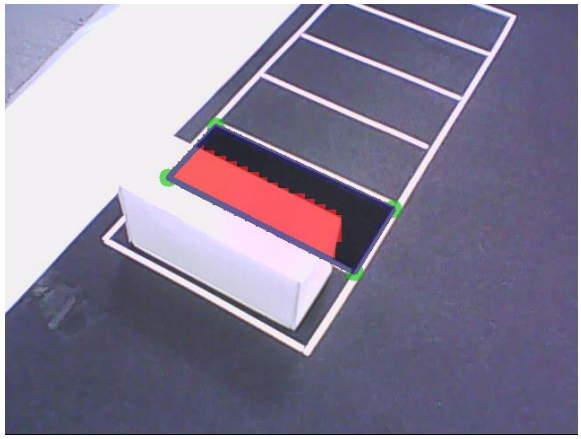
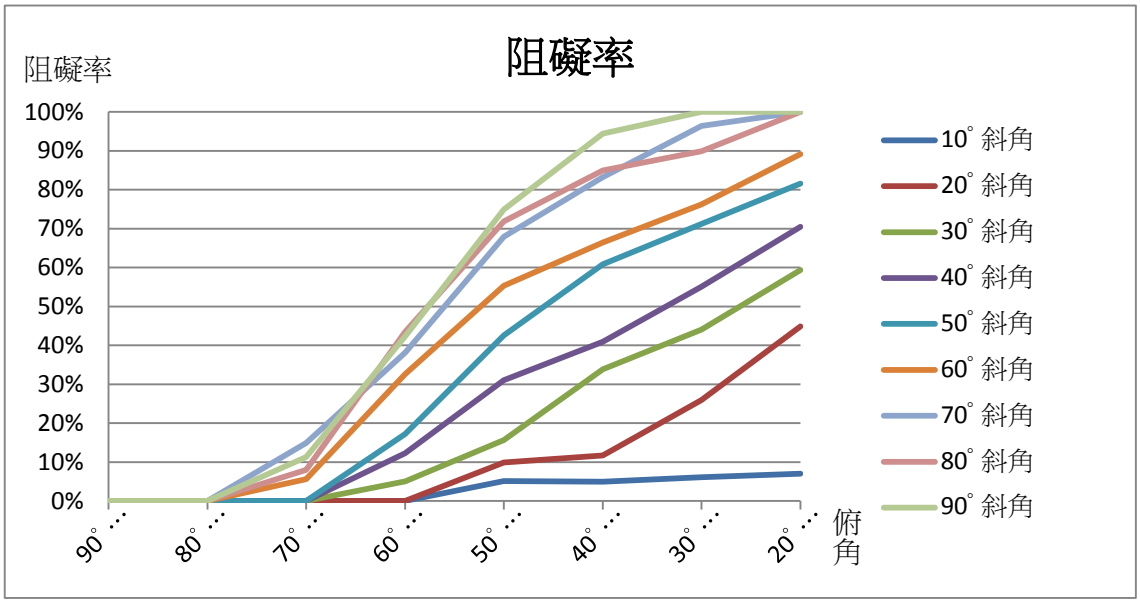


圖 41 阻礙示意圖

(三) 實驗結果

表 2 阻礙率



三、方法一找尋影像中白線的做法

(一)實驗目的

確認方法一找影像中白線的做法。

(二)實驗方法

在方法一中有兩種找直線的方法，分別為尋找同方向連續的白色向素與垂直方向色差，將二種方法分為 4 種組合如下：白色、垂直方向差、白色且具有垂直方向差、白色或具有垂直方向差，每組分別對電腦隨機產生的 200 張車格圖進行找線判別車格數，自動計算出正確率。

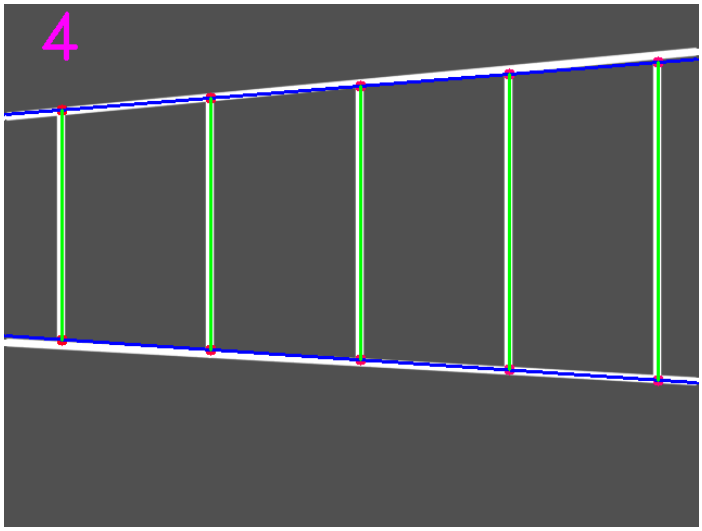


圖 42 使用白色且垂直方向差尋找結果

(三)實驗結果

表格 6 方法一中 4 種方法找車格結果

方法 / 結果	白色	垂直方向差	白色且垂直方向差	白色或垂直方向差
正確張數	144	188	200	100
錯誤張數	56	12	0	100
正確率	72%	94%	100%	50%

四、方法一找尋白線法與方法二霍夫轉換法比較

(一)實驗目的

確認找線方法，選擇準確率較高者作為本研究的找線方式。

(二)實驗方法

在模型停車場的影像 150 張，使用單一閾值做影像前處理(二值化)，再分別以兩種找線方法進行找線，比較兩者找到車格的準確率。

(三)實驗結果

表格 7 方法一找尋白線與方法二霍夫轉換法比較結果

方法 / 結果	方法一 找尋白線	方法二 霍夫轉換
正確張數	114	141
錯誤張數	36	9
準確率	77%	94%

霍夫轉換法找線準確率高於方法一，故選用方法二霍夫轉換法。

五、區域閾值的常數與正方形邊長

(一)實驗目的

在方法二中，本研究以座標(x,y)為中心，周圍一個正方形區域，將座標(x,y)的像素閥值定為 T(x,y) = μ + C(constant) * σ ，實驗確認方法二在影像前處理中所使用的常數 C 與正方形邊長的值。

(二)實驗方法

以不同的值，在實際停車場 254 張影像中以方法二尋找車格，再確認找出來的車格是否符合正確車格，並統計準確率。使用逼近的方式，在區間中找適合的值。

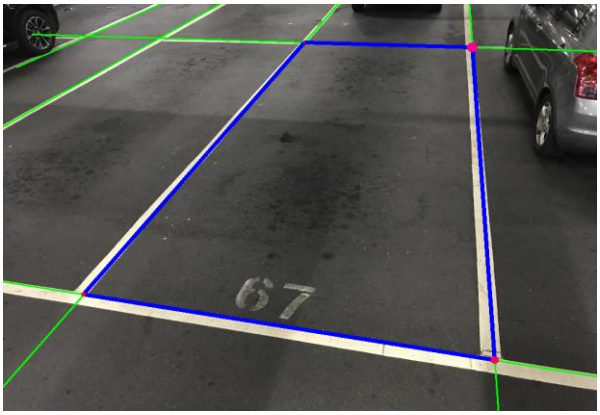


圖 43 使用區域閾值與霍夫找線找車格結果

(三)實驗結果

本次實驗的影像像素為 1008*756，先以 100 為邊長區間、0.2 為常數區間測試準確率。

表格 8 方法二以 100 為邊長區間、0.2 為常數區間測試結果

常數 / 邊長	1	1.2	1.4	1.6	1.8	2
51	56.99%	37%	21.99%	11.999%	12.99%	17%
151	62.99%	70.99%	68.99%	5.799%	50.99%	34.99%
251	69.99%	69.99%	69.99%	50.0%	44.99%	2.7%
351	70.99%	68.99%	68%	50.0%	34.99%	25.99%

