

2014 年臺灣國際科學展覽會 優勝作品專輯

作品編號 110017

參展科別 電腦科學

作品名稱 以長距離立體影像模型為基礎之汽車防撞
警示雷達系統的設計與研究

得獎獎項 大會獎：四等獎
英特爾電腦特別獎

就讀學校 臺北市立敦化國民中學

指導教師 劉睿荷

作者姓名 陳韋同

關鍵字 長距離立體影像(Long Distance 3D stereo
image)、相機光軸校正(Optical Axis
Calibration)、樣板匹配(Template Matching)

作者簡介



大家好，我是就讀於台北市敦化國中三年級的陳韋同，國一時加入數理資優班，老師開了發明課及認識研究等課程，經過這些課程後，我也研發出關於節約能源的新型發明，並已獲得了專利。這種成就感也讓我開始深入地接觸科學，也更有興趣去嘗試做研究並參加科展。

我從小就滿喜歡參加各種營隊及科學相關的展覽，如在美國參加太空及電腦工程相關的營隊，在美國參加營隊的時候也交了許多外國的朋友，我們不僅互相了解對方國家的文化，甚至討論太空及電腦工程相關的議題。就讀雙語小學時，已有實驗課可以作實驗，也因為這些經驗，讓我從小就對科學研究有著濃厚的興趣，並在國一開始了科學研究並參加科展。

在科展過程中經過了不斷的模擬、實作、焊接，甚至星期六日兩整天也都在作實驗，雖結果常常不如預期，但我們都持續努力並找出更好的解決方案。在這研究的過程中我要感謝師長們及家人的支持，他們除教導我們這方面的知識外，還犧牲了他們許多時間來陪伴我們。

未來我希望能將科學研究精神實踐在日常生活中，改善生活中不方便之處，這可讓我的生活充滿科學，充滿樂趣!

摘要

本研究基於雙鏡頭立體成像原理，提出適用於長距離的相機光軸校正演算法則。此演算法則可快速地將失真的影像修復，以提供正確物體影像的視差資訊來得到其 3D 立體影像的資訊。由於物體影像於長距離時會太小而變得難以辨識，因此加大影像解析度能有效提升物體影像的辨識度進而獲得長距離的立體影像，但電腦執行影像處理的運算時間也隨之變長而失去即時性。因此本研究採用圖形處理器及多重執行緒的技術，可將解析度 1280*720 的影像擷取和處理的運算時間縮短至每幅影像 100 毫秒以下，而達到影像處理的即時性。本研究最後將此研究成果用於汽車防撞警示雷達系統設計，此系統運用兩個網路攝影機來擷取影像，並利用 OpenCV 所提供之開放原始碼實現本研究所有使用之影像處理程序，以快速達到汽車防撞警示雷達系統的實現。

Abstract

In this project, an innovative optical axis calibration algorithm is proposed for long distance stereo image modeling. With this algorithm, 3D models can be created on objects that are 100 meters apart or further. As the distance increases, the object in the image becomes smaller.

In order to maintain a suitable image size, the resolution needs to be increased; however, this does decrease execution speed. To compensate for this, a graphics processor unit (GPU) with computing and multi-threading implementation was used to raise the execution speed of the program.

Using these two techniques, 10 frames of 1280x720 high resolution 3D models can be established per second. According to the results, a brand-new automobile anti-collision warning system can be designed, and combined with the optical axis calibration algorithm to achieve long distance 3D stereo image modeling. To design this system, two webcams were used to capture images, and source codes provided by OpenCV were used to actualize the image processing procedures. This can significantly reduce the costs of automobile anti-collision warning systems, as well as provide an efficient system to help avoid car collisions.

一、 前言

(一) 研究動機

近年來電腦科技及 CPU 等級的大幅提升，以及專屬圖形處理器(GPU)的普及應用，加速了電腦視覺的革命。例如機器人的視覺系統或家庭娛樂裝置如 Wii、Kinect，這些都可將所擷取到的周圍事物之影像建立成 3D 立體影像模型，並由此得知處在環境之深度或距離的資訊。但是立體影像模型的建立時，必須經過複雜的電腦運算程序將所得之影像資料進行立體影像校正程序及立體匹配，或是利用額外硬體設備如近紅外線(Near Infra-Red, NIR)光源對物體進行測距。傳統雙影像處理法及近紅外線影像測距法，均無法有效的針對長距離物體來進行其立體影像模型建立，使得目前 3D 影像建立的技術多侷限於短距離之應用。因此本研究將對長距離立體影像的建立進行可行性的研究與實現，並思考是否能夠捨去這種繁瑣的校正方法，而利用一特殊的立體影像演算法則，簡單且快速的將立體影像模型的建立延伸到 100 公尺以上的範圍，並嘗試從此模型中得知物體的距離資訊。最後並將研究成果應用於汽車防撞警示雷達系統的設計，進而證實此方法的可行性及即時性。可用此長距離的立體影像資料，來計算與前車的距離及相對速度，使其符合汽車防撞警示雷達系統的需求，並透過此系統的實現來證明長距離立體影像的正確性及運算的即時性，本研究只要使用相當普及的網路攝影機，即可實現低價的汽車防撞警示雷達系統，對於行車安全將會有更進一步的保障。

(二) 研究目的

本研究主要是利用影像處理法來研究長距離 3D 立體影像模型建立的可行性及即時性，一般 3D 立體影像模型建立的技術都只侷限於短距離偵測，並無法得知遠處物體的深度資訊，這是因為 3D 立體影像模型建立時，需要經過複雜的立體影像校正及立體匹配，隨著距離的增加，這些步驟在執行上將非常困難。本研究的主要目的是提出一全新相機校正的演算法則並結合影像樣板匹配，以期將目前 3D 立體影像模型建立的適用距離能夠延長至 100 公尺以上，下列五個項目將是主要研究的內容：

1. 雙鏡頭取像
2. 提出相機光軸校正演算法則(optical axis calibration)
3. 基於物體偵測之長距離立體影像模型建立
4. 3D 立體影像模型建立之即時處理
5. 汽車防撞警示雷達系統的實現

以下將對於這五項研究內容進行進一步說明：

1. 雙鏡頭取像

雙鏡頭如人類之雙眼，可根據所取得之兩個影像所造成的視差來對與前方物體的距離進行偵測，如下圖 1 至圖 3 為原始圖像及其視差表示圖，距離鏡頭愈近之物體的視差愈大，於是在視差圖上所代表之顏色偏白(灰階值較大)，反之，其顏色偏黑(灰階值較小)〔1〕。而本研究因目的在於建立長距離立體影像模型，於是欲將兩相機之間的距離加大，進而增加視差值，才能達到更長距離的量測。



圖 1 原始影像左圖



圖 2 原始影像右圖



圖 3 影像深度資訊圖

2. 提出相機光軸校正演算法則(optical axis calibration)

由於鏡頭上的鏡片曲率，以及感光元件於組裝時並沒有完全平行於鏡頭，於是會造成影像失真，所以在傳統立體影像模型建立時，立體影像校正的目的是要將取得之失真影像修復，並透過矯正映射的步驟，使得兩影像上同一點置於同一條水平線上，進而做下一步處理。但是現有的影像校正方法多侷限於 10 公尺以內的距離，而且此種立體影像模型建立法的運算量大，若要進行長距離立體影像模型建立，將會產生執行速度上的問題。

本研究使用雙鏡頭取像，進而由視差得出與主體間的距離，但是這兩個攝影機並不一定完全平行，而且攝影機中的感光元件也有可能不與鏡頭平行，這些變因將會造成攝影機的光軸偏轉，偏轉後再進行測距便會有一段落差，若要應用於長距離立體影像模型建立，準確度就會產生一些問題。於是本研究提出一創新的演算法則，欲將此光軸偏轉的現象校正，進而準確的計算出距離，此種校正方式可使執行速度加快，對於長距離的距離測量上將更為準確。

3. 基於物體偵測之長距離立體影像模型建立

使用雙鏡頭進行立體影像模型建立時，背景的影響往往會影響物體影像的視差判斷，特別是在長距離 100m 時，影像解析度 640*480 的物體影像在圖

片中的大小約只有 3x3 個 pixel 左右，計算視差時往往會被視為背景的一部分。為了辨認出長距離 100m 以上的物體，本研究提出一全新的方法，對高解析度 1280x720 以上的影像進行處理，並只處理及將感興趣的物體影像位置抽離出，將其設為一個影像樣板，用此樣板去對另一邊的影像資料做配對，找出最相似的區域後量測兩區域的偏移量，這即為視差值。由於視差值通常不會太大，因此並不需要去匹配整張影像的資料，僅需對感興趣物體周邊的區域進行匹配，這將加速整個立體影像建立的處理速度，而不會受到高解析影像的影響。由於不受到背景的干擾，所以能處理更遠的物體距離，為了達到此目的，需進行下列兩個動作：

(1) 對感興趣的物體影像偵測

本研究是針對汽車防撞警示雷達的應用進行探討，感興趣的物體主要為汽車、摩托車、行人、路樹等物體，汽車防撞警示雷達對於處理速度的要求很高，且須具備即時性，於是在物體偵測方面使用哈爾特徵法 (Haar-like features)，此技術提供快速的物體影像偵測，本研究將針對此部分進行分析，以確認哈爾特徵法可被應用於汽車防撞警示雷達的實現。〔3〕

(2) 樣板匹配(Template Match)法



圖 4 樣板匹配原理圖〔2〕

如圖 4，當感興趣的物體影像被偵測後，可利用樣板匹配法將感興趣

的物體(左上方的方格中)影像當成樣板，來對另一張影像進行搜尋，找出最配對的位置。由雙鏡頭所擷取到的影像大小相當類似，這將非常適用樣板匹配法來找取相同的物體影像，用此法可避免針對背景或整張影像進行處理。由於計算區域的範圍減少，樣板匹配法可大幅加快處理速度以達到即時性的要求，加上不受背景影像的干擾，於是可以對更遠物體的距離進行偵測，能符合本研究的需求。

4. 3D 立體影像模型建立之即時處理

本研究利用基於物體偵測之長距離立體影像模型建立法，對於長距離的物體進行偵測，並且利用樣板匹配的方式找出視差，進而計算出距離。但是進行長距離物體偵測時影像解析度必須要非常大，才能清楚辨認此物體的種類，然而影像解析度的提升會造成影像資料計算的更大負擔，這是應用 3D 立體影像解決問題的主要瓶頸之一。本研究的目的之一是將針對 3D 影像的即時運算，提出有效的解決方案，此方案包括硬體和軟體的解決方案。

關於硬體的解決方案，本研究是利用圖形處理器(GPU)取代原本的中央處理器(CPU)來處理大量的 3D 立體影像資料，圖形處理器已被廣泛地應用於影像處理，其具備大量的微型圖形處理運算單元，如 NVIDIA 的 GeForce 310M 則具有 16 個處理核心，這可對影像資料進行平行多工的運算，這將大幅降低處理的時間。本研究在建立 3D 立體影像模型時會將這部分送交給圖形處理器作 3D 影像處理，相信能提升影像處理的速度，使本研究之汽車防撞警示雷達系統能夠達到即時性。

另外，研究中也將思考採用軟體的解決方案，來提升影像資料運算的即時性。由於當雙鏡頭照相機在擷取立體影像資料時，圖形處理器是處於不工作的狀態，本研究考慮使用多重執行緒(Multi-Threading)的方案，可輕易的將影像資料擷取的時間，和影像資料處理的時間重疊，即當圖形處理器正在處

理目前的影像資料時，另一個執行緒將會同時啟動去擷取下一張影像，這亦可大幅提升整體影像資料的處理時間〔22〕〔26〕。

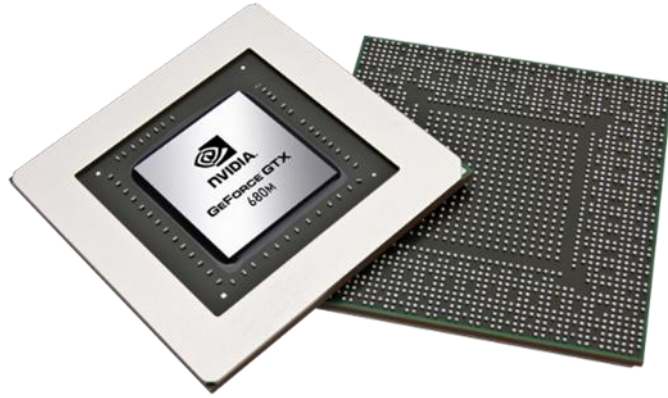


圖 5 圖形處理器(GPU)示意圖〔3〕

5. 汽車防撞警示雷達系統的實現

本研究在使用基於物體偵測的長距離立體影像模型建立法並且使其達到即時性之後，便將其實現於汽車防撞警示雷達系統中，在此系統中便會利用對於長距離量測的準確性，以及影像處理速度所達成的即時性，算出每秒與前車距離的變化，並估算出在多少時間後會與前方車撞上，此時間即為駕駛可反應的時間。而當接近前方車速較慢的汽車時，系統會提醒要減速，並與前車維持一個固定的秒差(即安全距離)，如下圖 6 及圖 7〔4〕。

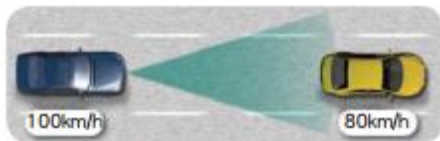


圖 6 偵測與前車相對速度

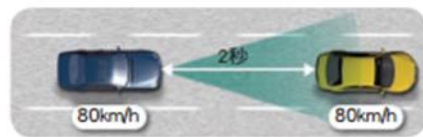


圖 7 執行防撞警示後駕駛減速

二、 研究方法或過程

本研究將依照(圖 8)研究流程圖的步驟來執行。首先文獻探討針對現在關於用影像處理法和近紅外線法進行 3D 立體影像的建立之相關學術研究來作最深入的了解，並說明為何目前研究的結果無法用於長距離立體影像的建立。接下來將針對長距離立體影像的建立提出有效的解決方案，並設計實驗對提出新的演算法則進行驗證，並將實驗的結果作分析比較，以證明此研究提出之演算法則的可行性，最後將此結果實際應用在汽車防撞警示雷達系統上，來說明本研究提出的方案確實可以被實際的實現與應用。

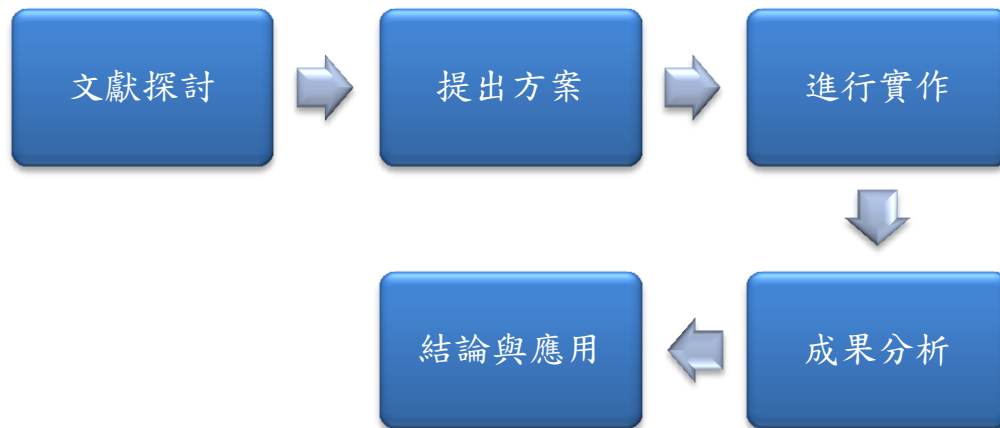


圖 8 研究流程圖

研究過程中，若發現可改善的空間，此流程會重複上一步驟，並重新評估其可行性及提出其他方案，目的是要達到更高的品質，使此研究更完整。

(一) 文獻探討

本研究在文獻探討的部分主要針對傳統雙鏡頭影像處理法和近紅外線影像處理法對 3D 立體影像建立的基本原理作探討，分析如何透過雙鏡頭攝影機形成 3D 影像及如何由兩張影像之視差資訊求出物體的距離；在另一方面要探討如何應用近紅外線光源對環境中的物體進行距離的量測，以及如何偵測感興趣的物體的位置；最後分析文獻並評估是否此技術能適用於汽車防撞警示雷達。