邊長 151 到 351 之間最高準確率皆為 70%左右，在此範圍內，以 20 為邊長區間、0.2 為常數的區間，再次測試準確率。

表格 9 方法二以 20 為邊長區間、0.2 為常數區間測試結果

常數 / 邊長	1	1.2	1.4	1.6	1.8	2
151	68%	75.99%	68.99%	58.99%	47.99%	31.99%
171	60%	69.99%	63.99%	61%	49%	27%
191	68.99%	79.99%	67%	56%	49%	34%
211	75%	68.99%	57.99%	57.99%	50%	31%
231	75%	72%	63.99%	60%	46%	31%
251	68%	62%	75%	54%	40%	37%
271	68%	76.99%	66%	52.99%	40.99%	40%
291	73%	75%	60%	56%	43.99%	34.99%
311	67%	73%	62.99%	54%	30%	36%
331	75%	69%	63.99%	50%	41.99%	37.99%
351	62.99%	66%	64.99%	46.99%	40.99%	33%

發現邊長為 191、常數為 1.2 時，出現了最高準確率 79.99%，再以此邊長測試不同的常數的準確率。

表格 10 方法二以 20 為邊長區間、0.2 為常數區間測試結果

常數 / 邊長	1	1.1	1.15	1.2	1.3
191	75.1%	81.4%	78.3%	79.9%	74.4%

發現邊長為 191、常數為 1.1 時，出現了最高準確率 81.4%，將影像面積開根號得 873，算出邊長 191 與 873 的比值為 0.2188，以邊長為影像面積開根號乘以 0.2188、常數為 1.1 作為區域閾值。

六、單一閾值與區域閾值之比較

(一)實驗目的

確認在尋找車格線前，影像前處理中二值化的閾值。

(二)實驗方法

使用兩種不同閾值二值化後的模型停車場 150 張與實際停車場 254 張，以霍夫轉換法找車格，比較找到車格的準確率。

(三)實驗結果

表格 11 單一閥值與區域閥值比較結果

前處理方法 / 停車場	單一閾值	區域閾值
模型停車格	94%	93.33%
實際停車格	52.66%	81.4%

從結果發現，兩種不同閾值二值化模型停車場的準確率差不多，但在實際停車場上有明顯的差距，而本研究希望研究結果能實用於實際生活中，因此使用區域閾值作為本研究的二值化閾值。

七、價格比較

(一)調查目的

為了比較出本研究導引與其他硬體方式導引在硬體設備上的價格高低。

(二)調查方法

以人工方式估計本研究導引需要的攝影機數量的價格，與其他硬體方式導引需要的硬體設備和監視器的價格。

若要影像清楚且辨識準確率高，加上考量到攝影機的視角，本研究大概 3 個車格會需要一台攝影機，若是攝影機可 360 度旋轉，大約 6 格車格一台攝影機。

現今許多停車場所使用的感測器導引，每一格車格需要一個硬體感測器以及一個車位指示燈顯示車位是否空，而在每一條車道的兩端，會以互相對照的形式設置監視器。另外需要感測器資料接收器以及車位顯示器。

經過上網及電話詢價後發現，在露天拍賣上一個超音波偵測器電子元件模組約 50 元。而廠商提供的設備報價中，一個超音波車位偵測器 2700 元，一個車位指示燈 600 元，一個車位資料接收控制器 18000 元，一般攝影機(監視器)8000 元，360 度智慧型高速球攝影機 13500 元。分別以廠商提供的設備報價與線上電子元件模組來估算價格，不計入施工相關的費用。

(三)調查結果

蒐集八個室內停車場之平面圖，用來估計與比較設備數量及價格。(停車場平面圖後附於附錄)。

表格 12 硬體設備與本研究比較結果

		感測器導引				本研究導引			
數量或價格 / 停車場編號	車格數量	硬體設備數量	監視器數量	電子元件感測器價格	廠商價格	攝影機數量	360 度攝影機數量	價格	360 度價格
1	25	25	6	49250	148500	11	8	88000	108000
2	38	38	8	65900	207400	15	11	120000	148500
3	85	85	10	84250	378500	32	21	256000	283500
4	40	40	10	82000	230000	18	10	144000	135000
5	50	50	8	68000	247000	18	11	144000	148500
6	41	41	8	66050	217300	17	14	136000	189000
7	506	506	40	345300	2007800	160	80	1280000	1080000
8	42	42	7	58100	212600	19	15	152000	202500

陸、討論與應用

一、色差閥值

(一)車子與柏油路顏色相近

有些車輛與柏油路的顏色 RGB 值相近，色差閥值越大，這些車輛越無法被辨識，為了辨識這些車輛，色差閥值會有一個上限，在色差閥值實驗中，我們得知上限值為 59。

(二)柏油路顏色有些許改變

我們發現在顏色閥值過低時，系統會誤判柏油路為物體，因為攝影機在拍攝時會有誤差，同一像素的 RGB 值會有些許不同，為了容許顏色上的誤差，色差閥值會有一個下限值，在色差閥值實驗中，我們得知下限值為 47。

二、角度限制

(一)阻礙率實驗角度與現實角度

在做阻礙率實驗時，我們只對單一車格計算阻礙率，但在實際停車場中，車格多為一排，因此，我們應該要看最斜車格的角度，才能確認攝影機拍攝的影像是否符合角度限制。

(二)找線步驟 1 的角度範圍

我們希望母線連到多個子線，因此母線為車格短邊。由於使用特定角度架設攝影機，我們拍攝的影像中，車格短邊多為水平線向上 45 度到向下 45 度。因此，我們只找這段角度。

三、方法一找尋影像中白線

(一)找尋影像中白線的做法

尋找白線的 4 種方法實驗中，我們發現，垂直方向差決定了方向，白色的像素決定了位置。當只尋找連續的白色時，會找出跟畫面中白線方向不一樣的線，導致車格判斷上的失誤；只尋找連續的垂直方向差時，雖然方向與畫面中白線方向相似，但是位置有可能不在白線上。因此，找到的白線必須同時通過這兩個條件。

(二)錯誤原因

程式對白線的定義為同方向連續取白色像素且具有垂直方向色差，在畫面中的白線過粗時，就會找出一條符合上述條件的線，但是它與畫面中的白線有方向又有所不同，如下圖。如果再尋找母線時發生這種情形可能會導致母線歪斜，母線歪斜後可能會導致找不出子線，因而找不出車格。

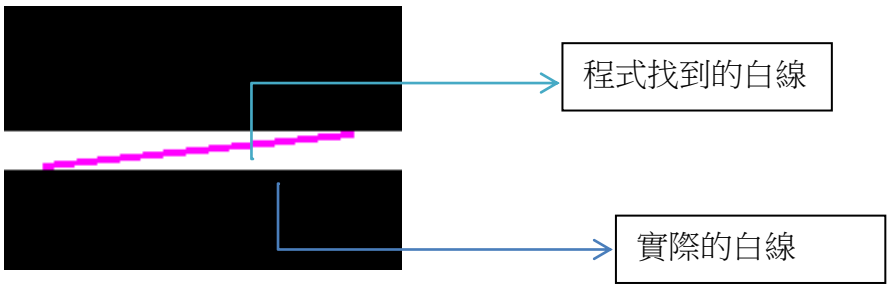


圖 44 實際的白線與系統找到的白線

(三)車格越多錯誤率越低

當車格越多時，畫面中車格越小，白線就會越細，如同上述錯誤原因，白線越細，錯誤率越低，因此車格越多時，錯誤率越低。

(四)電腦生成車格、模型車格、現實停車場車格的差別

電腦生成車格的背景無雜質，白線粗細一致，地面顏色一致；模型車格的背景有雜質，白線粗細不一，地面顏色稍有改變但反光不嚴重，主要差別是需要去背景雜質，雜質容易導致系統誤判；現實停車場車格的背景有雜質，比模型車格更多，白線粗細不一，有地面不平整的地方，使白線扭曲，有些地板的材質容易反光，導致系統誤判為白色。

(五)方法二錯誤原因

1.柱子：柱子與線一樣為白色，可能誤判。

2.大顆雜質：在開運算時，大顆雜質並不會被消除，可能會影響效率、甚至誤判。

3.線太模糊：在人眼也難以判斷的情形下，電腦無法判斷其為線。

4.線太細：太細的線會在開運算後被當作雜質侵蝕掉。

四、方法一找尋白線法與方法二霍夫轉換法

(一)正確率

由於模型停車場影像雜訊較少，找白線方法與霍夫轉換法在找線的正確率上僅略有些差異，而在實際停車場中，由於雜訊較多，找白線方法易受到影響，因此霍夫轉換法的正確率明顯優於找白線方法。