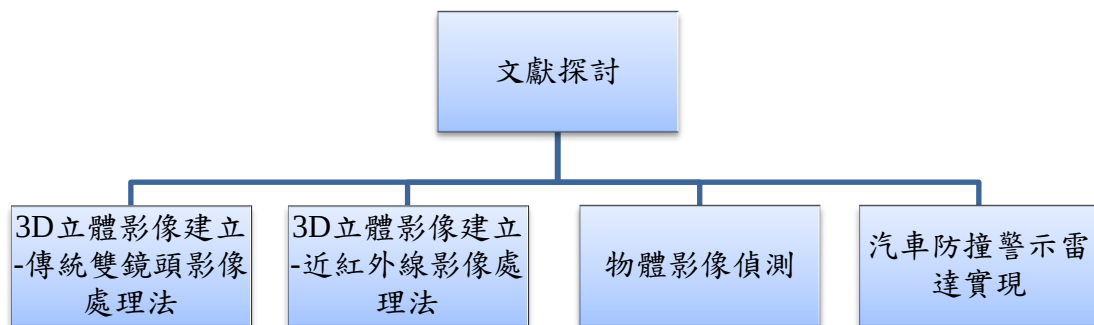


圖 9 文獻探討組織圖

1. 3D 立體影像建立-傳統雙鏡頭影像處理法

(1) 3D 立體影像原理

人對於周遭產生立體感的主要是因為有雙眼的視差，物體在兩眼視網膜上相對成像的位置有些許差異，大腦會藉此將左右眼得到的影像合成一個立體畫面，因此使用傳統雙鏡頭影像處理法建立 3D 影像，必須在相同背景下產生左右兩個影像，而在此背景下的物體因為相機的位置不同，在左右影像中的相對位置亦有不同，這就是物體的視差，這和人的雙眼所產生的視差是一樣的，如下圖 10 將左右相機所拍到的影像利用紅綠來重疊，若戴上由紅濾紙及綠濾紙製成的 3D 眼鏡，左右的紅綠兩張影像便會被分離，而進入的人的雙眼，大腦便會將此具有視差的影像合成一個立體畫面，視差大的物體距離比較近，視差小的物體為遠方的物體，因此視差會為產生遠近距離感，這是使用影像處理法建立 3D 立體影像的

重要關鍵。視差的形成是因為雙眼對主體所見的位置會落在遠方背景的 A、B 點上，雙眼所見背景是一樣，但是主體卻落在背景上不同位置，這位置的差別即為視差如圖 11。



圖 10 3D 影像〔5〕

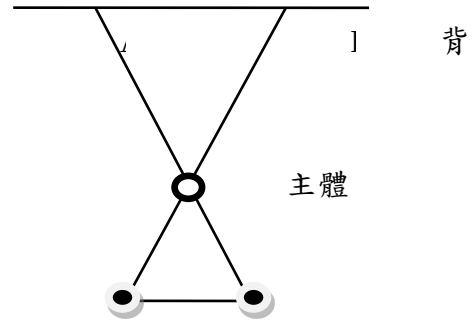


圖 11 視差形成示意圖

使用雙鏡頭影像法之兩張影像形成的視差求深度資訊的方法稱為三角測量法，如下圖 12，先將在現實世界中的主體(O)定義出來，而 M 為相機鏡頭的平面，N 為相機感光元件的平面，f 為相機的焦距，B 為兩相機的差距， x_L 及 x_R 線段為物體 O 在影像上的對應點與影像中心線的距離。將鏡頭中心設為原點，視差(d)的計算便是 x_L 及 x_R 的差，便可利用視差來算出與主體的距離，因此推導出一公式(1)。

D = 攝影機與主體的距離(m)，B = 雙鏡頭的距離(m)，f = 焦距(pixel)，
d = 視差(pixel)〔6〕：

$$D = \frac{B \times f}{d}$$

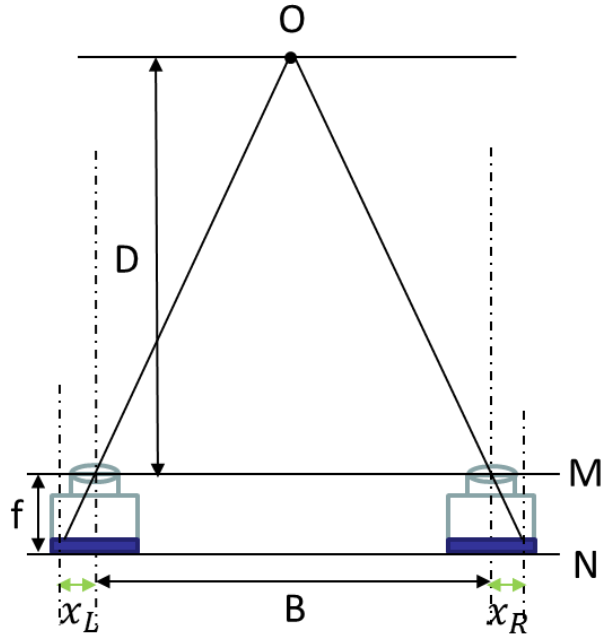


圖 12 利用視差求距離示意圖

根據上列公式(1)可得知只要能得到物體的視差，即可使用此視差的資訊建立其立體影像，目前傳統雙鏡頭影像處理法建立 3D 立體影像，即利用下列 3 個步驟來得到視差的資訊：

<1> 相機校正

相機的兩種主要形變分別是徑向形變以及切向形變。徑向形變是由於鏡片的形狀，使得距離鏡片中心越遠的光線在通過鏡片時彎折較大，越靠近鏡片中心的光線則彎折越小。切向形變則是由相機模組組裝的誤差造成的。主要是鏡片與感光元件並非平行對齊，相機校正步驟的目的就是要將此失真校正回來。圖 13 及 14 為兩種形變的影像〔7〕。

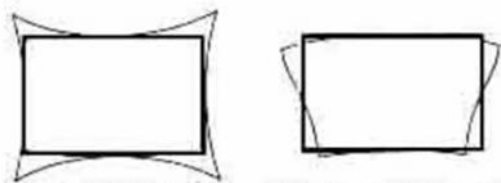


圖 13 徑向形變(左) 與切向形變(右)

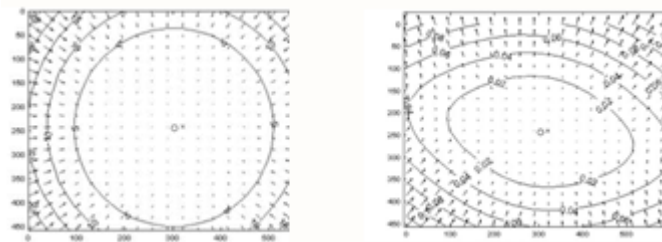


圖 14 徑向形變(左) 與切向形變(右)

利用公式(2)–(5)進行影像失真的修復：

$$x_k = x_d(1 + k_1r^2 + k_2r^4 + k_3r^6) \quad (2)$$

$$y_k = y_d(1 + k_1r^2 + k_2r^4 + k_3r^6) \quad (3)$$

$$x_k = x_d + [2p_1y_d + p_2(r^2 + 2x_d^2)] \quad (4)$$

$$y_k = y_d + [p_1(r^2 + 2y_d^2) + 2p_2x_d] \quad (5)$$

(x_k, y_k) 為完全無失真狀態之影像點

(x_d, y_d) 為失真影像點

$r = \sqrt{x_d^2 + y_d^2}$ 與原點之距

k_1 、 k_2 、 k_3 徑向形變參數

p_1 、 p_2 切向形變參數

利用目前最普遍的棋盤方格校正法將相機的形變參數找出，之後鏡頭所拍到的影像便能立刻用此組數據將失真影像修復，以避免兩個相機不一致造成視差量測的失真，如下圖：

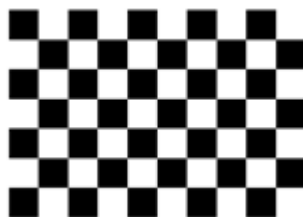


圖 15 棋盤方格圖

<2> 矯正映射(image rectification)

另外，雙鏡頭對於真實物體在影像中也會有不同的位置，如下圖 16 及 17，在此定義影像中X物之點為 X_L 與 X_R ，再將兩鏡頭的中心點連線，此線通過影像的點定義為 e_L 與 e_R ， X_L 與 X_R 以及 e_L 與 e_R 分別連成的線稱為極線。

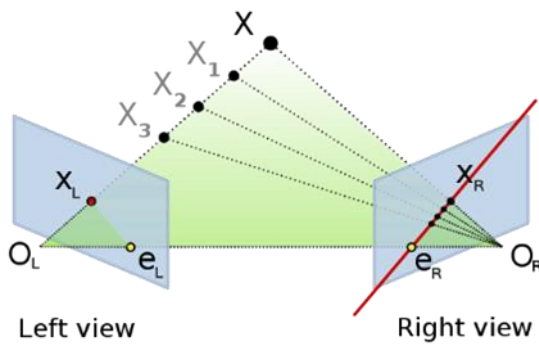


圖 16 矯正映射示意圖〔8〕

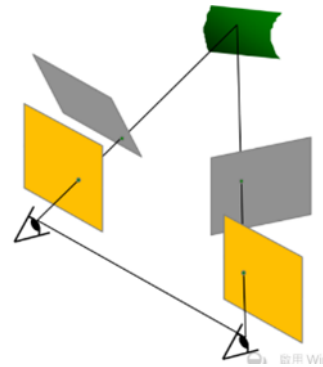


圖 17 矯正映射示意圖〔9〕

在影像校正的程序中，用程式進行矯正映射的目的是要使兩極線共線並與兩鏡頭中心之連線平行，將拍攝後的影像矯正並投影到正確的位置上，在計算視差時才可不考慮垂直方向，加速並簡化之後的計算過程，只對於兩影像同一點的偏移大小來計算視差，如圖 18 及 19。

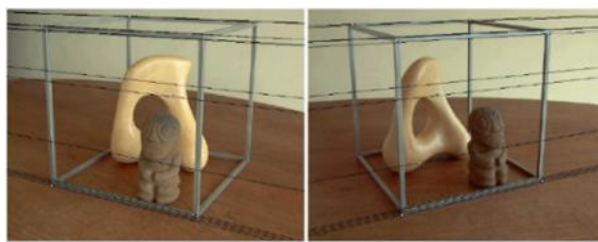


圖 18 矯正映射前之雙影像〔9〕

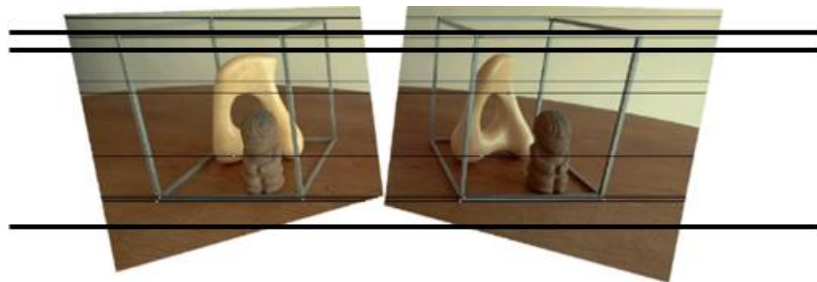


圖 19 矯正映射後之雙影像〔9〕

<3> 立體匹配 BM 算法求深度圖

BM 算法又稱(block matching)為常用視差計算的演算法，當左右兩張影像已經過矯正映射後，即可使用此法計算整張圖的視差，一開始先決定兩張影像比對的區塊大小，之後即可計算此區塊在同一水平的偏移量，這即為此區塊的視差值如下圖 20，由於是對整張圖進行立體匹配，其執行時間相當慢，在使用 CPU: i5-M460 的電腦對於 640x480 的解析度進行操作時，執行時間會大於 800ms。



圖 20 立體匹配 BM 算法求深度圖〔10〕

2. 3D 立體影像建立-近紅外線影像處理法

(1) 飛時測距影像法(Time-of-flight)

此技術根據何時收到反射光來決定距離，以每一個像素為單位，得知每一個像素的光收到的時間，以此時間即可得知此像素的距離。其做法為通過給目標發送近紅外線的光脈衝，然後用近紅外線相機接收從物體返回的光，通過探測光脈衝的飛行時間來得到目標物體的距離，TOF 的相機接收整幅近紅外線圖像，其相機具備時間量測的機制，可測得每一個像素是何時成像的，然後可由公式(6)得出其距離的深度資訊〔11〕。

(D：與主體間的距離； t_d ：從發出紅外線到接收的時間；c：光速)

$$D = \frac{t_d \times c}{2} \quad (6)$$

(2) 雷射散斑影像法(Kinect 技術原理)〔12〕

此技術利用連續光（近紅外線）對測量空間進行編碼，經感應器讀取編碼的光線，交由晶片運算進行解碼後，產生成一張具有深度的圖像。其以紅外線發出人眼看不見的雷射光，再透過鏡頭前的光柵、擴散片，將雷射光均勻分佈投射在測量空間中，最後用紅外線攝影機記錄下空間中的每個散斑，擷取原始資料後，再透過晶片計算成具有 3D 深度的圖像，如下圖 21 所示。

Kinect 可利用普通攝影機獲得立體影像，以及聲音等資訊，而且此裝置還包含一個紅外線的發射器，可以進而得到一深度圖，如下圖 22 所示：

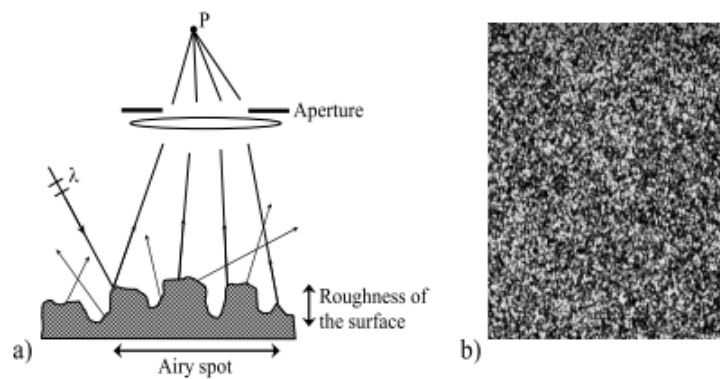


圖 21 散斑形成圖

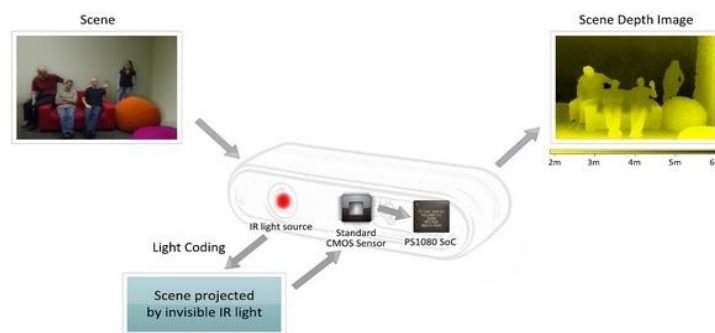


圖 22 雷射散斑影像法(Kinect 技術原理)圖

下表 1 為上述技術所擁有的一些缺點：

表 1 雙鏡頭及近紅外線影像處理技術之缺點

	缺點 1	缺點 2	缺點 3
傳統雙鏡頭 影像處理法	侷限於短距離 (校正程序無法在 長距離執行)	需經過複雜的 校正程序	運算時間長 (>800ms per frame 640x480)
近紅外線影像 處理法－飛時測距	侷限於短距離 (近紅外線光干擾 問題)	無法多台 同時使用	若有強烈的背景 光源則不能使用
近紅外線影像 處理法－雷射散斑法	侷限於短距離 (近紅外線光干擾 問題)	無法多台 同時使用	若有強烈的背景 光源則不能使用

3. 物體影像偵測 [13]

在實際 3D 影像的應用中會找出感興趣物體進行分析，例如，找出手掌的位置及其深度資訊，然後依據手掌的 3D 資訊來做應用，因此本研究將先進行物體偵測然後再對特定物體進行視差資訊的計算，如汽車、行人、及路樹等，將其由背景中分離出來，因此進行影像的分析及運算的時候便不會受到背景的影響，也無需處理整張影像，因此處理速度也將大大提升。研究過程中對於物體影像偵測主要使用哈爾特徵法(Haar-like features)，之後再利用樣板匹配法(Template Matching)將兩圖像的視差值求出，此兩種技術的詳細說明如下：

(1) 哈爾特徵法

此方法需要事先對其資料庫進行訓練，之後將拍攝到的影像與資料庫中的資料做比對，找出最符合的區域，而此比對則是利用區域內灰階值的總和，如下圖 23，點 1 代表 A 區域中的像素灰階值總和；點 2 代表 A+B 區域中的像素灰階值總和；點 3 代表 A+C 區域中的像素灰階值總和；點 4 代表 A+B+C+D 區域中的像素灰階值總和。當在做訓練時若想要計算 D 區域內的像素灰階值總和，需用 “點 4 + 點 1 - 點 3 - 點 2” 來求

出，而在此得到一系列數據之後，將會把此作為此物體的一種特徵，而在之後進行辨識的時候便是利用特徵值來搜尋相似物體。

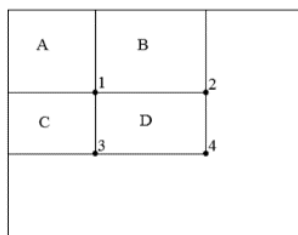


圖 23 影像中區域劃分及點之定義圖

在訓練的時候將 A、B、C、D 定義，在影像中將出現與其最相似的區域框出。拿 C 當例子，若在影像中出現與先前訓練之資料庫中的特徵有類似之處，便會開始計算與 C 的相似度，將相似度最高的幾個結果框出。

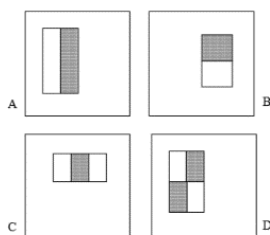


圖 24 哈爾特徵法之灰階值比對

如下圖 25，若出現類似人類眼睛的特徵，如有眼白及黑眼球等特徵，便可偵測此區塊為人的眼睛區域，本研究便是利用此技術偵測汽車、行人、或路樹等物的位置後再作判斷。



圖 25 哈爾特徵法實際使用

(2) 樣板匹配法

樣板匹配是一種在一幅圖像中尋找與另一幅樣板圖像最匹配(相似)部分的技術，而執行此技術必須要有兩個資料，一幅圖像，以及一個事先設定好的樣板，最終的目標是在此圖像中檢測出最匹配的區域。但為達到此目的必須要使此樣板滑動，透過滑動才可對照影像上的每一個區域，使樣板與影像各區塊的灰階值比對，找出最相近的區域即為另一圖像中的此樣板位置，本研究便是利用此技術求出視差，如下圖 26 為樣板匹配的運作圖。

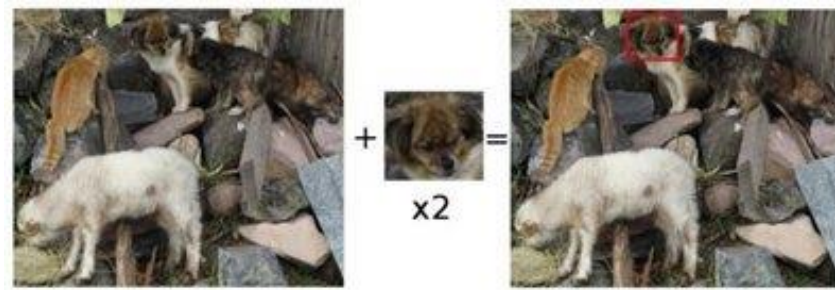


圖 26 樣板匹配執行圖

4. 汽車防撞警示雷達

(1) 雷射測距雷達

此測距的原理是藉由投影點光源至待測物表面的三角關係，以計算出待測物與光源的距離。圖 27 中，待測物在 B1 時，其表面上的散射或反射光在位移檢測器上成像 C1。同理，當待測物在 B2 時，成像為 C2。由 C1-C2 間的距離可以推出位移量 D。光電感測器通常必須要固定，此法應用於測速照相上，偵測距離可達數百公尺，但不適用於行進間車輛使用。

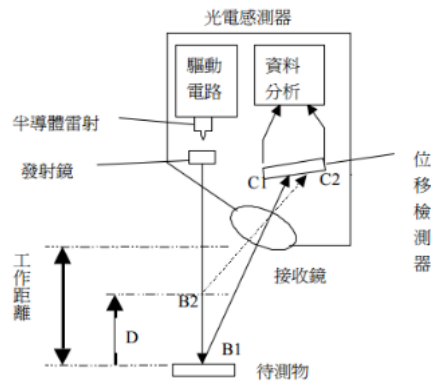


圖 27 雷射雷達運作原理圖 [14]

(2) 毫米波雷達

毫米波汽車防撞警示雷達，是所有雷達中受到的天候影響最少的種類，此技術是使用無線射頻技術設計波長毫米等級的發射與接收電路，來偵測前車的距離及相對速度，目前的主流為 24GHz 和 77GHz 兩種頻段，偵測距離可達一百公尺至兩百公尺長。由於使用頻段相當高頻，其製作的困難度及成本也相對較高，雖然雷達波可克服天候的影響，但卻無法辨識物體的種類，這容易造成誤報。下圖 28 為毫米波雷達示意圖。

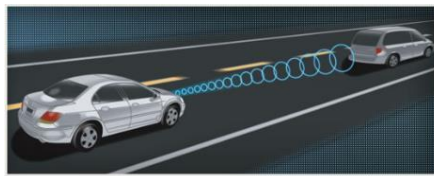


圖 28 毫米波雷達運作示意圖 [15]

(3) 先進駕駛協助系統

此系統除了利用毫米波雷達之外，並同時使用立體攝影機的技術來解決原本毫米波雷達的缺點，無法判別物體的種類及其潛在的危險。此系統之立體攝影機負責近距離物體的危險因素判斷，並辨別及分析潛在危險，同時避免不必要的緊急煞車與轉彎（如閃避小動物）。雷達能辨別前方災害，攝影機則在更靠近時判斷是否要煞車。



圖 29 先進駕駛協助系統實際運作情形〔16〕

(4) 汽車雷達種類比較

表 2 汽車雷達種類比較表

	偵測距離	成本	安裝困難度	缺點
雷射測距雷達	中偏長	高	難	受天候影響
毫米波雷達	長	高	難	非常容易誤判， 準確度上不能依賴
先進駕駛協助系統	中偏短	高	難	適用距離較短

本研究將利用全新概念的演算法來建立長距離的3D立體影像模型，並使用其中的深度資訊偵測遠距離之物體深度並計算出相對速度，不僅能夠即時警示駕駛人前方路況，也因結合樣板匹配長距離立體影像模型建立法，使得偵測目標更鎖定並且能適用於更長的距離，由於使用一般的網路攝影機可達到不錯的效果，實現低成本之汽車防撞警示雷達系統，相信對行車安全有非常大的助益。

(二) 提出方案

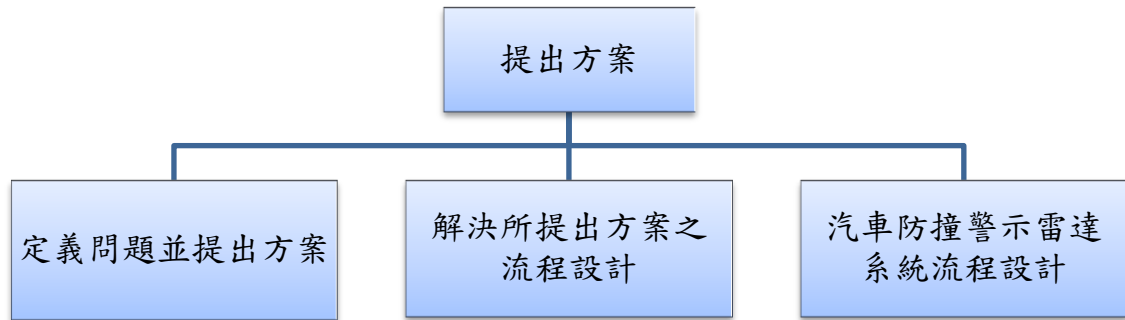


圖 30 提出方案組織圖

在提供解決方案時，會依照目前遭遇到的問題，設法改善並設計出一套影像處理的流程，以期將此應用在長距離的立體影像模型建立上。

1. 定義問題並提出方案

本研究依據文獻所定義出問題及提出方案如下表 3：

表 3 定義問題及提出方案列表

定義問題(What?)	提出方案(How?)	為何使用(Why?)
如何得到具有深度資訊的 3D 立體影像?	使用兩台 Logitech C920 之網路攝影機進行拍攝，利用影像處理的方式把雙鏡頭所擷取到的兩張影像形成之視差進行計算，可得知其深度資訊	文獻中的近紅外線立體影像建立法只能侷限在短距離的偵測，較無法符合本研究的需求
兩攝影機的光軸並不平行	設計一全新的演算法則將偏移的光軸校正，先假定一攝影機的光軸為理想狀態，再對另一相機的光軸進行校正，能減少許多計算過程	兩相機的光軸若沒有進行校正，其視差量測值將會與理想狀態下不同，而在依據公式(1)所算出來的距離將會有很大的誤差
如何使用雙相機進行長距離物體影像模型的執行速度提升	對於感興趣的物體進行辨識，並將感興趣物體所在區域設為樣板，針對另一個攝影機所得到的影像中同一位置的延伸做樣板匹配，即能找出兩張影像中的視差。只針對便是出來的區域內進行視差的計算，如以便能簡化計算過程且不受背景干擾。	此系統(汽車防撞警示雷達系統)只需要針對車道上的物體進行偵測以及辨識即可，對於兩旁以及天空的區域並不需要做辨識，這可提升系統執行速度
如何即時擷取及處理	使用多重執行緒以及徒刑處理	因為物體在長距離時必須

高解析度的影像	器 GPU，從程序以及電腦硬體兩方面加速影像處理速度	使用高解析度的影像，使得進行立體影像模型建立時所算出之視差值較為準確
如何實現此系統?	利用網路上 OpeCV(影像處理函式資料庫)所提供的開放原始碼進行實作	使用開放原始碼可使研究過程只需專注在演算法則的建立

2. 解決所提出方案之流程設計

根據表 3 所列出的問題以及方案，在此章節將會對這些解決方案的流程進行設計。本研究在對汽車防撞警示雷達系統所需之資料進行運算之前，要先對於雙攝影鏡頭作校正，依據先前文獻資料，找出相機光軸的偏轉角度，將影像修復，再透過物體偵測的技術使執行速度提升，在物體偵測前必須要建立其影像的資料庫以及定義物體所在的辨識區域，此能大大減少誤報的機率，而上述這幾點將皆由 OpenCV 提供之原始碼在程式中實現。圖 31 為此流程的說明：

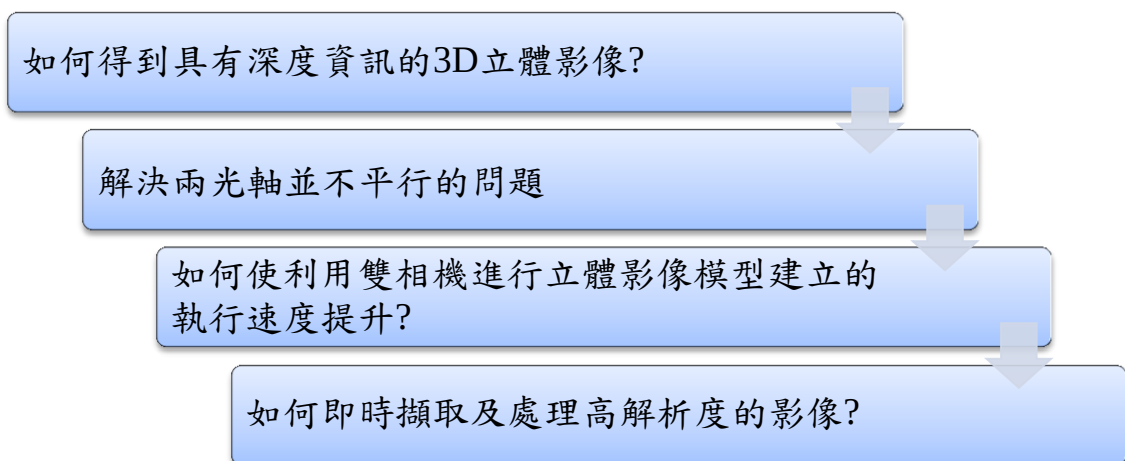


圖 31 影像前置處理流程圖

(1) 如何得到具有深度資訊的 3D 立體影像

本研究根據文獻之立體影像模型建立的方法，發現單鏡頭以及近紅外線等技術在對於本研究對於長距離立體影像模型建立的需求並不一致，於是採用雙相機的方式，從兩張影像所形成的視差進行立體影像模型的

建立，取得其深度資訊。在計算之前將兩鏡頭的間距定義為 90cm，取得 640×480 、 1280×720 、 1920×1080 等三種解析度的影像進行後續實驗比較，以找出最適合本研究研發之系統的影像解析度。

(2) 解決兩光軸並不平行的問題

先前的文獻中有提到，攝影機的光軸偏移會造成距離計算的很大誤差，於是本研究嘗試利用作圖的方式推導出一個公式，欲將攝影機偏轉的角度計算出來，並將此資訊加入汽車防撞警示系統中，進而準確的計算與前車的距離，如下圖為雙攝影機的模擬情形。

M 為攝影機鏡頭平面，N 為感光元件平面；f 為攝影機焦距；O 點為物體的位置； θ_L 及 θ_R 為攝影機光軸偏轉的角度； β_L 及 β_R 為攝影機觀察物體的視角；**黑虛線**為理想狀態下的光軸；**藍虛線**為偏轉後的光軸；**紅線**為此物體反射的光線射入攝影機的路徑； x_L 及 x_R 為理想狀態下物體在影像上的偏移(視差為兩偏移量的差)； x_2 為光軸偏轉後物體在影像上的偏移；

在下列各種情況，本研究皆設定光軸的左方為正，光軸的右邊為負，若此物體在影像上的對應點相對於光軸在左，則此偏移量即為正；反之，即為負值；而角度的正負計算則設定光軸向右偏轉的角度為正，光軸向左偏轉的角度為負。