(二)時間複雜度

找白線方法找線的時間複雜度為 O(n3)，霍夫轉換法則是 O(n2)。

五、區域閾值的常數與正方形邊長

(一)區域閥值的常數

為了去除影像雜訊，使用區域閥值進行影像前處理，從實驗可看出當常數值過低時，難以去雜訊，使得車格影像辨識準確率較低。但常數值過高時，白線可能被當作雜訊去除，因此設定常數為 1.1，在實際停車場中，車格準確率最高。

(二)正方形邊長

以區域閥值進行影像前處理，當正方形邊長長度過短時，無法去掉雜訊，因此準確率較低。而正方形邊長長度過長時，則無法解決照光不均問題，會使得無法準確的辨識車格。因此根據實驗，設定正方形邊長長度為影像面積開根號乘以 0.2188。

六、單一閾值與區域閾值

單一閥值是對於每一張影像皆使用同一閥值進行二值化來簡化圖片，而在實際停車場影像，由於照光不均，使得地面亮度不均勻，而不同的停車場影像，地面顏色強度也不同，在車格辨識時會有干擾，而使得辨識準確率降低，因此提出區域閾值來解決此問題。區域閾值可解決此問題，而單一閾值無法處理。

七、價格比較

電子元件的價格低廉，因此總價也非常的便宜，但感測器模組的訊號接收以及裝設等方式與設備需要再進一步的研究。本研究的導引在與停車場內架設的監視器結合的情況下，雖然停車場內需架設的監視器數量增加，價錢上較電子元件的價格高上許多，但本研究可同時提高停車場內安全性，另外，相較於廠商提供的導引系統，本研究價錢上較低。

柒、結論

本研究解析影像檔案後，將影像使用區域二值化的方式簡化，再經過膨脹、侵蝕，以及開運算、閉運算之處理，減少影像中雜訊之干擾。使用霍夫轉換法偵測直線來找出影像中的白色線，以便辨識車格，並將車格計數，透過空間接續方法得到並整合停車場之資訊。本研究另設計角度實驗，希望達到盡量最小成本的目的。讓電腦自動判斷停車場現況，停車場之現況辨識出來後，提供給使用者，讓使用者能夠快速瞭解停車場狀況，並提供客製化導引。

未來展望有

一、 對常來的客戶提供手機 APP，讓使用者能夠使用手機在進入停車場前，更清楚瞭解停車場現況，提供更客製化的導引。

二、 使用可以 360 度旋轉的攝影機，在辨識車格時，每幾度就拍攝一張影像，若影像中有車格，就記錄下此角度，在判斷車格內有無車時，會在記錄到的角度拍攝一張影像，判斷那角度的車格內有無車，回傳給導引系統，提供導引給使用者。這麼做可以大大降低攝影機的數量，達到降低成本的效果。

三、 統計停車場不同停車格在不同時段的使用率，使用率低的部分可以做其他使用。

四、 增加車牌辨識功能，讓車主在找車時，可以藉由車牌收尋車的位置，同時也可以提升車子的安全性。

捌、參考文獻

1. Paul Deitel、Harvey Detiel (2013)．C 程式設計藝術(第七版)．新北市：全華

2. 毛星雲、冷雪飛．OpenCV 程式設計參考手冊．台北市：松崗

3. 賴岱佑、郭忠義、黃福助．數位影像處理技術手冊第二版．台北市：松崗

4. OpenCV 教學| 阿洲的程式教學．取至：http://monkeycoding.com/?page_id=12

5. Myths 的个人博客 PNG 文件格式详解 (2015 年 12 月 8 日)．取至：
   https://blog.mythsman.com/2015/12/08/1/

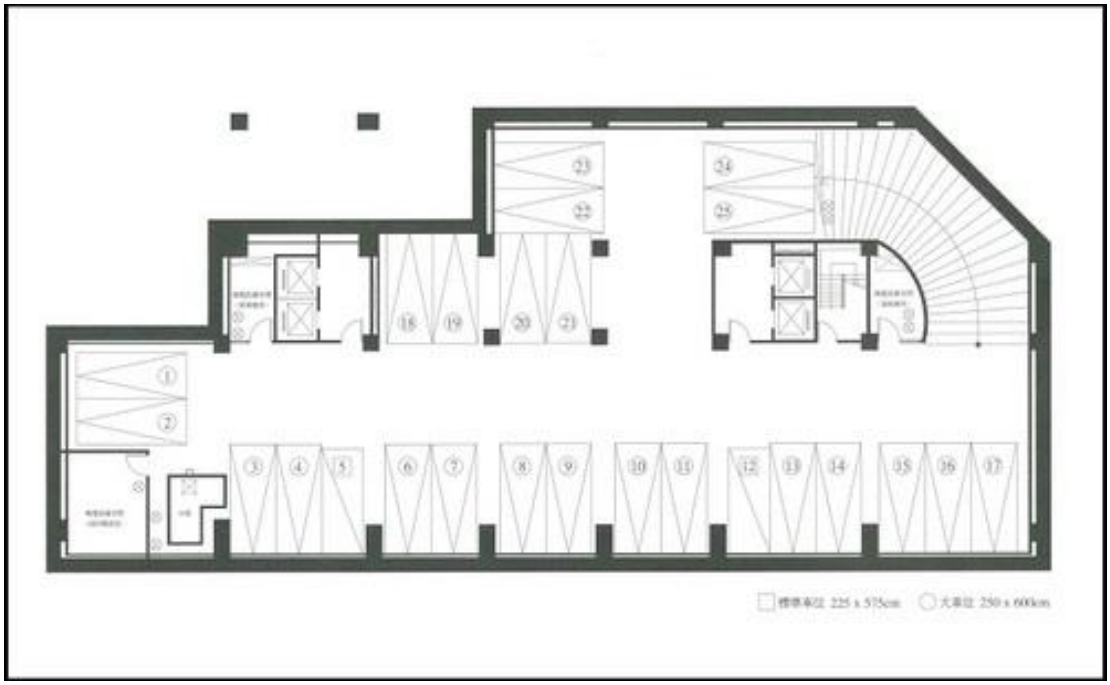
6. JPEG 文件编/解码详解 (2007 年 7 月 28 日)．取至：
   http://blog.csdn.net/lpt19832003/article/details/1713718