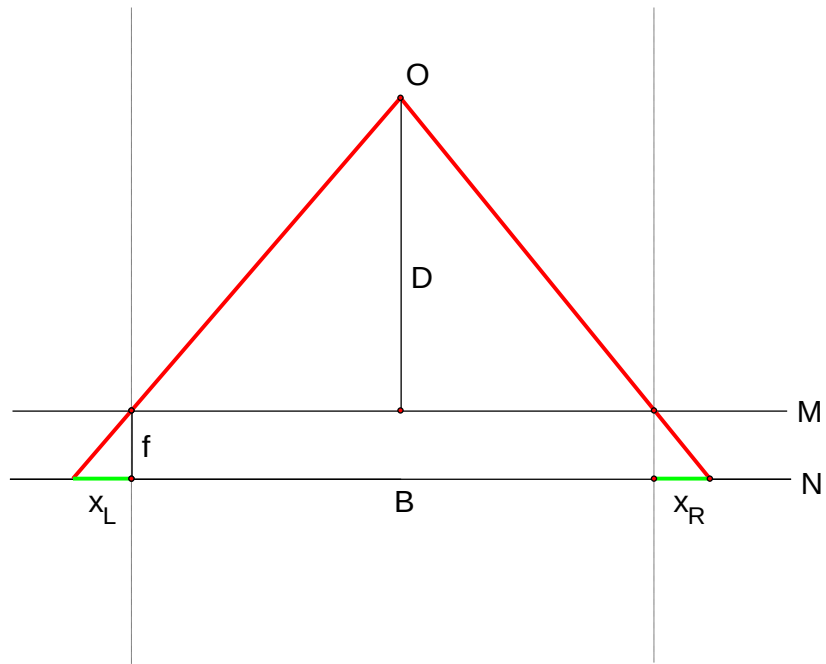


圖 32 雙鏡頭系統理想狀態圖

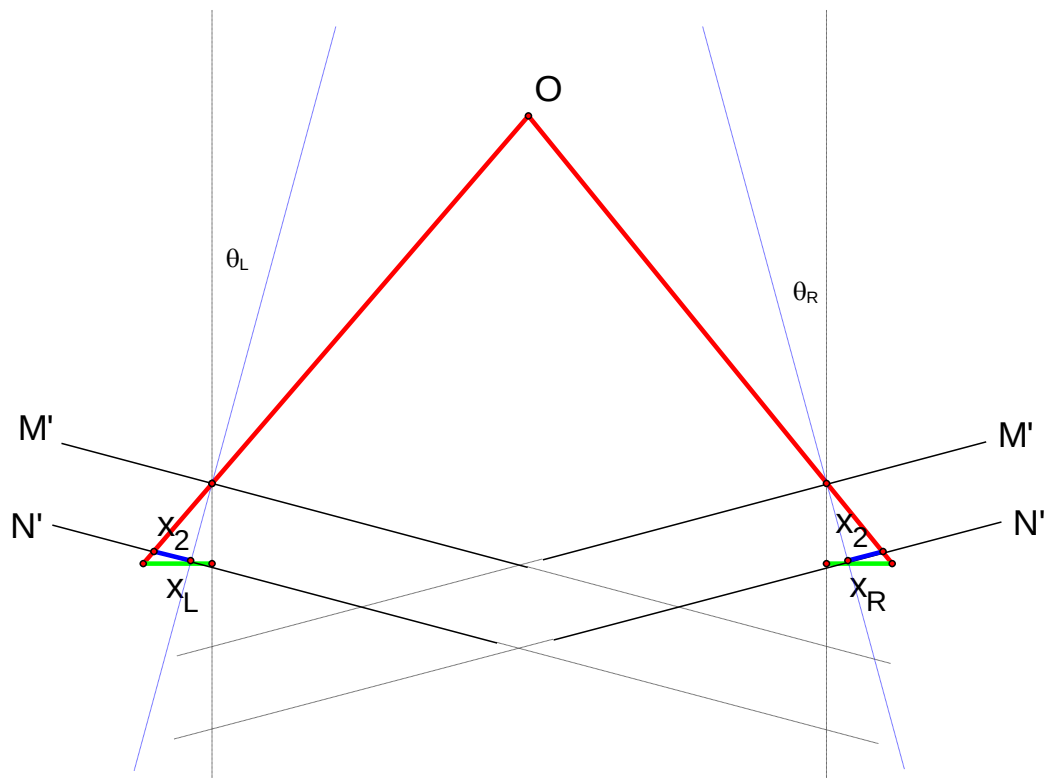


圖 33 雙鏡頭光軸偏轉情形圖

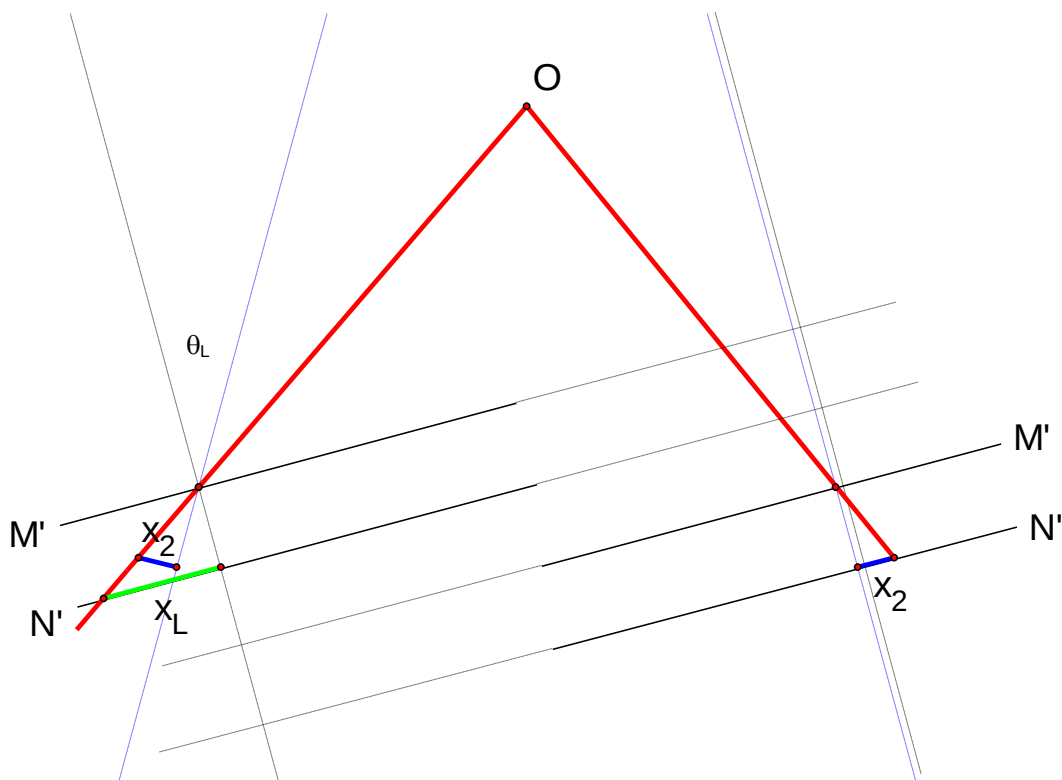


圖 34 本研究欲校正光軸偏轉情形構想圖

圖 32 為最理想的雙攝影機，依此模型所量測出的距離將會是最理想的距離，而圖 33 為攝影機光軸有偏轉角度情況下的模型，於是本研究依據圖 33 的模型推出圖 34 的模型，也就是將其中一攝影機的光軸平行，所以之後探討的問題將只要探討一個相機光軸偏轉的角度如何得知，即可當作整個系統的光軸總偏移量，下圖 35 至圖 42 為本研究歸納出的八種攝影機光軸偏轉的情形，以及利用此來計算出一個全新的演算法則，欲在進行距離計算之前得知攝影機的偏轉角度，進而將量測到不正確的視差修復為正確的視差，相信在距離增加之後能夠大大提升其準確度，對於汽車防撞警示雷達系統的應用上也會相當的實用。

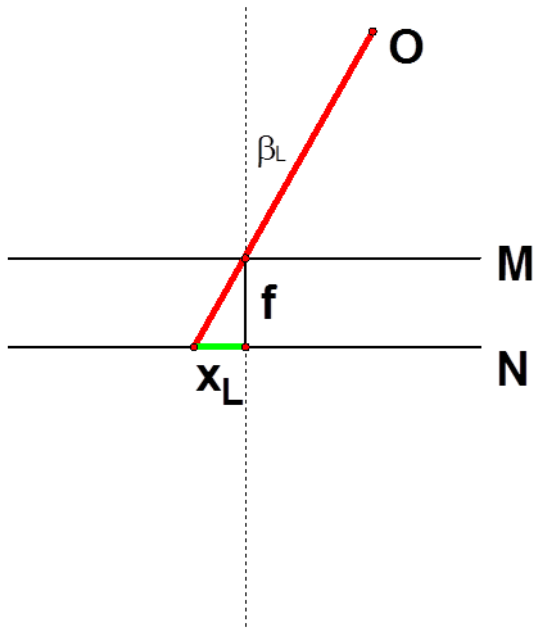


圖 35 左相機理想狀態圖

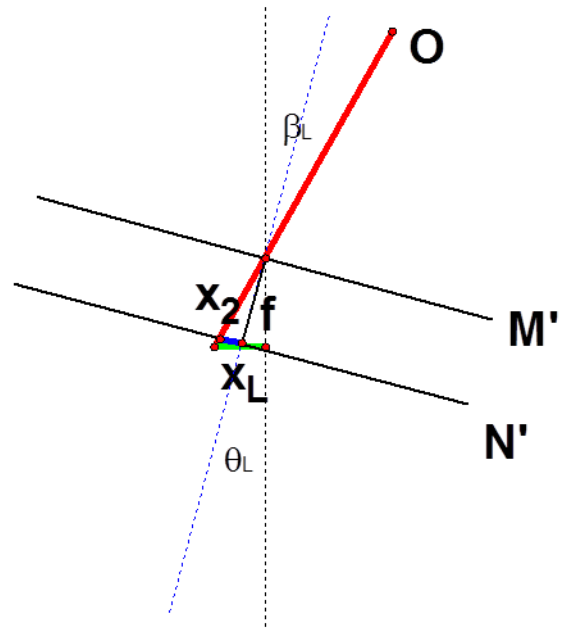


圖 36 左相機 A 情形偏轉圖

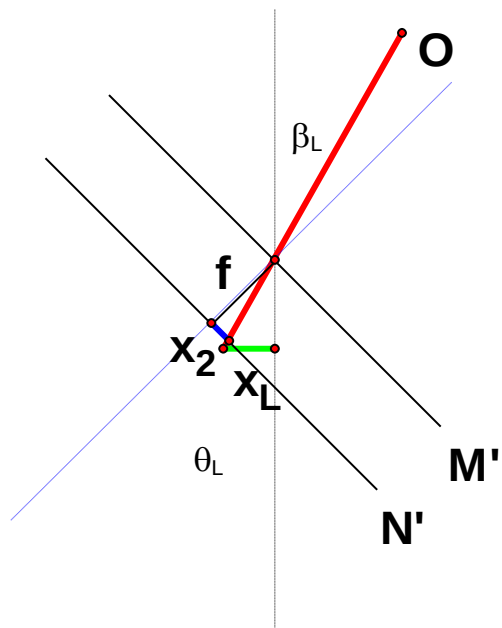


圖 37 左相機 B 情形偏轉圖

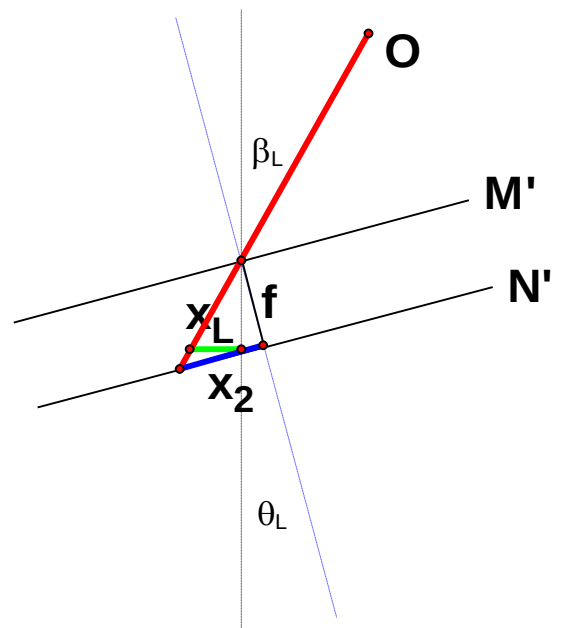


圖 38 左相機 C 情形偏轉圖

上述四種情形為發生在左攝影機的四種偏轉狀況，分別為理想情形、光軸偏轉角度小於視角 β (理想光軸與物體反射光線夾角)、光軸偏轉角度大於視角，以及光軸往物體的反方向偏移，可用下列四個式子計算。

圖 35： $x_2 = x_L = \tan \beta_L$ ； 圖 36： $x_2 = \tan(\beta_L - \theta_L)$

圖 37： $x_2 = \tan(\theta_L - \beta_L)$ ； 圖 38： $x_2 = \tan(\theta_L + \beta_L)$

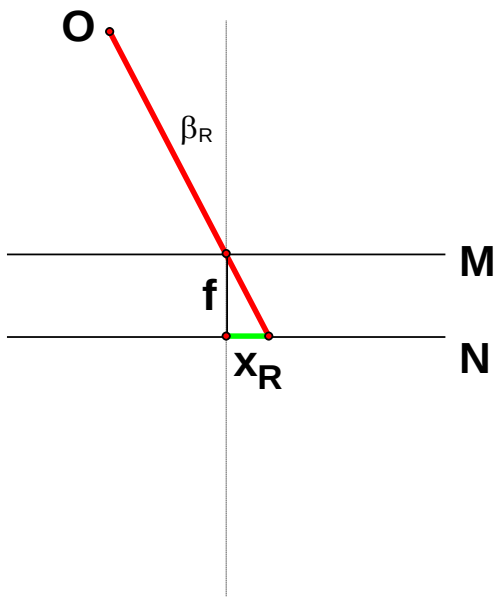


圖 39 右相機理想偏轉圖

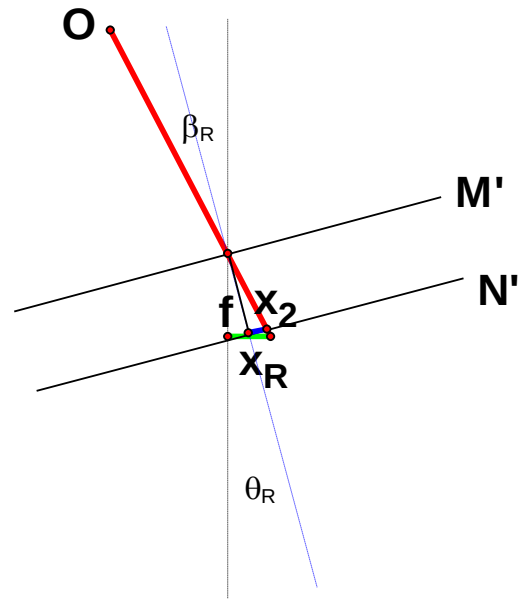


圖 40 右相機 A 情形偏轉圖

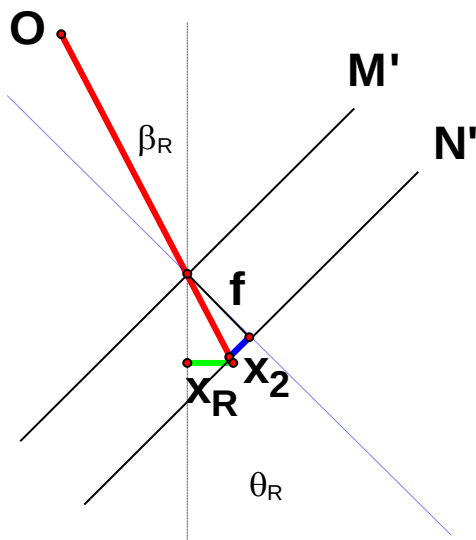


圖 41 右相機 B 情形偏轉圖

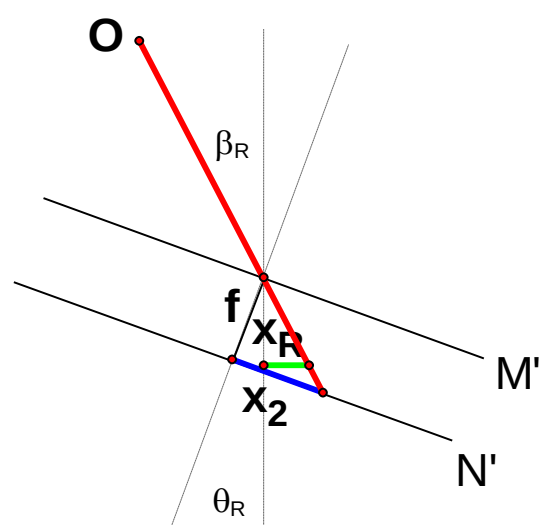


圖 42 右相機 C 情形偏轉圖

上述四種情形為發生在右攝影機的四種偏轉狀況，分別為理想情形、光軸偏轉角度小於視角 β (理想光軸與物體反射光線夾角)、光軸偏轉角度大於視角，以及光軸往物體的反方向偏移，可用下列四個式子計算。

$$\text{圖 39: } x_2 = x_R = \tan \beta_R \quad ; \quad \text{圖 40: } x_2 = \tan(\beta_R - \theta_R)$$

$$\text{圖 41: } x_2 = \tan(\theta_R - \beta_R) \quad ; \quad \text{圖 42: } x_2 = \tan(\theta_R + \beta_R)$$

利用上述八種情形所導出的方程式歸納出一個通式，並將本研究正負號的設定納入考慮範圍內，最後得出 x_2 與 x_1 的關係式為 (x_1 為左右相機在理想狀態下的偏移，極為 x_L 與 x_R)：

$$x_1 = f \left(\frac{x_2 + f \tan \theta}{f - x_2 \tan \theta} \right) \quad (7)$$

再經過移項之後得到的攝影機偏轉角度即為：

$$\theta = \tan^{-1} \left[\frac{f(x_1 - x_2)}{x_2 x_1 + f^2} \right] \quad (8)$$

本研究在接下來的實作過程中將會將此公式整合入系統中，欲能更快速且準確的計算出距離。

(3) 如何使利用雙相機進行立體影像模型建立的執行速度提升?