7. PNG 文件格式详解 (2008 年 8 月 6 日)．取至：
   http://blog.csdn.net/bisword/article/details/2777121

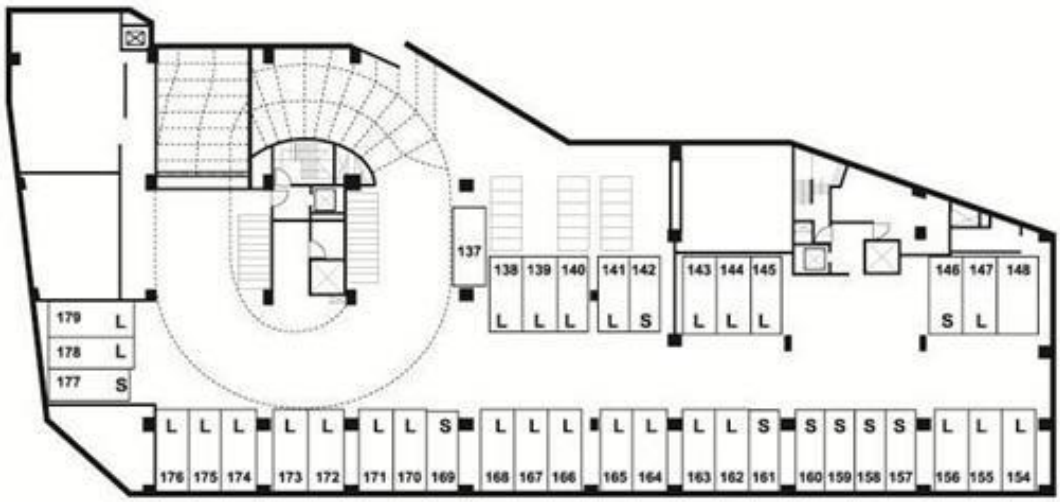
附錄

附錄一、 研究結果中價格比較的停車場

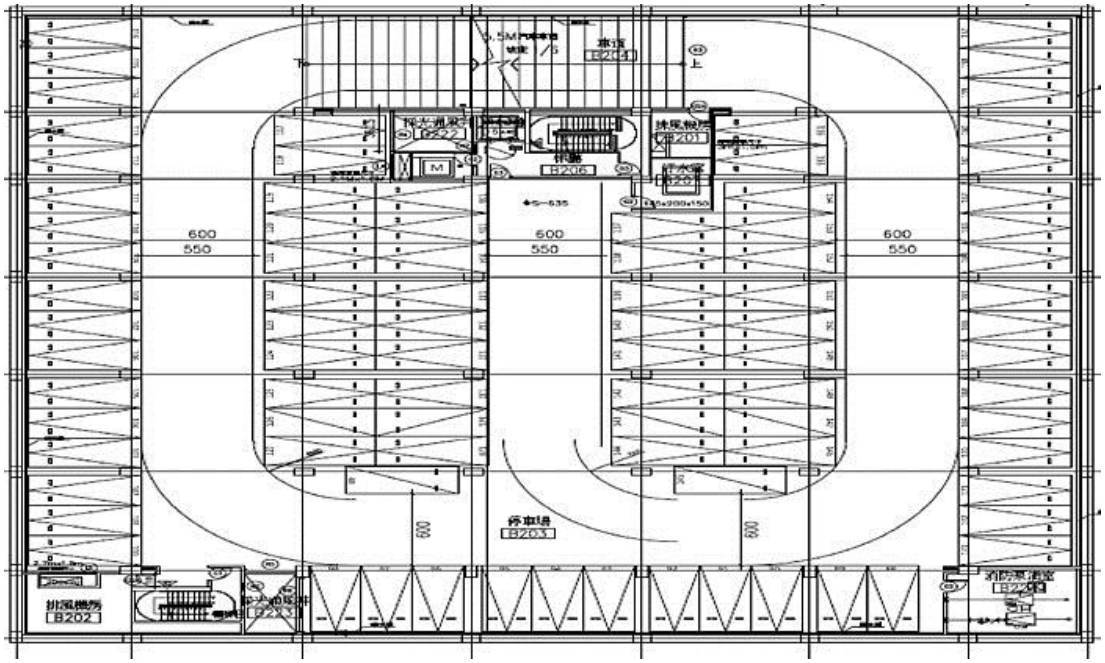
編號 1 停車場



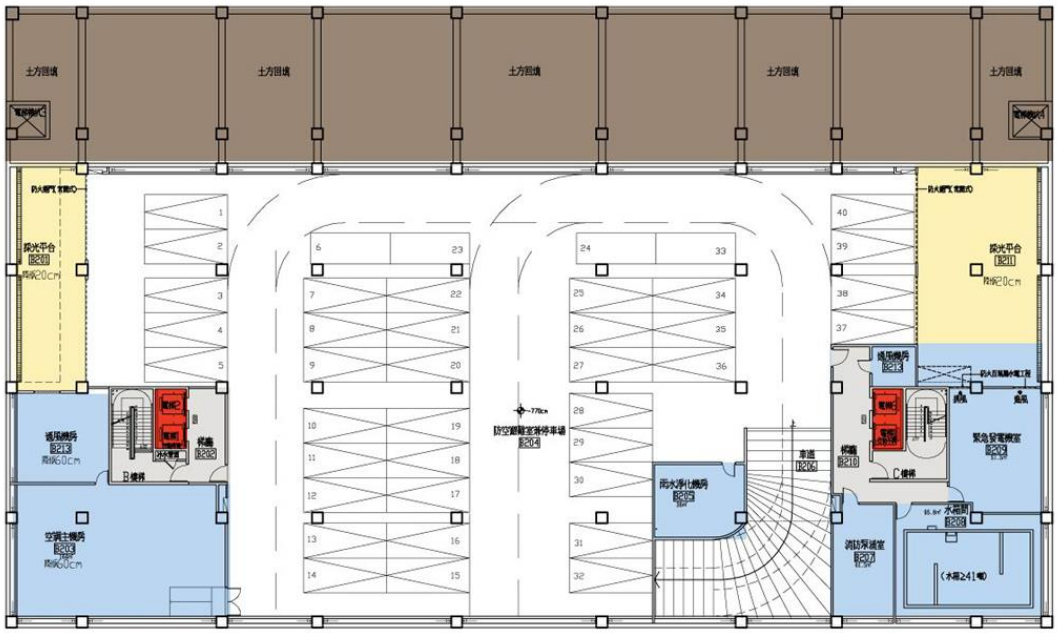
編號 2 停車場



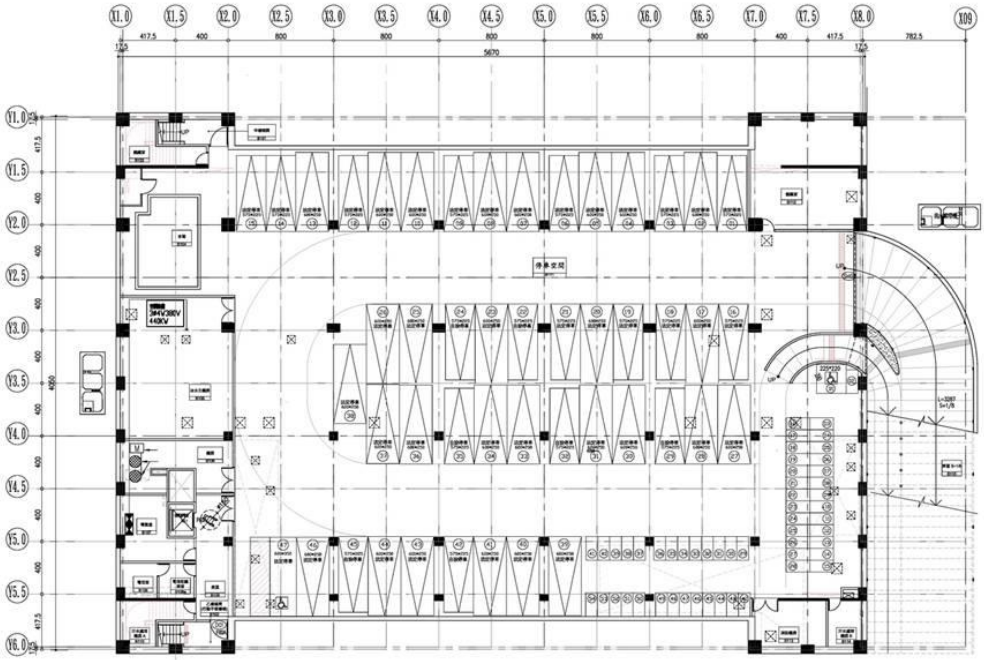
編號 3 停車場



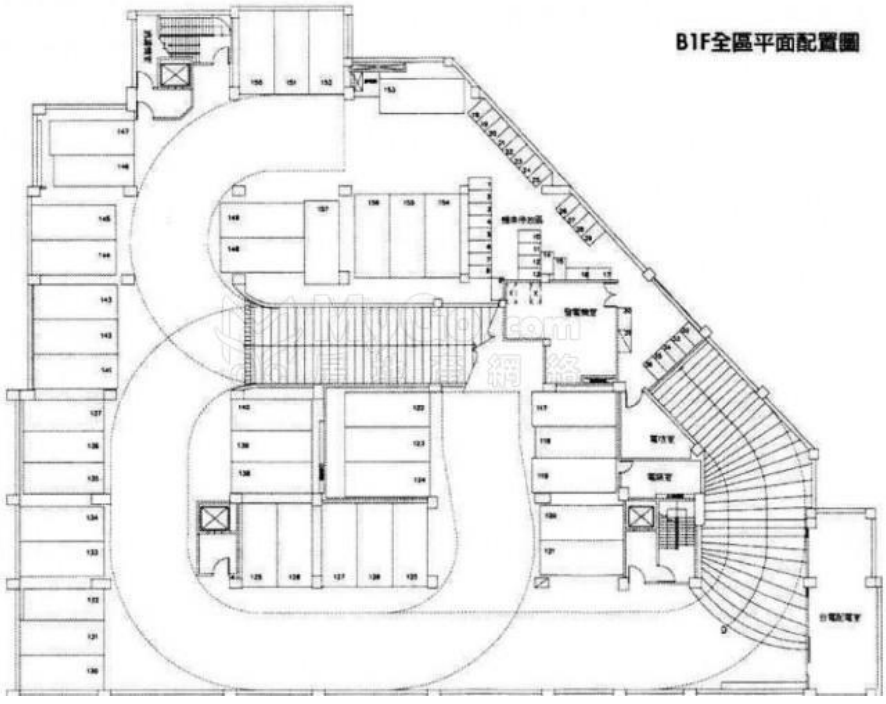
編號 4 停車場



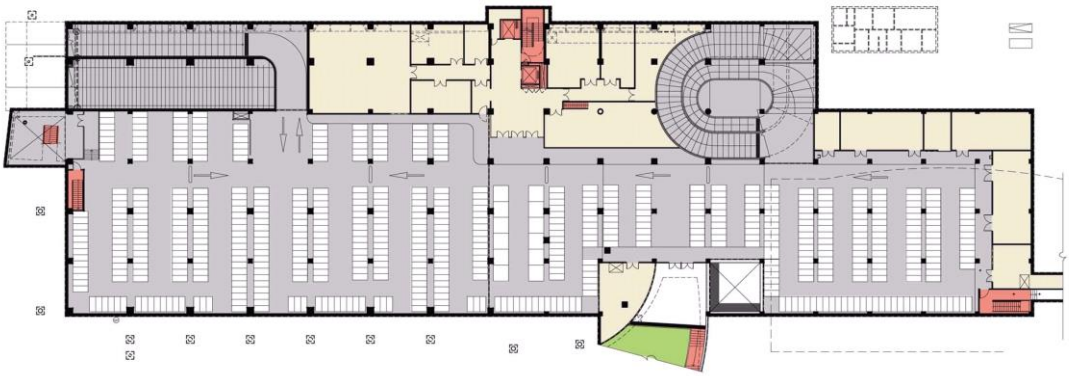
編號 5 停車場



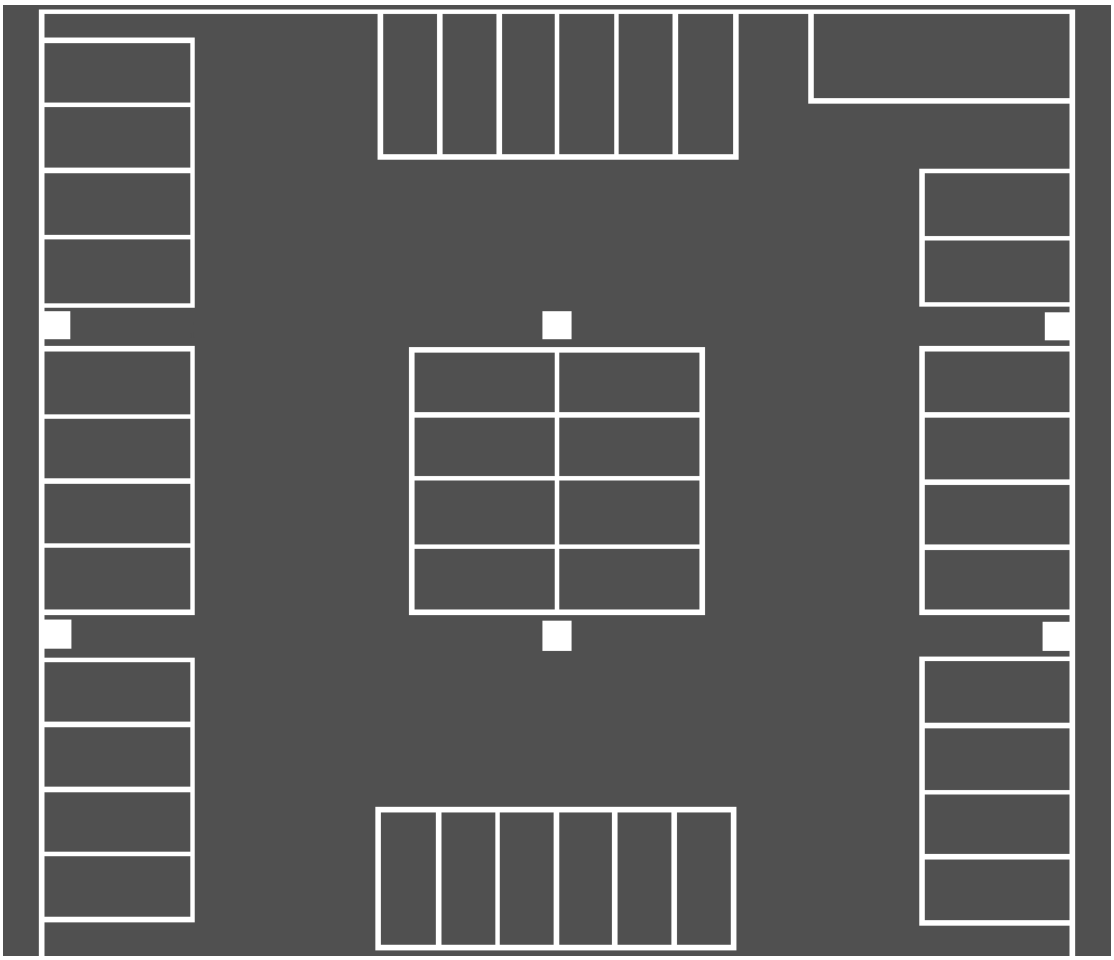
編號 6 停車場



編號 7 停車場



編號 8 停車場



附錄二、 程式實作

```
//  Created by 鍾安 on 2017/9/10.
//  Copyright © 2017 年 鍾安. All rights reserved.



//法一找線
int find_line(Mat &img,const double theta1,const double theta2,int jump1,int jump_line
              ,ParaT stand180_line,ParaT stand90_line
              ,const double longth_limit,const double line_r,const double
toleration_distant,const double big_toleration_distant
              ,vector<ParaT> &vec_line,const int model)//model0=G,model1=W
{
    ParaT _line;        // main line in find_line
    _line.set(theta1,Coordinate(0,0),0);
    ParaT hor_line;        // horizontal line
    ParaT ver_line;        // vertical line
    ParaT temp;
    Coordinate p(0,0);    // main point to find line
    Coordinate p1,p2,p3;
    int has_line=0;
    int merge_line=0;
    int c,c1,c2;              // eat color
    Coordinate vir_stand;      // virtual stand
    int vir_stand_distant;    // virtual "stand" distant
    int test_n=0;      //debug
    bool stand_bool;        // for stand_loop
    int has_new_line = 0;

    while( _line.theta > theta2 )    /// theta
    {
//            img1.copyTo(img_show);
//            printf("theta\n");

        vir_stand_distant=0;

        if (_line.theta > 90    || abs(_line.theta-90)<0.000001 )    // set p
        {
            p = stand180_line.stand;
            stand_bool = ( vir_stand_distant <= stand180_line.distant );
        }
        else if( _line.theta < 90|| abs(_line.theta-0)<0.0000001)
        {
            p = stand90_line.stand;
            stand_bool = ( vir_stand_distant <= stand90_line.distant);
        }

        vir_stand = p;
        _line.stand = p;
```

```
        _line.distant = 0;

//          printf("stand\n");

        while( stand_bool )    /// stand
        {
            has_new_line = 0;

            if(_line.theta>90 || abs(_line.theta-90)<0.00001 )    // get vir_stand
            {
                vir_stand = stand180_line.point(vir_stand_distant);
            }
            else if( _line.theta <90 || abs(_line.theta)<0.0000001 )
            {
                vir_stand = stand90_line.point(vir_stand_distant);
            }
            //end get_vir_stand

            if( vir_stand.x>img.rows )        // _line get stand
            {
                _line.stand.y=round( ( img.rows-vir_stand.x )*tan(pi/180*_line.theta) );
                _line.stand.x=img.rows-1;
            }
            else if(vir_stand.x<0)
            {
                if( abs(_line.theta-90) <= 0.000001 )      // cut 90
                    _line.stand.y = img.cols-1;
                else
                    _line.stand.y=round( abs(vir_stand.x)*tan(pi/180*_line.theta));

                _line.stand.x=0;
            }
            else if(vir_stand.y>img.cols)
            {
                _line.stand = vir_stand;
                _line.stand = Coordinate( _line.x(img.cols-1), img.cols-1);
            }
            else if(vir_stand.y<0)
            {
                _line.stand = vir_stand;_line.stand = vir_stand;
                _line.stand = Coordinate( _line.x(1) , 1);
            }
            else
            {
                _line.stand = vir_stand;
            }
            //end get_line

            _line.distant = 0;
            p = _line.end_point();
```

```
                while( is_inlaw(img,p) && !( _line.distant >= big_toleration_distant &&
has_new_line==0 ) )    /// distant
                {
                        p = _line.end_point();

                        if( is_inlaw(img,p) )        // draw
                        {
//                                      p.draw(img_show,Scalar(0,0,255),1);

                                p1 =
_line.vertical_point(_line.distant,round(line_r*0.25));//line_r*0.25
                                p2 = _line.vertical_point(_line.distant,-round(line_r*0.25));
                                c1 = p1.read_color(img);
                                c2 = p2.read_color(img);
                                c    = p.read_color(img);
                                bool test_line_bool;

                                if(model==0)
                                {
                                        test_line_bool = (c-c1)>200 || (c-c2)>200;
                                }
                                else if(model==1)
                                {
                                        test_line_bool = c>250;
                                }
                                else if(model==2)
                                {
                                        test_line_bool = (c>250) && ( (c-c1)>200 && (c-c2)>200 );
                                }
                                else if(model==4 )
                                {
                                        test_line_bool = (c>250) || ( (c-c1)>200 && (c-c2)>200 );
                                }
                                else
                                {
                                        printf("\nFUCK YOU \n");
                                }

                                if( test_line_bool ) //test_line
                                {
                                        _line.distant -= jump1/2;
                                        p = _line.end_point();

                                        bool gradient_limit;
                                        bool white_limit;
                                        hor_line                = _line;
                                        hor_line.stand      = p;
                                        hor_line.distant = 0;