為解決利用雙相機進行立體影像模型建立的執行速度的問題，本研究使用了物體偵測以及樣板匹配等兩個技術，在感興趣物體的區域內進行運算，可減少許多運算量，使得計算過程更為單純、執行速度更快。以下為物體偵測以及樣板匹配法的流程設計：

<1> 物體偵測

為解決測距時背景所造成的干擾，以及加速影像處理的速度，本研究對於文獻中之立體影像模型建立的方法進行改良，在做視差計算

的時候只對物體本身的視差做運算。而要如何找出影像所在的位置呢？這時就需要仰賴影像辨識技術。本研究使用了哈爾特徵法來辨識物體，並利用先前訓練好之資料庫中各物體獨特的灰階值進行比對，找出最相近的物體，而在一個定義出之影像辨識區域中作影像偵測，以減少誤報的機率。以下為訓練資料庫以及定義影像辨識區域的流程設計：

A. 建立影像辨識資料庫

本研究在影像辨識的部分將使用哈爾特徵法來進行影像的偵測，但既然其名為特徵法就是要用此物體上的特徵去作比對。因為電腦在沒有資料的情況下是無法得知欲辨識物體，如汽車、摩托車、腳踏車、路樹、動物或人類的特徵，於是在使用此種特徵法之前，依據先前之文獻，必須先建立一個資料庫，以便進行影像辨識時可以去做比對。本研究之系統在進行影像辨識時所需使用的資料庫由 OpenCV 提供，其資料庫有針對汽車以及行人的辨識，對於摩托車、腳踏車以及動物等物必須自行建立其資料庫。自行建立資料庫的最大好處在於日後可以對於任何物體進行辨識，也能減少物體突然衝出至道路中所造成的傷害，能提升行車安全。

影像辨識資料庫需針對不同的物體進行哈爾特徵資料庫訓練，經此訓練之後會產生相對應知影像辨識資料 xml 檔案，在接下來的物體偵測時，會利用此訓練的成果檔案進行比對，以找出符合特徵之物體，首先需要準備正向影像(圖 43)以及負向影像(圖 44)。正向影像為具備欲辨識物體之影像資料，如汽車或行人等，其影像資料數量愈多愈好，可準備 2000 到 3000 張正向影像；而負向影像為不含偵測物體之背景影像資料，如道路背景資料，影像資

材料的數量可約為 3000 張。〔17〕



圖 43 哈爾特徵法訓練影像資料庫---正向影像

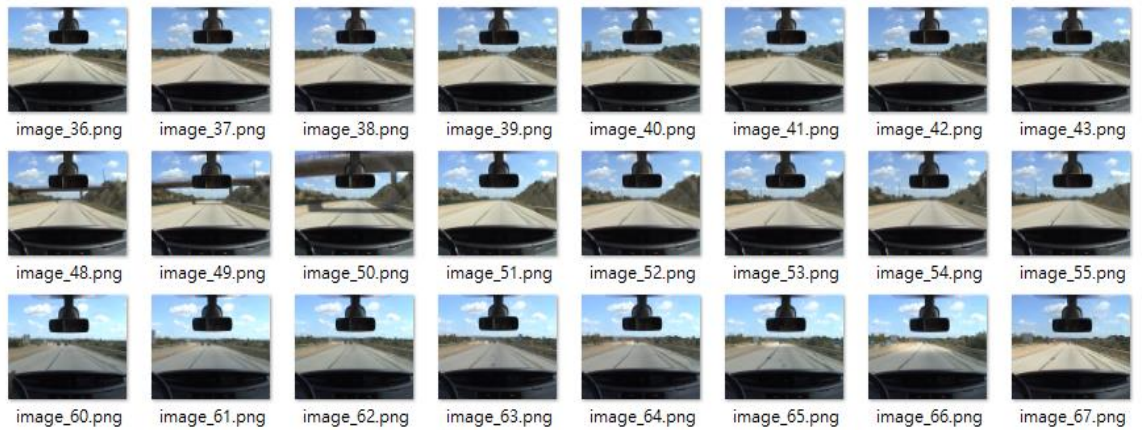


圖 44 哈爾特徵法訓練影像資料庫---負向影像

B. 定義物體的辨識區域

本研究在進行影像辨識時，須先定義出一個物體辨識的區域，以降低誤報率。在執行此系統前須先將車道線，以及能看到前車的區域先大致定義出來，之後在系統中便是利用這個區域內所出現的汽車物體進行偵測，這將會大大的減少誤報的機率：



圖 45 定義影像辨識區域

<2> 樣板匹配法

如圖 46 及 47，在偵測到物體後，要進行視差的測量。。本研究擬利用樣板匹配的方法，此方法為先在左右圖其中一張中找出物體的位置並定義一方塊區域為物體位置(紅色實線)，再至右圖中同一處向上下左右延伸(紅色虛線)，之後把左圖中的區域當作樣板在右圖延伸後的區域內做比對，找出最相近的區域後(藍色實線)，只計算水平座標的偏移，即為視差。找出視差後便可利用先前文獻中的距離計算公式(1)算出，這便是攝影鏡頭到物體的實際距離。



圖 46 樣板匹配原理說明左圖



圖 47 樣板匹配原理說明右圖

(4) 如何即時擷取及處理高解析度的影像?

物體在長距離時大小通常較小，所以在進行長距離立體影像模型建立時必須要使用高解析度的影像，進而從兩張影像中找出更精準的視差

值，但是高解析度的影像會造成處理速度變慢，於是本研究打算採用圖形處理器(GPU)以及多重執行緒(Multi-threading)的方式，利用電腦硬體以及程序等兩方面加速影像處理的執行速度。下圖 48 為單一執行緒以及多重執行緒對於本系統的處理時間比較，可以從下圖中得知將不同區域所處理的部分重疊能加快此系統的運作，攝影機擷取完一張圖後，當電腦將其影像資訊取得並做運算時，攝影機即擷取下一張圖，這可使得整個程序執行速度提升，達到本研究的需求。

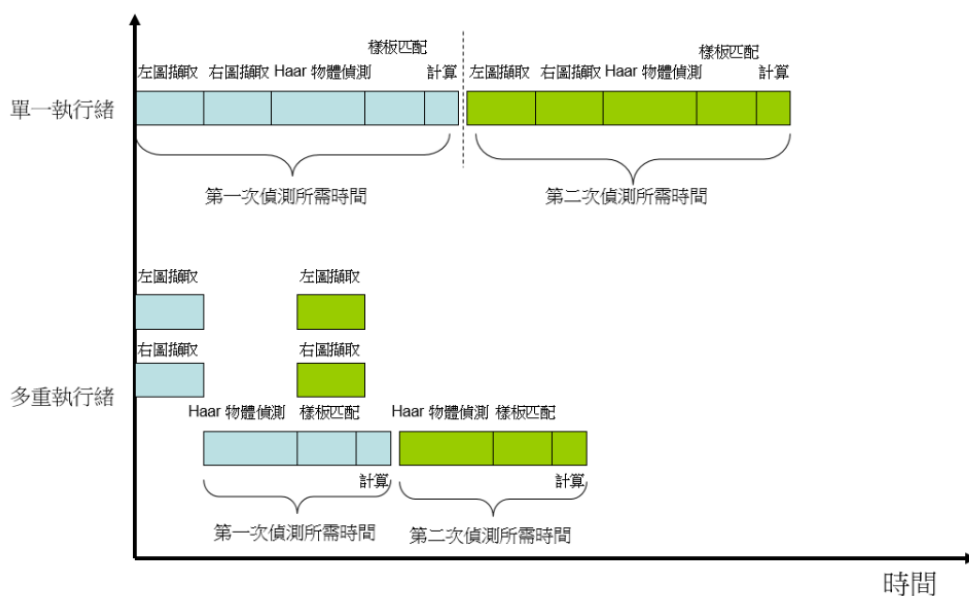


圖 48 單一及多重執行緒運作情形比較

3. 汽車防撞雷達警示系統流程設計

本研究設計之汽車防撞警示雷達系統為下圖 49，嘗試利用多重執行緒的方式將左影像、右影像，以及主執行緒的程序同步執行，使得影像處理的速度能夠更快，執行緒的聯結則使用標誌的方式，當此標誌被開啟時，則另一執行緒也會跟著啟動，本研究便是利用此方法將主執行緒以及左右相機的影像擷取達到同步的動作。主執行緒包含了偵測物體、樣板匹配求視差並計算距離、防撞警示等幾個步驟，先利用哈爾特徵法找出物體的所在位置後，使

用樣板匹配的方式進行視差的計算。找出視差後即可計算出距離，且更進一步得知與前車之相對速度，再來進行判斷是否對駕駛人執行防撞警示，這樣便是一個完整的汽車防撞警示雷達系統：

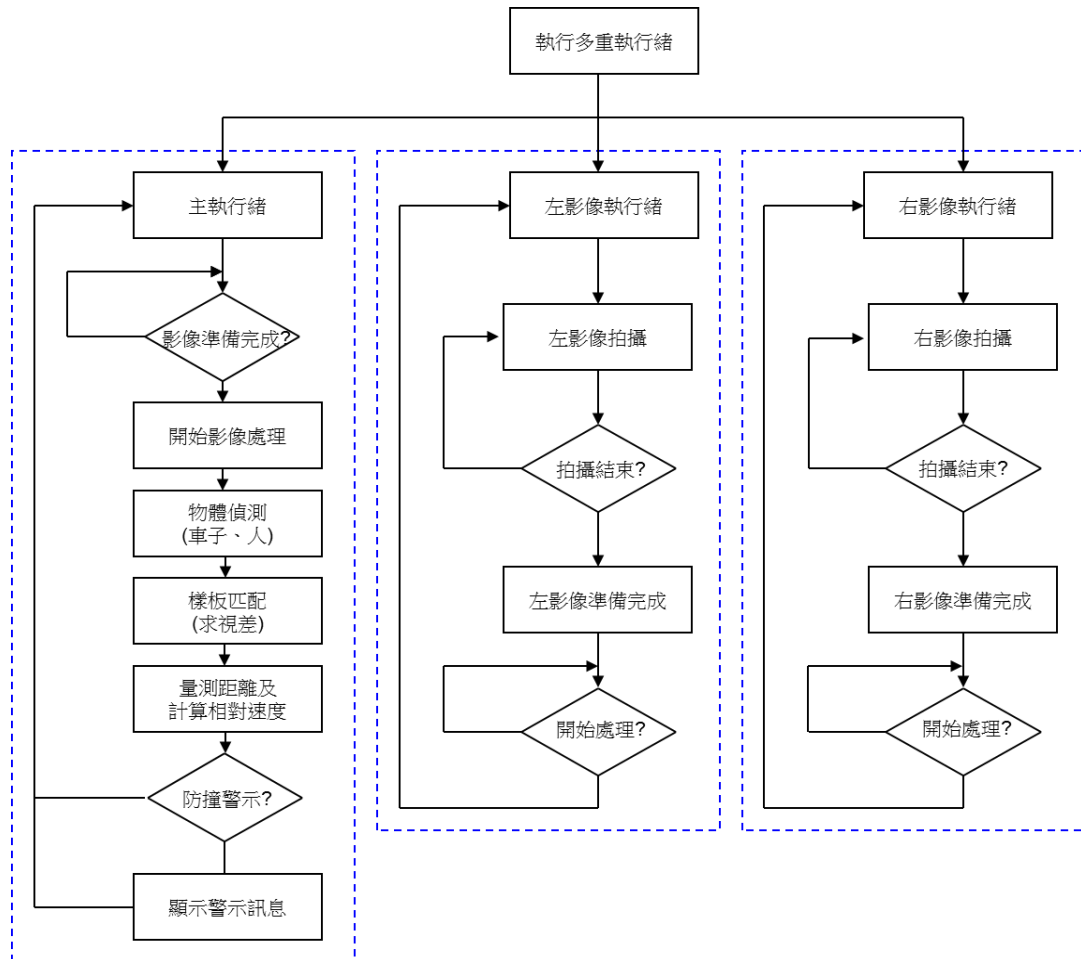


圖 49 汽車防撞警示雷達系統流程圖

汽車防撞警示雷達主要的偵測物體為汽車以及路上的行人，本研究定義三種情況下此防撞警示會啟動來提醒駕駛人。第一種情況是當與前車之距離連續三次皆減少，意即前車減速或本車加速，這時的防撞警示雷達就會立刻執行警示；第二種情況是當與前車的距離為 10 公尺以內時也會對駕駛人進行警告；第三種情況是前方突然有緊急煞車的情況，在 0.1 秒內減少了 2 公尺以上時，也會對駕駛人進行警示，而上述三種情況若同時發生則只會執行一次。對於行人、動物、自行車、以及摩托車等部分，若其突然衝至車道中央，便會直接發出警示，相信這對行車安全也會有更進一步的保障。

在下圖 50 之範例中，兩車的時速分別為 90 及 70km/hr，而其相對速度約為 5.5m/s，所以後方車輛在兩秒後前進 11 公尺，達到本研究與前車距離 10 公尺以內便會警示的條件，於是會通知駕駛人減速。

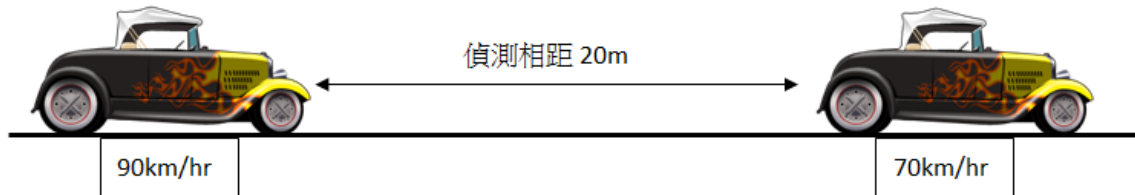


圖 50 汽車防撞警示系統運作範例 (兩秒內)

(三) 進行實作

在提出研究方法之後本研究進行實作，主要是透過寫程式並結合 OpenCV 所提供之原始碼來實現，實作的過程是先建立影像資料庫，並定義出偵測區域；之後進行汽車防撞警示雷達系統的建立，並驗證本研究所提出基於樣板匹配之長距離立體影像模型建立法是否與實際算出距離相近，以及算出其誤差率和影像辨識造成的誤報率；最後將所有條件下所做出之結果進行分析，包含不同情況下的處理時間比較以及不同解析度比較等，找出其中最好結果並應用於系統中〔18〕〔19〕。

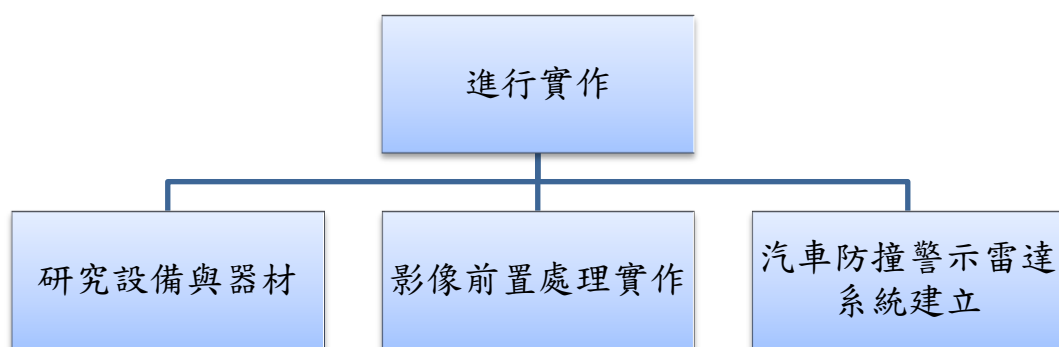


圖 51 進行實作流程圖

1. 研究設備與器材

下表 4 為本研究所使用之設備與器材：

表 4 實驗設備列表

實驗設備	型號	數量
網路攝影機	Logitech C920 (HD 1080P) 焦距 3.8 mm	2
電腦	ASUS G75W (i7-3610QM CPU@2.30GHz ; NVIDIA GTX670M)	1
電腦	ASUS A52J (i5-M460 CPU @2.53GHz ; NVIDIA GT 310M)	1
電腦	ASUS UX32VD (i5-3317U CPU @1.70GHz ; NVIDIA GT 620M)	1
1/14 公路模型	利用瓦楞紙以及珍珠板製作的 1/14 公路 模型主要為本研究實際拍攝雷達影像的 68 號快速公路。包括車子、人、機車、 腳踏車，以及動物。	1
捲尺	五公尺長，用於量測實際距離	1
長條木板	約 1 公尺長，用於裝兩個網路間距 90cm 之網路攝影機	1

下圖 52 至圖 59 為本研究所使用的器材與設備，以及於空曠的河濱公園
進行實作時的實際情形：



圖 52 網路攝影機



圖 53 電腦

	
圖 54 1/14 公路模型	圖 55 捲尺
	
圖 56 長條木板	圖 57 實驗時實際距離標示
	
圖 58 實驗情形	圖 59 實驗環境

2. 影像前置處理實作

在此階段，本研究將會利用先前設計出的流程進行實作，會實際進行攝影鏡頭光軸校正、建立影像資料庫，以及定義影像辨識區域，將會對這四個部分的實作過程詳細的紀錄，並將其步驟及得到的資訊呈現出來，如下內容所示：

(1) 攝影鏡頭光軸校正

因為雙攝影鏡頭以及感光元件並無法完美平行，於是大多會產生光軸偏轉的現象，在進行取像之前需將此修復，才可作後續的處理，以下為本研究整合公式(8)後的局部攝影機光軸偏轉修復程式的虛擬碼，完整的檔案於附錄 1-0：

```
double ideal_distance = 2000.0; // calibration object distance
double fL = 949.19078736281699; // Logitech C920 focal length value (Left camera)
double fR = 902.18669057603711; // Logitech C920 focal length value (Right camera)
double B_MUL_F_constant = ((fL + fR)/2) * 90; // 90 => two cameras offset
先設定校正物體放置的距離、兩攝影機的焦距(pixel)，以及公式(1)中 B*F 的值

double x2R = xR - 640.0; // 640 is the middle point of image resolution 1280*720
double x2L = xL - 640.0;
double x1L = ( B_MUL_F_constant / ideal_distance) + x2R;
double x1R = x2R;
找出兩影像中待測物體與影像中線的偏移距離，並且假設其中右相機光軸為理想狀態，再利用公式(1)即可算出左相機的理想偏移

double tan_theta_xR = (fR * (x1R-x2R)) / (x2R*x1R + fR*fR);
double tan_theta_xL = (fL * (x1L-x2L)) / (x2L*x1L + fL*fL);
將其帶入先前推導出之公式(8)，即可將相機的偏移角度計算出來，方便之後測距時能夠更為準確地得知距離資訊
```

(2) 建立影像辨識資料庫

本研究為滿足系統的需求，需自行建立不同物體的資料庫，可利用下列建立資料庫的步驟，將事先準備好的影像，即正向影像資料和負向影像資料進行資料庫的建立，即可得到物體偵測所需之影像辨識資料檔案 XML 檔，下列步驟為 OpenCV 提供的哈爾特徵資料庫建立方法，用電腦中的命令提示視窗進行操作 [17][20][21]。

建立資料庫

首先要建立一個 negative.dat 的檔案，當中包含所有負向影像的路徑，並且確定

此檔案以及其他正向影像的檔案皆位於同一個路徑內

```
createsamples -img face.png -num 10 -bg negatives.dat -vec samples.vec -maxxangle 0.6 -maxyangle 0 -maxzangle 0.3 -maxidev 100 -bgcolor 0 -bgthresh 0 -w 20 -h 20
```

接下來將單張正向影像產生出 10 張不同角度及亮度的影像，並且利用 negatives.dat 中的影像背景進行搭配，最後產生 samples.vec 的檔案

```
mergevec samples.dat samples_total.vec
```

若有多張正向影像產生多個.vec 檔案，則用此指令將可以把所有的.vec 檔融合，samples.dat 為是先產生之所有.vec 檔的路徑

```
haartraining -data haarcascade -vec samples_total.vec -bg negatives.dat -nstages 20 -nsplits 2 -minhitrate 0.999 -maxfalsealarm 0.5 -npos 7000 -nneg 3019 -w 20 -h 20 -nonsym -mem 512 -mode ALL
```

最後使用此指令建立資料庫，唯一需要調整的參數為npos(正向影像)以及nneg(負向影像)的值，若有十張正向照片而每張產生十張，則填入100；而負向影像則因 negatives.dat中的影像數目多寡而變，若當中有100個影像路徑(可重複)，則乘上建立samples的次數將會是nneg的數目，使用此指令後的執行時間可能需要很多天，甚至是一個禮拜都有可能

```
convert_cascade --size="20x20" haarcascade haarcascade.xml
```

將產生之haarcascade存入haarcascade.xml中供本研究的系統使用

本研究的汽車以及行人的資料庫將直接使用 OpenCV 和 University of Aveiro 已訓練完成之影像辨識資料 XML 檔，由於這些研究機構所提供之行人及車輛的辨識資料，均由 3000 張以上之正向影像資料和負向影像資料訓練所得，其物體偵測率已可達到本研究之需求，其他的部分則自行建立資料庫。

(3) 定義物體的辨識區域

利用前一章設計之流程定義出辨識區域後，利用程式定義此區域為影像辨識的區域，若辨識出之物體區域不在此區域內則當作誤報而不將此結果顯示，這能大大降低誤報率，以下為定義物體辨識區域的虛擬碼：

```
for(i=0;i<detect_num;i++) {  
    cRect_detect = objects[i];
```

```

detect_object_center_y = cRect_detect.y + cRect_detect.height/2;
detect_object_center_x = cRect_detect.x + cRect_detect.width/2;
x_lower=
(W/2+DETECT_AREA_X_OFFSET1)*(1.0-(double)((double)detect_object_center_y/(double)H));
x_upper=
(W/2+DETECT_AREA_X_OFFSET2)*(1.0+(double)((double)detect_object_center_y/(double)H));
if(detect_object_center_x >= x_lower && detect_object_center_x <= x_upper) {
    cRect_obj1[object_cnt] = cRect_detect;
    detect_obj_center[object_cnt].x =detect_object_center_x;
    detect_obj_center[object_cnt].y =detect_object_center_y;
    object_cnt++;
}
}

```

此迴圈中先找出偵測影像區塊的座標值，利用先前定義好的影像辨識區域，若此物體的中心點在此區內，則將其視為一辨識結果，若不在此區域內則不討論。

下圖 60 為本研究定義出之影像辨識區域，如圖中的汽車位於此區域內，於是進行辨識。



圖 60 定義物體影像辨識區域實作

3. 汽車防撞警示雷達系統建立

所有影像前置處理作業都準備就緒後，即可開始建立汽車防撞雷達系統，並開始針對其影像辨識及測距的功能作測試。本研究在實驗過程中為了使整個過程進行得更加順利，建立了一個 1:14 的公路模型，進而在上面標上刻度

來進行測量，並且包括人、摩托車、腳踏車以及動物等物體，也可用此測試此系統對於這幾種物體的反應，下圖 61 為整個公路模型的圖片，本研究在實驗過程中便會利用此模型以及實際公路所拍得的影像進行處理以及距離的實際量測，進而分析此方法的可行性：



圖 61 1:14 公路模型圖

本系統的建立最重要的過程為以下四點，包含即時的影像取得、偵測物體、樣板匹配法、及防撞警示的部分，在建立系統時，有無使用 GPU 及多重執行緒影像處理是執行速度提升的重要關鍵，在此章節將針對這幾點的步驟作詳細的說明並呈現出來〔22〕：

(1) 雙鏡頭取像

本研究在實驗階段取得影像資料所使用的程式只適用於靜態影像，但使用於汽車防撞警示雷達上就必須是動態的影像，由靜態轉換為動態最大的差別就是影像的處理速度，唯一需要做的步驟就是將影像存取及計算的程序交由圖形處理器作計算，並且利用多重執行緒的方式使這些動作同步處理，這樣即可大大提升影像處理的速度，所得到的動態影像即符合本系統的要求，以下為動態影像取得的虛擬程式碼。

```
VideoCapture capL(1);
```

```
VideoCapture capR(2);
```

首先利用 OpenCV 所提供的攝影機影像擷取指令，用此可方便的將多台攝影機的影像匯入電腦中做運算。

```
capL.set(CV_CAP_PROP_FRAME_WIDTH, 1920);
```

```
capL.set(CV_CAP_PROP_FRAME_HEIGHT, 1080);
```

```
capR.set(CV_CAP_PROP_FRAME_WIDTH, 1920);
```

```
capR.set(CV_CAP_PROP_FRAME_HEIGHT, 1080);
```

利用此指令可定義左右兩鏡頭所拍攝影像之解析度

```
cameraL_enable_capture = false;
```

```
cameraR_enable_capture = false;
```

```
cameraL_capture_done = false;
```

```
cameraR_capture_done = false;
```

要做同步處理前必須先設定很多標誌，當此標誌被啟動時便執行所對應程式

```
_beginthread(capture_camera_L, 0, (void *)NULL);
```

```
_beginthread(capture_camera_R, 0, (void *)NULL);
```

利用此指令可以使這兩個動作同步進行

```
while(1) {
```

```
    if(cameraL_enable_capture) {
```

```
        cameraL_capture_done = false;
```

```
        capL.read(cameraL_frame); // read a new frame from video
```

```
        cameraL_capture_done = true;
```

```
        cameraL_enable_capture = false;
```

```
    }
```

```
}
```

左右兩鏡頭同時使用此程序進行影像擷取，當結束後把影像擷取標誌關閉

下圖 62 為取得之連續影像：



圖 62 取得之連續影像圖

(2) 偵測物體

偵測物體時利用哈爾特徵法，本研究使用 OpenCV 所提供汽車以及行人的資料庫進行辨識，而自行建立資料庫的部分也有加入機車、腳踏車，以及動物等資料庫供本系統辨識。偵測物體時先判斷偵測到的物體是否在先前定義之影像偵測區域內，再辨認是否有與之前辨識到影像的重複，將重複的部分去除，即可找出最符合的物體位置，便依此來完成下一步的處理。以下為利用哈爾特徵法作影像辨識的虛擬程式碼，為了比較其處理速度，此過程將藉由使用 GPU 及不使用 GPU 的程式來實作〔23〕〔24〕〔25〕。

不使用GPU

```
String car_cascade_name = "training/cars3.xml";
```

```
CascadeClassifier car_cascade;
```

程式一開始先將 OpenCV 所提供之資料庫匯入

```
car_cascade.detectMultiScale( frame, objects, 1.1, 2, 0|CV_HAAR_SCALE_IMAGE,  
Size(25, 25));
```

使用哈爾特徵法作影像辨識時最主要的指令就是這一行，將變數 frame 與資料庫中的 object 比較，每比較一次即放大 1.1 倍，而 Size(25,25)代表辨識出的影像區域小於此大小便不考慮

```

for(i=0;i<detect_num;i++) {
    cRect_detect = objects[i];
    detect_object_center_y = cRect_detect.y + cRect_detect.height/2;
    detect_object_center_x = cRect_detect.x + cRect_detect.width/2;
    x_lower=
(W/2+DETECT_AREA_X_OFFSET1)*(1.0-(double)((double)detect_object_center_y/(double)H));
    x_upper=
(W/2+DETECT_AREA_X_OFFSET2)*(1.0+(double)((double)detect_object_center_y/(double)H));
    if(detect_object_center_x >= x_lower && detect_object_center_x <= x_upper) {
        cRect_obj1[object_cnt] = cRect_detect;
        detect_obj_center[object_cnt].x =detect_object_center_x;
        detect_obj_center[object_cnt].y =detect_object_center_y;
        object_cnt++;
    }
}

```

定義影像辨識區域(見上一章)

```

for(i=0;i<object_cnt;i++) {
    same_object = false;
    if(pre_x > 0 && pre_y > 0) {
        if(cRect_obj1[i].x > pre_x-cRect_obj1[i].width && cRect_obj1[i].x <pre_x
+ cRect_obj1[i].width && cRect_obj1[i].y > pre_y-cRect_obj1[i].height &&
cRect_obj1[i].y < pre_y+cRect_obj1[i].height) {
            same_object = true;
        }
    }
    if(same_object == false) {
        cRect_obj[object_cnt1] = cRect_obj1[i];
        object_cnt1++;
        pre_x = cRect_obj1[i].x;
        pre_y = cRect_obj1[i].y;
    }
}

```

此指令意在除去重複辨識同一物體的影像，運作方式為若第二物體的像素點與前一物體的同一像素點的差距只有在前一影像辨識區域的長與寬的範圍內，則電腦將其判斷為同一物體。

使用GPU

接下來之程序的使用方式與不使用GPU時皆相同，只是指令中皆多加了gpu的程序：

```
String car_cascade_name = "training/cars3.xml";
```

```
CascadeClassifier_GPU car_cascade_gpu;
```

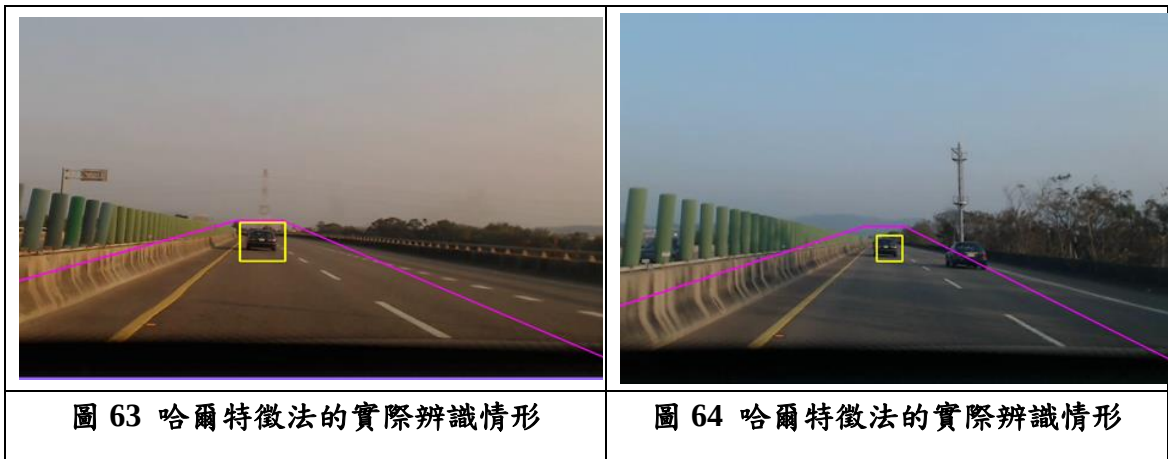
定義影像辨識資料庫

```
int detect_num = car_cascade_gpu.detectMultiScale(gray_gpu, objects, Size(300, 300), Size(50, 50), 1.3, 1);
```

利用哈爾特徵法辨識物體

定義物體辨識區域及去除重複物件與上述程序相同，於是不再作講解。

下圖 63 及圖 64 為哈爾特徵法實際運作狀況：



(3) 樣板匹配法

本研究使用樣板匹配法進行兩個影像的視差計算，希望能藉此計算

出與前方物體的距離，以下為樣板匹配的虛擬程式碼〔3〕。

```
int match_method = CV_TM_CCORR_NORMED;
```

定義樣板匹配時所使用的計算方法

```
Image = Image_grey(Rect(550,330, 200, 140));
```

```
Template = Image_grey(Rect(650,380, 100, 70));
```

定義樣板的位置及大小

```
matchTemplate( Image_grey, Template, result, match_method );
```

最主要的執行指令，利用 Template 再 Image_grey 的範圍內進行比對，將結果放

入 result 中

```
double minVal; double maxVal; Point minLoc; Point maxLoc;
Point matchLoc;
minMaxLoc( result, &minVal, &maxVal, &minLoc, &maxLoc, Mat() );
matchLoc = maxLoc;
```

把比對過後相似度最大的結果才當成同一物體，之後再計算出偏移的距離(即為視差)

圖 65 及圖 66 為樣板匹配實作的實際情形：



圖 65 樣板匹配比對區域定義



圖 66 比對區域(灰階)及樣板

用樣板匹配法找出兩影像的視差值之後，在距離計算的部分，可用先前計算出之相機偏轉角度將光軸偏轉後相機拍出之影像的視差修復為最理想狀態的視差，這可使距離計算變得十分準確，且將立體影像模型建立的適用範圍提升至 100 公尺，使其應用層面更廣，以下為距離計算的虛擬碼。

距離計算

```
FileStorage fs;
fs.open("calibration_file_5000.xml", FileStorage::READ);
將先前計算校正好的檔案開啟

double fL, fR;
double tan_theta_R, tan_theta_L;
fs["tan_theta_xR"] >> tan_theta_R;
fs["tan_theta_xL"] >> tan_theta_L;
fs["fL"] >> fL;
fs["fR"] >> fR;
```

將當中的偏轉角度的值讀入，並且將兩相機的焦距長度值也讀入，利用這樣的程序可方便實驗過程使用不同的校正檔案以及不同的影像解析度

```
double B_MUL_F_constant = ((fL + fR)/2) * 90;
```

```
double xR_test = matchLoc.x;
```

```
double xL_test = box.x;
```

```
double x2R_test = xR_test - 640.0;
```

```
double x2L_test = xL_test - 640.0;
```

找出左右影像中物體點與影像中線的位移

```
double x1R_test = fR * ((x2R_test + fR * tan_theta_R) / (fR - x2R_test * tan_theta_R));
```

```
double x1L_test = fL * ((x2L_test + fL * tan_theta_L) / (fL - x2L_test * tan_theta_L));
```

利用本研究提出的演算法則將失真的位移修復，使其能更精確的計算距離

```
double disparity = x1L_test - x1R_test;
```

```
double distance = B_MUL_F_constant*0.01/ disparity;
```

計算出視差，並且利用公式(1)計算出距離

(4) 防撞警示

防撞警示的建立，依據先前定義的情況，判斷在何種條件下需要進

行警示，以下為其虛擬程式碼：

```
if(distance < prev_distance) {  
    if(image_seq_no-danger_start > 20) danger_cnt = 0; else danger_cnt++;  
    danger_start = image_seq_no;  
}  
else {  
    danger_cnt = 0;  
}
```

當偵測距離小於前一個距離時，逼近本車次數(danger_cnt)增加；而image_seq_no是用來計算距離減少的情況是否為一個連續狀態，若不是則將danger_cnt重置

```
bool had_fired = false;
```

```
if(prev_distance-distance > 2.0) {alarm_fired = true; had_fired = true;}
```

```
if(distance <= 10 && had_fired == false) {alarm_fired = true; had_fired = true;}
```

```
if(danger_cnt > 3 && had_fired == false) alarm_fired = true;
```

alarm_fired 為啟動警示的程序，而當它因為前一個條件而執行警示(had_fired)時，則後面都不再進行警示

本系統的建立即是將實作中所有的步驟依照圖 48 的流程，整合成為一個大型的程式，而最終的整合測試模擬將會在成果分析的部分進行探討。

三、 研究結果與討論

在研究結果與討論的部分將會對於實作過後所得到的結果進行分析比較，包含影像處理的速度、不同解析度的偵測率以及本研究計算距離的誤差率等。接著將得到的結果列成表格，並對這部分進行討論，最後再將全部的程式整合。下圖 67 為研究結果與討論的組織圖：