                                        ver_line                = _line;
                                        ver_line.stand      = p;
                                        ver_line.distant = 0;
```

```
                                ver_line.theta      = _line.theta + 90;
                                int vir_distant = 0;
                                int vir_distant2 = 0;

                                while( true ) //check vertical gradient
                                {
                                      vir_distant += 1;
                                      vir_distant2 += 1;
                                      hor_line.distant += 1;
                                      p    = hor_line.end_point();
                                      p1 =
hor_line.vertical_point(hor_line.distant,round(line_r*0.25) );//
                                      p2 =
hor_line.vertical_point(hor_line.distant,-round(line_r*0.25) );
                                      c1 = p1.read_color(img);
                                      c2 = p2.read_color(img);

//                                       gradient_limit =( abs(p.read_color(img)-c1)<200 &&
abs(p.read_color(img)-c2)<200 );
//                                       white_limit = ( (p.read_color(img)<200)    );

                                      if( !is_inlaw(img,p) || !is_inlaw(img,p1) || !is_inlaw(img,p2) )
// image.size limit
                                            break;

                                      if(c1==-1||c2==-1)// image.size limit
                                            break;

                                      if(model==0)
                                      {
                                          // vertical error limit and white limit
                                            if( (p.read_color(img)-c1)<200 &&
(p.read_color(img)-c2)<200 )
                                            {
                                                  if(    vir_distant >= round(toleration_distant)    )
                                                  {
                                                        hor_line.distant -= vir_distant;
                                                        break;
                                                  }
                                            }
                                            else
                                            {
                                                  if(c1!=-1 || c2!=-1)
                                                        vir_distant = 0;
                                            }
                                      }
                                      else if(model==1)
                                      {
                                            //white limit
                                            if( p.read_color(img)<250 )
                                            {
```

```
                    if(    vir_distant2 >= round(toleration_distant)    )
                    {
                        hor_line.distant -= vir_distant2;
                        break;
                    }

                }
                else
                {
                    if(c1!=-1 || c2!=-1)
                        vir_distant2 = 0;
                }
            }
            else if(model==2)
            {
                if( (p.read_color(img)-c1)<200 ||
(p.read_color(img)-c2)<200 )
                {
                    if(vir_distant > round(toleration_distant))
                    {
                        hor_line.distant -= vir_distant;
                        break;
                    }
                }
                else
                {
                    vir_distant=0;
                }

                if( p.read_color(img)<250 )
                    break;
            }
            else if(model==3)
            {
                //only vertical error limit
                if( abs(p.read_color(img)-c1)<200 &&
abs(p.read_color(img)-c2)<200 )
                {
                    if(    vir_distant >= round(toleration_distant)    )
                    {
                        hor_line.distant -= vir_distant;
                        break;
                    }
                }
                else
                {
                    if(c1!=-1 || c2!=-1)
                        vir_distant = 0;
                }
            }
            else if(model==4)
```

```
                {
                    if( abs(p.read_color(img)-c1)<200 &&
abs(p.read_color(img)-c2)<200 && p.read_color(img)<250 )
                    {
                        if(    vir_distant >= round(toleration_distant)    )
                        {
                            hor_line.distant -= vir_distant;
                            break;
                        }
                    }
                    else
                    {
                        if(c1!=-1 || c2!=-1)
                            vir_distant = 0;
                    }
                }
                else
                {
                    printf("\nFUCK YOU,there's no model:%d\n",model);
                    system("pause");
                }

            }

            if(hor_line.distant > longth_limit)        ///is line?
            {

                vec_line.push_back(hor_line);
                has_new_line = 1;

                ///merge
                merge_line =0;
                for(vector<ParaT>::iterator i=vec_line.begin();
i<vec_line.end(); i++)
                {
                    for(vector<ParaT>::iterator j=i+1; j<vec_line.end();)
                    {
                        merge_line =
merge_cut( *i,*j,(double)line_r,line_r*10);

                        if(merge_line==0)
                            j++;
                        else
                            vec_line.erase(j);
                    }
                }
                ///end merge

            }/// end is_line?

            _line.distant += hor_line.distant+jump_line;
```

```