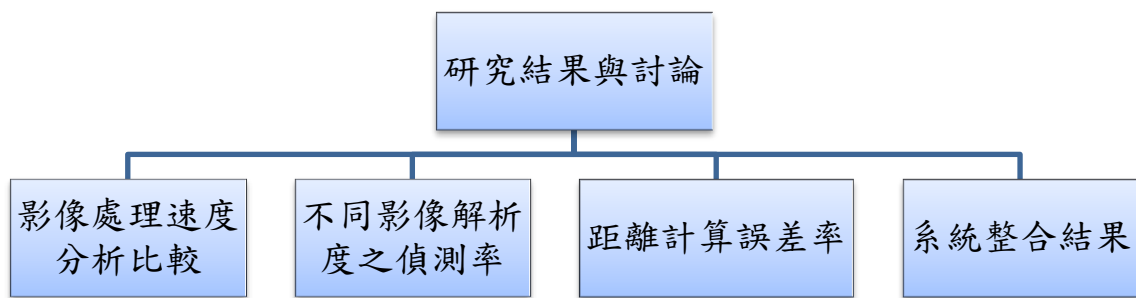


圖 67 研究結果與討論組織圖

(一) 影像處理速度分析比較

在經過實作之後，本研究對於在不同電腦規格下，哈爾特徵法以及樣板匹配法執行時有無圖形處理器，及使用不同影像解析度時的影像處理的速度來進行比較，下表 5 為實驗過程中蒐集到的影像處理時間數據，透過對這些數據進行分析，希望找到最有效率的影像處理方法，並且將其應用於本研究最後建立的系統上。

表 5 執行時間比較表 (CPU: i7-3610 ; GPU: GTX-670M)

解析度(pixel)	320x240	640x480	1280x720	1920x1080
哈爾特徵法不使用 GPU 處理時間(ms)	37.156	123.874	302.476	543.68
哈爾特徵法使用 GPU 處理時間(ms)	N/A	24.469	55.369	107.303
樣板匹配法不使用 GPU 處理時間(ms)	42.85	120.646	251.609	427.963
樣板匹配法使用 GPU 處理時間(ms)	N/A	10.81	19.467	25.724

由上表 5 中可以看到不同解析度的影像執行哈爾特徵法之特徵點抽取時所花費的時間差距可以說是十分大，不使用圖形處理器時甚至差距可達 15 倍之多，而當把此程序交由圖形處理器運算時，速度也較不使用時相對加快了 5 倍，於是若

只從此表中所得的結果來比較，640x480 以及 1280x720 兩種解析度搭配圖形處理器較符合本研究的需求。使用圖形處理器於影像解析度 320x240 的影像上時沒辦法有所作用，因為此影像的解析度太小，並無進行圖形處理器處理的價值，所以在此並不討論。

本研究在執行樣板匹配的部分採用定義部分區域偵測的方式以及 GPU 處理，進而提升其影像處理速度，在表 6 中針對樣板匹配時的執行速度進行比較，可以發現在不使用與使用圖形處理器的情況下，影像處理速度相差最大可達 15 倍之多。下表 6-7 為另外兩種電腦規格所量測到的執行速度比較：

表 6 執行時間比較表 (CPU: i5-3317U ; GPU: GT 620M)

解析度(pixel)	320x240	640x480	1280x720	1920x1080
哈爾特徵法不使用 GPU 處理時間(ms)	16.7605	105.433	365.744	730.394
哈爾特徵法使用 GPU 處理時間(ms)	N/A	37.2519	109.671	250.955
樣板匹配法不使用 GPU 處理時間(ms)	19.8871	77.1185	215.335	524.314
樣板匹配法使用 GPU 處理時間(ms)	N/A	16.4562	29.5708	50.0828

表 7 執行時間比較表 (CPU: i5-M460 ; GPU: GT-310M)

解析度(pixel)	320x240	640x480	1280x720	1920x1080
哈爾特徵法不使用 GPU 處理時間(ms)	18.0032	89.0339	264.703	558.139
哈爾特徵法使用 GPU 處理時間(ms)	N/A	71.1146	153.292	224.003
樣板匹配法不使用 GPU 處理時間(ms)	28.877	84.1689	292.465	681.657
樣板匹配法使用 GPU 處理時間(ms)	N/A	50.9784	88.5615	156.855

(二) 不同影像解析度之辨識率

從比較不同解析度的影像處理速度所得到的結果並不能直接找出何種解析度適用於本研究建立的系統中，影像的辨識率也可說是非常重要的一環，於是本研究也對於這方面進行比較，以四種解析度的影像在不同距離下是否能辨識到物體為基準，建立下面的表格(表 8)，以找出能在各種距離下完全辨識出物體位置的影像解析度大小為佳。(V 代表可以偵測到物體；X 代表無法偵測到物體)

表 8 不同影像解析度在不同距離下之辨識率比較
(CPU: i5-M460 ; GPU: GT-310M)

	20m	30m	40m	50m	60m	70m	80m	90m	100m
320x240	V	V	X	X	X	X	X	X	X
640x480	V	V	V	V	V	V	X	X	X
1280x720	V	V	V	V	V	V	V	V	V
1920x1080	V	V	V	V	V	V	V	V	V

由表 8 中可以看出 1280x720 及 1920x1080 這兩種解析度大小的影像可以在各種距離下偵測出物體的所在位置，這兩種解析度較符合本研究針對此部分的需求。

(三) 距離計算誤差率

本研究在使用特殊演算法則計算出光軸偏轉的角度後，並在計算距離時將其校正回來，再利用其與實際距離的偏差計算出誤差率，而實際距離則是利用一條捲尺從汽車後面測量並作比對，實驗中可能因為量測時並沒有拉直捲尺或是其他的環境因素而有些許的誤差，預估的誤差率約為 1%，還在可以接受的範圍內。但是依據文獻中提到的距離計算方式，其量測到的視差值必為整數，這就造成了同一個視差值代表著不同距離的情況，而這現象又與影像解析度有關，於是下表 9 至 11 為探討不同解析度的情形下立體視覺的精度，代表著量測距離可能對到不同的實際距離，於是在進行誤差率計算時必須將此考慮進去，下表 12 至 14 為在實際道路上量測距離的誤差率的比較。依據不同解析度的辨識率比較，320x240 的影像解析度只能夠對於 20 及 30 公尺的車輛進行辨識，於是並不適用於更長距離測距，在此並不作討論。(下表 9-表 17 為 CPU: i5-M460 ; GPU: GT-310M)

表 9 影像解析度 640*480 立體視覺精度測試

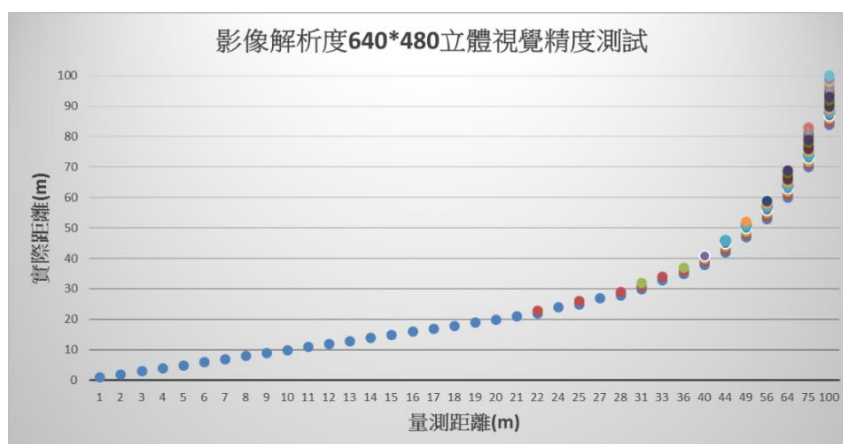


表 10 影像解析度 1280*720 立體視覺精度測試

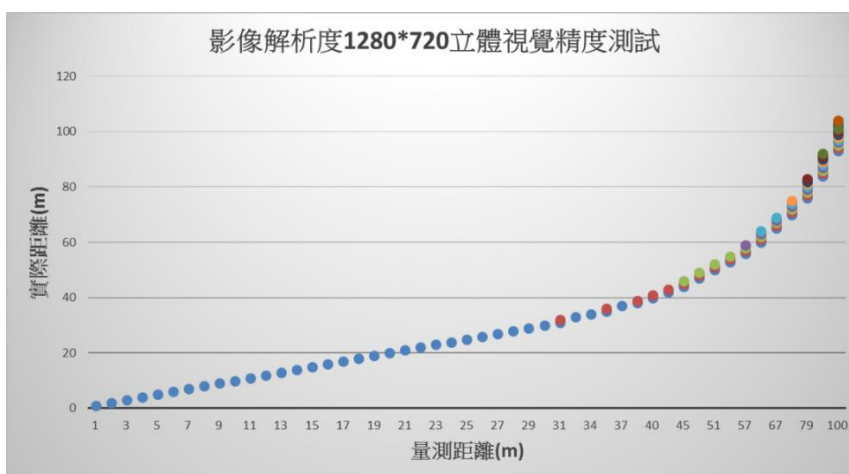


表 11 影像解析度 1920*1080 立體視覺精度測試

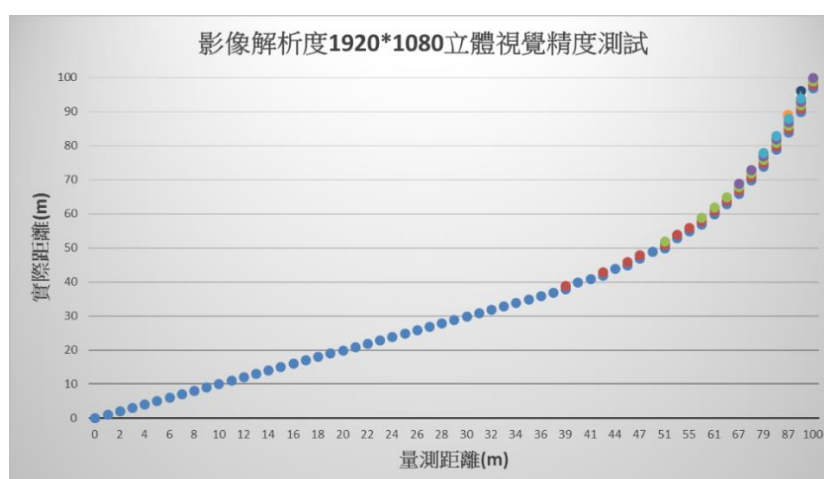


表 12 影像解析度 640*480 在實際道路進行距離量測的誤差比較

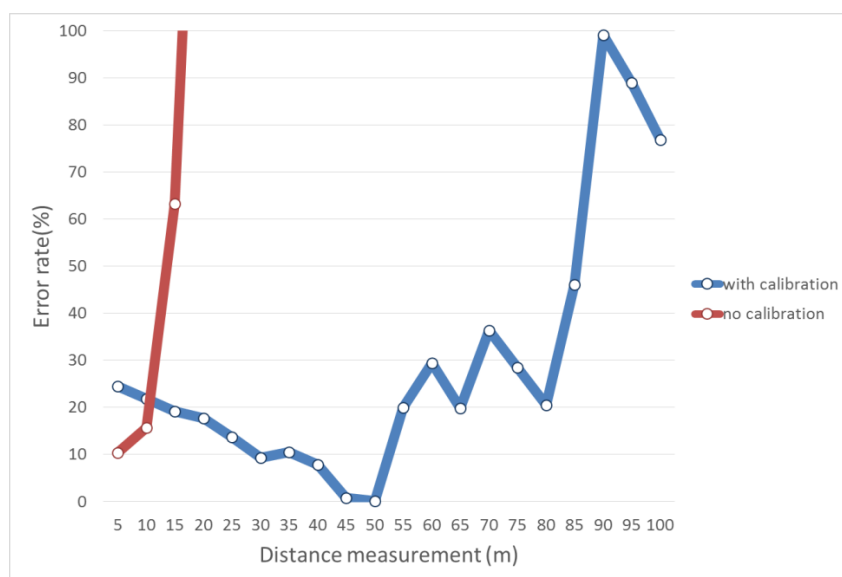


表 13 影像解析度 1280*720 在實際道路進行距離量測的誤差比較

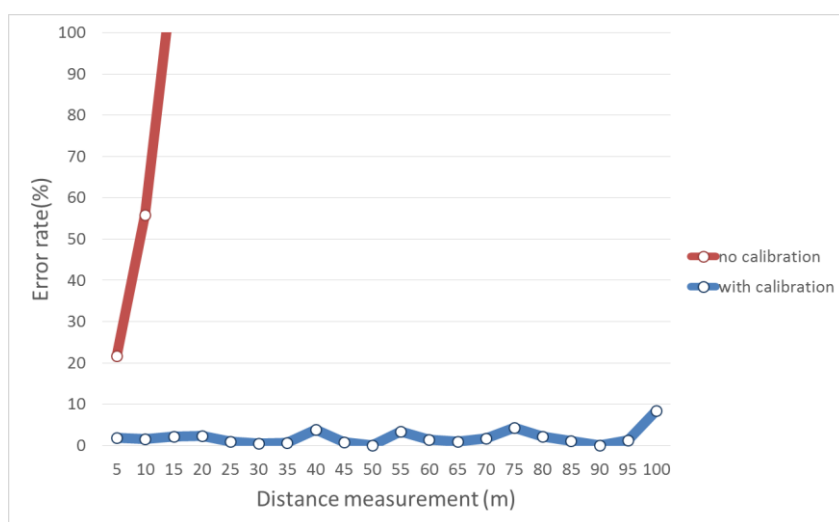
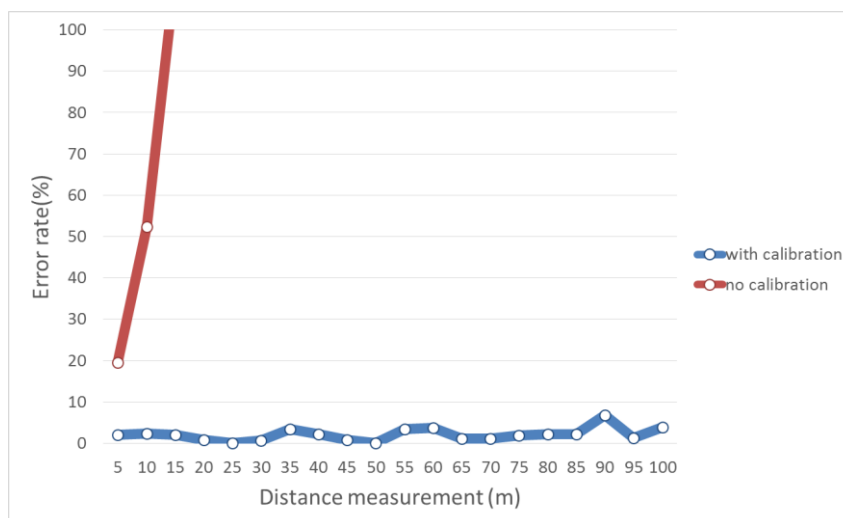


表 14 影像解析度 1920*1080 在實際道路進行距離量測的誤差比較



本研究在不同解析度的距離計算誤差率比較後發現 1280*720 及 1920*1080 兩種解析度對於此方面都可以符合系統的需求。

這三種影像解析度在未經相機光軸校正的情況下，誤差都顯得非常大，經過光軸校正後得知的偏轉角度約為 3 度，於是便在未校正的這個情況使用了偏轉 3 度的未校正模型進行距離量測，此結果顯示光軸偏轉的確對長距離測具有極大的影響，且從上述結果也可得知其實對於高解析度影像在不同距離校正對於距離計算誤差率的影響並不大，由上述這兩點可證明本研究所提出之相機光軸校正演算法則的可行性。