                    }///end test_line

                }//end draw

                    _line.distant += jump1;
                }///end distant



                vir_stand_distant++;

                if(_line.theta > 90 || abs(_line.theta-90)<0.000001 )    // stand_bool
                    stand_bool = ( vir_stand_distant <= stand180_line.distant );
                else if( _line.theta < 90 || abs(_line.theta)<0.0000001 )
                    stand_bool = ( vir_stand_distant <= stand90_line.distant);

            }///end stand

            _line.theta -= 0.4;
        }///end theta
        return 0;
}
```

```
/**
 *  法二找線
 *  function FindLine
 *  \brief: useing HoughLine, find the line and span then;
 *  \param img : the input img
 *  \param outputVecLine : the output lines
 *  \param binaryGate : the threshold
 *  \param lineLengthLimit : the limit of line's length
 *  \param lineGapLimit : the limit of the distance between  two lines
 */
void FindLine( Mat& img, vector<CLine>& outputVecLine, const int blockSize, const double
constant, const float lineLengthLimit, const float lineGapLimit)
{
	Mat imgg( img.rows, img.cols, CV_8UC1);

//	char testOutputName[40] = {0};

	for (int i = 0; i < img.cols; i++) {
		for (int j = 0; j < img.rows; j++) {
			char g,r,b;

			b = img.at<Vec3b>(j,i)[0];
			g = img.at<Vec3b>(j,i)[1];
			r = img.at<Vec3b>(j,i)[2];

			imgg.at<uchar>(j,i) = 0.299*r + 0.58*g + 0.114*b;
		}
	}

	Binary(imgg, imgg, blockSize, constant);

	//preprocess
	Mat erdoeImg, dilateImg;
	Mat element;

	element = getStructuringElement( MORPH_RECT, Size(2,2));
	morphologyEx(imgg, imgg, MORPH_OPEN, element);

	//edge
	element = getStructuringElement( MORPH_RECT, Size(5,5));
	erode(imgg, erdoeImg, element);
	dilate(imgg, dilateImg, element);
	imgg = dilateImg - erdoeImg;

	element = getStructuringElement( MORPH_RECT, Size(3,3));
	morphologyEx(imgg, imgg, MORPH_CLOSE, element);
	//end preprocess

	vector< Vec4i > vec_linep;
```

```
vector<CLine> vecLines;

//finde line
//vote:650
//theta step:0.01
//pixle step:5
HoughLinesP(imgg, vec_linep, 4, 0.01, 500, lineLenthLimit,    lineGapLimit);//
for ( int i = 0; i < vec_linep.size(); i++) {

    Vec4i vecp = vec_linep.at(i);
    CLine templine;

    templine.SetStand( CPoint( vecp[0], vecp[1]) );
    templine.MoveEndPoint( CPoint( vecp[2], vecp[3]) );
    templine.SetThetaProperly();
    //
    if (templine.GetTheta() >= 180 || abs( templine.GetTheta() - 180) <= 0.01) {
        templine.Rotation(-180);
    }

    vecLines.push_back( templine );
}
//end find line


//if NULL
if ( vecLines.size() == 0 ) {
    printf("ERROR from FindLine : the size of vecLines is 0\n");
    printf("QUITE!\n");
    return;
}

//merge line
/*不斷的 merge lines that they were overlapped, unti there's No more lines were
overlapped*/

int mergeTimes;

do {
    mergeTimes = 0;

    for ( int i = 0; i < vecLines.size(); i++) {
        CLine templine = vecLines.at(i);

        for ( int j = i; j < vecLines.size(); j++) {
            CLine testline = vecLines.at(j);

            if ( j != i ) {

                if ( MergeLine( templine, testline, lineGapLimit * 2.5, 9.5) ) {
```

```
                        mergeTimes++;
                        vecLines.at(i) = templine;
                        vecLines.erase( vecLines.begin() + j);
                        j--;
                    }
                }
            }

            //add distant
            vecLines.at(i).AddDistnat( vecLines.at(i).GetDistant() * 0.3 );      //÷2
        }
} while ( mergeTimes > 0);
//end merge


//output
outputVecLine = vecLines;}
```

```
/**
 * 建車格
 *   function MakeGrid
 *   @brief:
 *   @param inputVecLines : input line
 *   @param outputVecGrid : output polygon(Grid)
 *   @param areaLimit : the limit of grid's area
 *   @param ar_uperLimit : the uper limit of aspect ratio, lenth/width
 *   @param ar_lowerLimit : the lower limit of asepct ratio, lenth/width
 *   @param ule_uperLimit : the uper limit of ratio that is uper edge / lower edge(上底除下底)
 *   @param ule_lowerLimit : the uper limit of ratio that is uper edge / lower edge
 */
void MakeGrid( vector<CLine>& inputVecLines, vector<CPolygon>& outputVecGrid, const double areaLimit, const double ar_lowerLimit, const double ar_uperLimit, const double ule_lowerLimit, const double ule_uperLimit)
{
	/*make grid*/
	vector<CLine> vecLines = inputVecLines;
	vector<CLine> vecRLines;
	vector<CLine> vecCLines;
	//@param vecRLine : Row Line in vector
	//@param vecCLine : colunm line in vector


	for (int i = 0; i < vecLines.size(); i++) {
		vecLines.at(i).SetThetaProperly();

		CLine line = vecLines.at(i);

		if ( (line.GetTheta() >= 40 && line.GetTheta() <= 140) || (line.GetTheta() <= 320 && line.GetTheta() >= 220) ) {
			vecCLines.push_back(line);
		}
		else {
			vecRLines.push_back(line);
		}
	}


	//srot row line
	for (int i = 1; i < vecRLines.size(); i++) {
		CPoint midPointI = vecRLines.at(i).GetMidPoint();
		CPoint midPoint1I = vecRLines.at(i-1).GetMidPoint();

		if ( midPointI.Get_y() <= midPoint1I.Get_y() ) {
			int j = i;
```

```
            while ( j > 0 ) {
                CPoint midPointJ = vecRLines.at(j).GetMidPoint();
                CPoint midPoint1J = vecRLines.at(j-1).GetMidPoint();

                if ( midPointJ.Get_y() >= midPointI.Get_y() && midPoint1J.Get_y() <=
midPointI.Get_y()) {
                    break;
                }
                j--;
            }
            CLine temp = vecRLines.at(i);
            vecRLines.erase( vecRLines.begin() + i);
            vecRLines.insert( vecRLines.begin() + j, temp);
        }
    }


    //sort col line
    for (int i = 1; i < vecCLines.size(); i++) {
        CPoint midPointI = vecCLines.at(i).GetMidPoint();
        CPoint midPoint1I = vecCLines.at(i-1).GetMidPoint();

        if ( midPointI.Get_x() <= midPoint1I.Get_x() ) {
            int j = i;

            while ( j > 0 ) {
                CPoint midPointJ = vecCLines.at(j).GetMidPoint();
                CPoint midPoint1J = vecCLines.at(j-1).GetMidPoint();

                if ( midPointJ.Get_x() >= midPointI.Get_x() && midPoint1J.Get_x() <=
midPointI.Get_x()) {
                    break;
                }
                j--;
            }
            CLine temp = vecCLines.at(i);
            vecCLines.erase( vecCLines.begin() + i);
            vecCLines.insert( vecCLines.begin() + j, temp);
        }
    }

    //remove vecLines
    vecLines.clear();

    //if NULL
    if (vecRLines.size()==0) {
        printf("ERROR: size of vecRLine is 0\nQUITE!\n");
        return;
    }
    if ( vecCLines.size()==0) {
        printf("ERROR: size of vecCLine is 0\nQUITE!\n");
```

```
        return;
    }

    /*make grid*/
    /*
     make grid method:

          FOR baseLine = vecRLine.at( from 0 to size )
               vecBaseCLine = all the Line that intersect to baseLine
               FOR baseCLine = vecBaseCLine.at( from 0 to size )
                    FOR RLine = vecRLine.at( from 0 to size )
                         IF RLine is intersect to baseCLine AND dirction > 0
                              FOR tempBaseCLine = vecBaseCLine.at( from 0 to size)
                                   IF RLIne is intersect to tempBaseCLine AND direction > 0
                                        tempPolygon =    baseLine, baseCLine, RLine,
tempBaseCLine
                                        IF tempPolygon is inlow
                                             Push tempPolygon into vecGrid
                                        END If
                                   END IF
                              END FOR
                         END IF
                    END FOR
               END FOR
          END FOR

     make grid method
     */


    vector<CPolygon> vecGrid;

    for (int i = vecRLines.size() -1; i >= 0; i--) {

        //if build a grid secced
        if ( i <= vecRLines.size() - 1 && vecGrid.size() > 0) {
            break;
        }


        CLine baseLine = vecRLines.at(i);
        vector<CLine> vecBaseCLine;
        //@param baseLine : base line for car grids
        //@param vecBaseLine : the vecLines collected all the line that intersect to baseLine

        //find all the Cline that intersect to baseLine
        for (int j = 0; j < vecCLines.size(); j++) {
            if ( IsIntersected( vecCLines.at(j), baseLine)) {
                vecBaseCLine.push_back( vecCLines.at(j));
            }
        }
```

```
        //if NULL
        if ( vecBaseCLine.size() <= 1 ) {
            //the number of CLine which intersect to baseLine is 0
            continue;
        }


        /*make grid */

        //for each baseCLine
        for (int i = 0; i < vecBaseCLine.size(); i++) {
            CLine baseCLine    = vecBaseCLine.at(i);
            int buildGridTimes = 0;

            //for each RLine that intersect to baseCLine
            for (int j = vecRLines.size() -1; j >= 0; j--) {

                //if build
                if ( j <= vecRLines.size() -1 && buildGridTimes >= 1) {
                    break;
                }


                CLine RLine = vecRLines.at(j);

                //is intersect
                if (    IsIntersected( baseCLine, RLine) && RLine != baseLine ) {
                    bool directionLimit = baseLine.GetMidPoint().Get_y() -
RLine.GetMidPoint().Get_y()>0;

                    if (directionLimit) {

                        //for others baseCLine that intersect to RLine
                        for (int k = 0; k < vecBaseCLine.size(); k++) {
                            CLine tempBaseCLine = vecBaseCLine.at(k);       //temp
BaseCLine
                            bool dirctionLimit = tempBaseCLine.GetMidPoint().Get_x() -
   baseCLine.GetMidPoint().Get_x() > 0;

                            if (   IsIntersected( tempBaseCLine, RLine)
 && tempBaseCLine != baseCLine
 && dirctionLimit )    {

                                CPolygon tempPoly;

                                tempPoly.push_back( IntersectPoint( RLine,
baseCLine));
                                tempPoly.push_back( IntersectPoint( baseLine,
baseCLine));
                                tempPoly.push_back( IntersectPoint( baseLine,
```

```
tempBaseCLine));
                                        tempPoly.push_back( IntersectPoint( tempBaseCLine,
RLine));

                                        //ratio limit
                                        CPoint p[4];
                                        double aspectRatio, ulEdgeRatio;
                                        bool area_b, aspectRatio_b, ulEdgeRatio_b,
edgeIntersect_b;
                                        //@param aspectRatio : aspect ratio
                                        //@param ulEdgeRatio : upper,lower edge ratio

                                        //set p
                                        for (int i = 0; i < 4; i++) {
                                              p[i] = tempPoly.at(i);
                                        }

                                        aspectRatio = (tempPoly.EdgeDistance(0) +
tempPoly.EdgeDistance(2)) /      (tempPoly.EdgeDistance(1) + tempPoly.EdgeDistance(3) );
                                        ulEdgeRatio = tempPoly.EdgeDistance(3) /
tempPoly.EdgeDistance(1);

                                        area_b = tempPoly.Area() >= areaLimit;
                                        aspectRatio_b = ar_lowerLimit <= aspectRatio &&
aspectRatio <=       ar_uperLimit;
                                        ulEdgeRatio_b = ule_lowerLimit <= ulEdgeRatio &&
ulEdgeRatio <=       ule_uperLimit;
                                        edgeIntersect_b = !IsIntersected( p[0], p[1], p[2], p[3]);

                                        //ar_uper--

                                        if ( area_b && aspectRatio_b && ulEdgeRatio_b &&
edgeIntersect_b) {
                                              vecGrid.push_back(tempPoly);
                                              buildGridTimes++;
                                              break;    //brek for tempCline
                                        }
                                  //END for others base colunm line that intersect to Rline
                                  }
                            }
                      }
                  }
            //END for RLine that intersect to base colunm line
            }
      //END for each baseCLine
      }
//END for each RLine is baseLine
}

      outputVecGrid = vecGrid;
}
```

```
//區域二值化
void Binary(cv::Mat& src, cv::Mat& dst, const int blockSize, const double constant)
{
	using namespace cv;
	using namespace std;

	dst.create(src.rows, src.cols, CV_8UC1);

	const int size = blockSize * blockSize;
	const int width = (blockSize - 1) / 2;
	Mat iImg, isqImg;

	integral( src, iImg, isqImg);

	for (int i = 0; i < src.cols; i++) {
		for (int j = 0; j < src.rows; j++) {
			double sum, sqsum, mean, sd;
			int threshold;

			CPoint center;
			CPoint p[4];
			double iVal[4] = {0};
			double isqVal[4] = {0};

			center.Set( i, j);

			p[0].Set( center.Get_x() -    width, center.Get_y() - width);
			p[1].Set( center.Get_x() - width, center.Get_y() + width);
			p[2].Set( center.Get_x() + width, center.Get_y() + width);
			p[3].Set( center.Get_x() + width, center.Get_y() - width);

			if ( !p[0].IsInlaw(src) || !p[1].IsInlaw(src) || !p[2].IsInlaw(src)
|| !p[3].IsInlaw(src) ) {
				CPoint v;
				int xAddT = 0, yAddT = 0;

				for (int i = 0; i < 4; i++) {

					if ( p[i].Get_x() < 0) {
						xAddT++;
						v.Add_x( 0 - p[i].Get_x() );
					}
					else if ( p[i].Get_x() > src.cols) {
						xAddT++;
						v.Add_x( src.cols - p[i].Get_x() );
					}

					if ( p[i].Get_y() < 0) {
						yAddT++;
```

```
                        v.Add_y( 0 - p[i].Get_y() );
                    }
                    else if ( p[i].Get_y() > src.rows) {
                        yAddT++;
                        v.Add_y( src.rows - p[i].Get_y() );
                    }
                }

                if ( xAddT > 0)
                    v.Set_x( v.Get_x() / xAddT);
                if ( yAddT > 0)
                    v.Set_y( v.Get_y() / yAddT);

                for (int i = 0; i < 4; i++) {
                    p[i] += v;
                }

            }


            for (int k = 0; k < 4; k++) {
                int x,y;

                x = p[k].Get_x();
                y = p[k].Get_y();

                iVal[k] = iImg.at<int>( y, x );
                isqVal[k] = isqImg.at<double>( y, x);

            }

            sum = iVal[2] + iVal[0] - iVal[1] - iVal[3];
            sqsum = isqVal[2] + isqVal[0] - isqVal[1] - isqVal[3];
            mean = sum / size;
            sd = sqrt( (sqsum/size) - mean * mean );

            threshold = mean + constant * sd;



            if ( center.ReadColor(src) >= threshold ) {
                center.WriteColor( dst, 255);
            }
            else {
                center.WriteColor( dst, 0);
            }
        }
    }

}
```

```
//判斷車格是否有車
bool is_car_exist(Car car, Mat& img0, Mat& img1    )
{
	Coordinate x_start_point;
	Coordinate x_end_point;
	double block_size;
	double total_area = area(car.point[0], car.point[1], car.point[2], car.point[3] );
	int exist_area=0;// block

	block_size = sqrt( total_area/730 );

	x_start_point = (car.point[0].x < car.point[3].x )? car.point[0] : car.point[3] ;
	x_end_point = (car.point[1].x > car.point[2].x )? car.point[1] : car.point[2] ;

	///for a car grid
	for(Coordinate p=x_start_point; p.x<= x_end_point.x; p.x+= block_size)
	{
		Coordinate y_start_point,y_end_point;

		///set y_start_point and y_end_point
		{
			ParaT y_line;
			vector<Coordinate> vec_point;
			y_line.stand = p;
			y_line.theta = 90;
			y_line.distant = 0;

			for(int n=0; n<4; n++)
			{
				Coordinate point;
				point = contact( y_line , car.line_parat(n) );
				vec_point.push_back(point);
			}

			int min=0;

			for(int n=0; n<vec_point.size(); n++)
			{
				double dm,dn;

				dm = distant(p,vec_point.at(min) );
				dn = distant(p,vec_point.at(n) );

				if( dn<= dm)
				{
					min = n;
				}
			}
```

```
        y_start_point = vec_point.at(min);

        vec_point.erase( vec_point.begin() + min);

        min = 0;
        for(int n=0; n<vec_point.size(); n++)
        {
                double dm,dn;

                dm = distant(p,vec_point.at(min) );
                dn = distant(p,vec_point.at(n) );

                if( dn<= dm)
                {
                        min = n;
                }
        }
        y_end_point = vec_point.at(min);
}

ParaT line;
line.stand = y_start_point;
line.set_end_point( y_end_point );

///y
for(double distant=0; distant<line.distant; distant+= block_size )
{
        Coordinate block_center;
        block_center = line.point(distant);

        int block_error=0;

        //for a block
        for(int x = block_center.x- block_size/2 ; x < block_center.x + block_size/2; x++)
        {
                for(int y=block_center.y-block_size/2; y<block_center.y+block_size/2; y++)
                {
                        Coordinate block_point;
                        block_point.set(x,y);

                        for(int n=0; n<3; n++)
                        {
                                block_error += abs( block_point.read_color(img1,n)
                                                - block_point.read_color(img0,n) );
                        }
                }
        }

        if( block_error >= error_limit*block_size*block_size )
        {
                exist_area++;
```

```
                        //for a block
                        for(int x = block_center.x - block_size/2 ; x < block_center.x+block_size/2;
x++)
                        {
                                for(int y = block_center.y-block_size/2; y <
block_center.y+block_size/2; y++)
                                {
                                        Coordinate block_point;
                                        block_point.set(x,y);
                                }
                        }
                }
        }
    }

    return ( (exist_area*block_size*block_size) /total_area ) >= area_limit;
}
```