而本研究也對於自行建立的公路模型做一連串的距離量測以及誤差率的比較，表 15-表 17 為其結果：

表 15 影像解析度 640*480 在公路模型進行距離量測的誤差比較

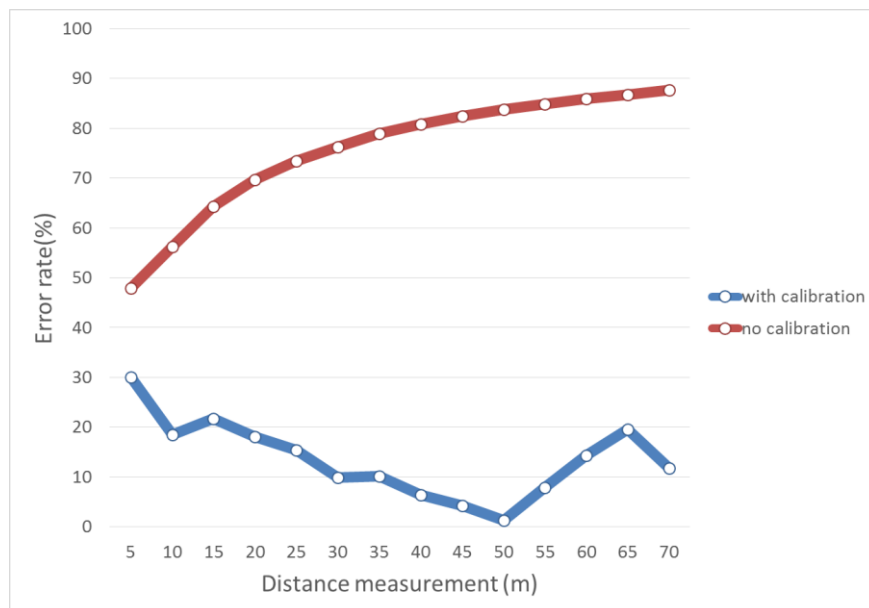


表 16 影像解析度 1280*720 在公路模型進行距離量測的誤差比較

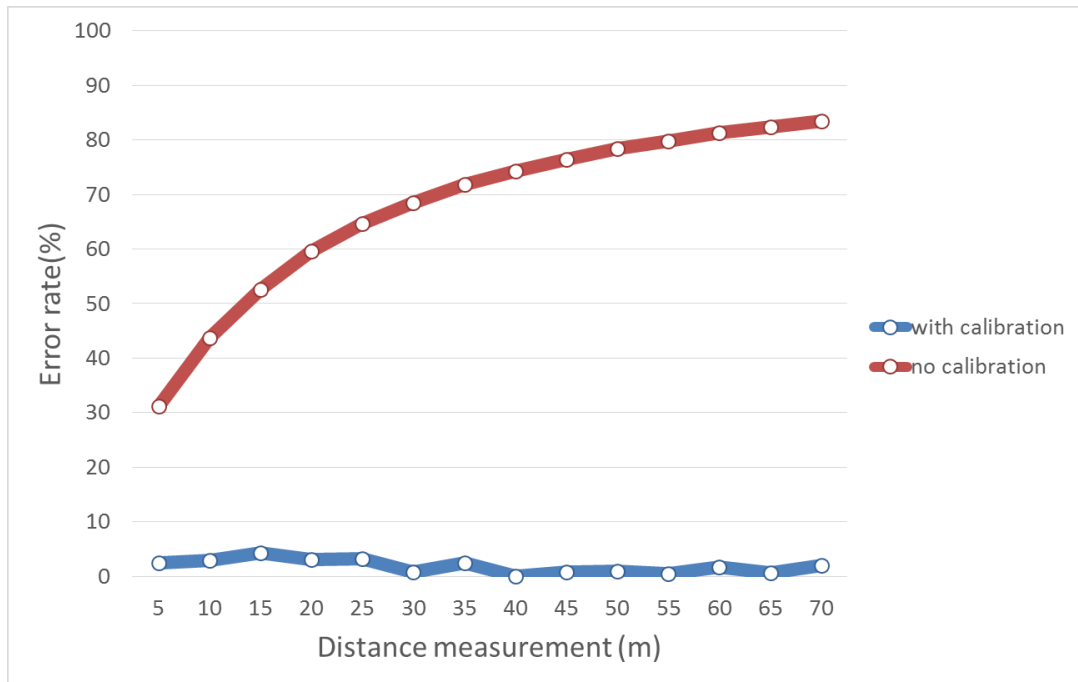
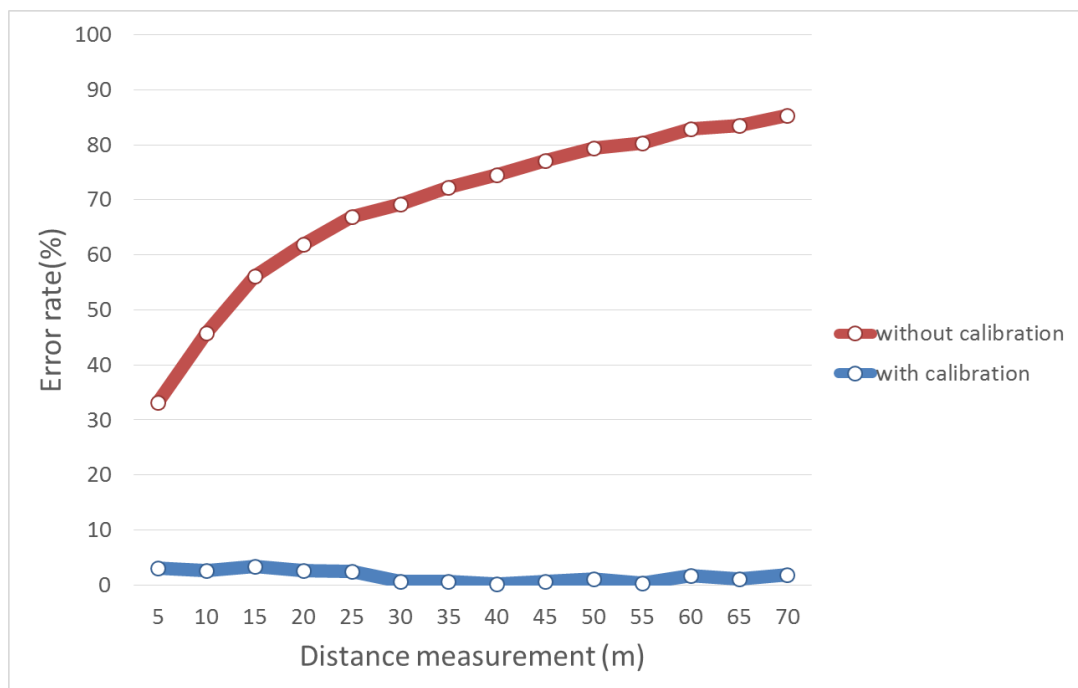


表 17 影像解析度 1920*1080 在公路模型進行距離量測的誤差比較



由上述的誤差比較數據可發現低解析度的影像較不適用本研究的相機光軸校正演算法則，而高解析度的影像在經過光軸校正後距離量測的誤差率則是大約都小於 5%，這即證明了此演算法則的可行性。

經過了多次的比較以及不同步驟的程序分析，本研究最後決定使用影像解析度 1280x720 的影像用於防撞警示系統中，相信這樣就能夠符合本研究的需求，達到處理速度快、辨識率高以及誤差率低等條件。

(四) 系統整合結果

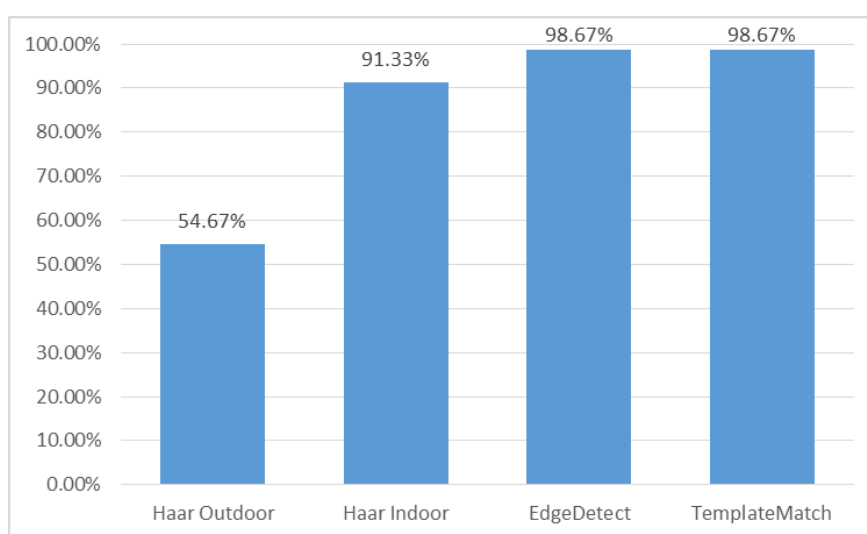
在結束所有的實作以及將結果分析比較並找出最適合的影像解析度後，本研究便開始建立防撞警示雷達系統，建立時需要把所有步驟的程式碼整合，並且結合先前提到的多重執行緒，能將整個系統的處理時間縮短，這樣能符合此防撞警示雷達系統要求的即時性。本研究目前建立的模型為鏡頭間距 9cm 及鏡頭焦距 3.8mm 並搭配電腦所建立的系統，如下圖 69。但由於此為 1:14 的模型，所以實際應用在汽車上的系統都必須要放大十四倍，本研究目前打算將兩相機放置於相對於車燈寬度的位置，即可滿足所有需求，此系統的完整程式碼於附錄 2-0。



圖 69 本研究建立之汽車防撞警示雷達系統模型

在進行系統整合的時候，對於物體偵測的辨識率將會是一個很大的考量，於是本研究加入了一個邊緣偵測的方法，針對特定的物體-汽車，提升物體的偵測率，如下表 18 為哈爾特徵法在室內室外時的物體偵測率，以及邊緣偵測法的偵測率和樣板匹配法的配對成功率：

表 18 不同技術之物體偵測率及配對成功率



可從表 18 中看出，在室內進行哈爾特徵法物體辨識時偵測率也可高達 91.33%，這可證明哈爾特徵法的可行性，而在室外進行物體偵測時可能會因為背景光源等因素影響，所以會造成偵測率下降，在未來延伸研究時，會再自行建立更完善的資料庫，並且對於特殊物體搭配利用影像處理(邊緣偵測)的方式，可使辨識率接近 99.9%。

本研究在系統執行的部分只會對局部的區域進行物體偵測以及樣板匹配，所以下表 19 的影像執行時間較表 5-表 7 的時間短，且下表中將邊緣偵測法加入此系統，其目的是要使得本研究在辨識汽車時的偵測率提升，表 19 為不同偵測技術以及樣板匹配法的系統執行速度表：

表 19 不同技術之物體偵測及樣板匹配法的執行速度

電腦型號	Harr (ms)	Edge Detect (ms)	Template Match (ms)
i7 CPU& GTX670M	32.80193	60.94826375	14.53824
i5 CPU& GT310M	43.68062	80.39313	12.350598
i5 CPU& GT620M	24.1065	45.0948	12.7258

本研究使用影像解析度 1280x720 的影像，結合多重執行緒的處理方式，完整從擷取影像到計算距離所花費的時間，依據表 19，約為 70ms，所以最終的汽車防撞警示雷達系統每秒鐘可以拍攝約 15 張影像並計算其距離，這樣已經可以達到雷

達所要求的即時性。

接下來實際將此系統應用於汽車上，並利用兩個時間所拍攝到的影像，嘗試計算與前車的相對距離，如下圖 70 及圖 71 為汽車防撞警示雷達運作情況，先將拍攝影像間隔設為 1 秒鐘。



圖 70 汽車防撞警示雷達系統 1 秒前所拍攝到的影像(距離本車 41.90 公尺)



圖 71 汽車防撞警示雷達系統 1 秒後所拍攝到的影像(距離本車 49.22 公尺)

可以從圖片中看出，1 秒後與前車的距離增加了 7.32 公尺，代表本車相對於前車減少了 7.32m/s，圖 70 及 71 中的例子並沒有符合本研究定義的警示條件，於是不會進行防撞警示。

若有摩托車、腳踏車、動物及一些物體時，此系統會立刻進行警示，這使得此研究所建立的系統更加完善，在辨識到這些特殊物體時，即警示駕駛人，這可大大降低車禍發生的機率。

四、 結論與未來展望

本研究在經過實作以及結果分析之後，對於利用長距離的立體影像模型建立有了相當大的突破，可將得到的結論分為以下五點：

(一) 利用雙鏡頭建立長距離立體影像的可行性高

根據實驗數據的結果，利用雙鏡頭再配合本研究所提出之演算法則，計算出的距離資訊與實際距離的誤差率相當低，而使其誤差率低的原因就在於影像處理前半段會先進行影像的辨識，之後再利用樣板匹配的方式進行視差值的計算，以及本研究所提出之光軸校正的方法。相較於單鏡頭進行長距離立體影像模型建立，使用雙鏡頭進行長距離立體影像的準確度更高，於是本研究的汽車防撞警示雷達系統則使用雙鏡頭，即能符合需求。

(二) 相機光軸校正演算法則的準確性以及可行性

本研究提出之相機光軸校正演算法則不僅能夠在使用相機前先算出其光軸的偏轉角度，在進行距離量測時也能夠捨去立體匹配以及矯正映射等繁瑣的步驟，僅利用樣板匹配的技術即可準確的算出與前方的距離。由研究結果中可以發現在高解析度的影像中使用光軸校正演算法則進行距離的量測，其誤差率都在 5% 以下，這證明了此演算法則的準確度及可行性。

(三) 利用影像處理的方式可使感興趣物體區域偵測率提升

本研究所建立的汽車防撞警示雷達系統首先需要利用物體偵測的技術找出感興趣物體的所在位置，而在此使用的技術為哈爾特徵法，但是根據表(18)，哈爾特徵法在戶外實際使用的偵測率約為 55%，於是在研究中也發現，若是對於特殊的物體，如汽車進行辨識時，可利用影像處理的方法，透過物體的輪廓進行辨識。本研究將兩個物體偵測技術同時使用，物體的偵測率可高達 98%，所以在此發現

利用影像處理的方式進行物體偵測可提高辨識率。

(四) 結合圖形處理器與多重執行緒可提升高解析度立體影像的處理速度

根據文獻中公式(1)指出，影像的解析度也是影響是否能做遠距離物體的距離量測的重要關鍵。而從實驗中所得到的數據顯示，不同的影像解析度進行特徵點抽取、樣板匹配等步驟時所得到的結果也會不大相同。高解析度的影像進行特徵點抽取以及樣板匹配時的偵測率會提升，但是其影像處理的速度與低解析度影像處理的速度相較之下比較慢，於是本研究做了多次分析比較後決定先使用 1280x720 以及 1920x1080 等高影像解析度進行長距離立體影像模型的建立，而高解析度所造成的處理速度問題將藉由使用圖形處理器及多重執行緒兩種技術來解決。

本研究的結果發現使用多重執行緒的確能夠使得程序在執行上同步進行，讓影像處理的時間都能夠疊加，而搭配圖形處理器和多重執行緒執行的程式之影像處理速度與不使用的情形下，兩者相差最大有 10 倍之多，若要將本研究提出之長距離立體影像建立方法應用於需要即時性的雷達系統上，使用這兩種技術即能符合需求。

(五) 使用 OpenCV 提供之開放原始碼實現系統程序

本研究所有的程序皆使用 OpenCV 所提供之開放原始碼實現，這樣的一個平台是由大家放上不同的原始碼做影像處理，且除了提供原始碼，還有對於如何使用此種功能的敘述，於是對於本研究要實現之汽車防撞警示雷達系統可說是相當的便利，讓這樣一個複雜的程序能夠簡單的被實現。OpenCV 也使得研究者並不需花太多的時間探討如何實現現有的影像處理技術，而是專注在開發全新的演算法則，最後可再把研究成果也提供在此平台上，方便大家使用。

(六) 未來展望

本研究實現了所提出的長距離立體影像建立演算法則，對於未來應用、改良

以及延伸研究的方面主要分為以下三點：

1. 增加物體偵測種類及方式

當系統能夠偵測到的物體種類增加，長距離立體影像模型即可提供更多的資訊，對實際應用面將有更大的貢獻。物體偵測的方式可再結合影像處理的方法，使得更多物體的偵測率能夠提升。

2. 改善現有機器人的視覺系統

目前的機器人視覺系統的適用範圍是在 10m 以內，而將此系統中原有的雙鏡頭結合本研究的演算法則能將機器人視覺系統升級，使其能夠對於 80 公尺以上物體進行偵測以及立體影像模型建立，讓這個系統的功能能夠更多元、更強大。

3. 發展無人自動駕駛系統

以影像處理法所設計的汽車防撞警示雷達與毫米波雷達各有其優缺點，最佳的方案應為合併兩者的優點，進而實現最佳的汽車防撞警示雷達系統，甚至能夠將此技術發展成為自動駕駛系統的汽車，由影像處理偵測前方物體的種類以及距離，再由毫米波雷達進行更進一步的測距，這樣就能夠解決毫米波雷達容易誤報的問題。對於突然出現的物體也可由影像雷達偵測，並且緊急煞車，相信這些都是本研究長距離立體影像模型建立的方法能夠對自動駕駛系統的貢獻，配合日漸進步的影像辨識技術，此系統所提供的資訊將會變得更有價值。

五、 參考文獻

- [1] EL Samuko , 2013 , ”OpenCV Depthmap” , elsamuko(C++)
- [2] OpenCV , ” Template Matching” , OpenCV 2.4.6.0 documentation(OpenCV Tutorials)
- [3] Andrew Burnes , 2012 , ”Introducing The GeForce GTX 680M Mobile GPU” , GeForce
- [4] 菲謝蒂 , 2008 , ”聰明的自動駕駛” , 科學人雜誌第 71 期 1 月號 P.121
- [5] Buytheway , 2012 , ”拋棄式 3D 眼鏡貼紙 , 告別六眼田雞的窘境”
- [6] Katrina Sokolova , Barak Shilo , 2006 , ” Experiments in Stereo Vision ” , University of Tsukuba
- [7] Richard Wang , ”攝像頭校正 camera calibration” , Richard’s Blog , 2012
- [8] Wikipedia , 2007 , ” Epipolar geometry”
- [9] Guido Gerig , 2012 , ”Image Rectification (Stereo)” , Scientific Computing and Imaging Institute
- [10] Martin Peris , 2011 , ” OPENCV: STEREO MATCHING ” , Martin Peris’ Blog
- [11] Wikipedia , 2013 , ” Time of flight camera”
- [12] 國立政治大學資訊科學系 , “Human computer Interaction : KINECT Sensor”
- [13] Michael Jones 、 Paul Viola , 2001 , ”Rapid Object Detection using a Boosted Cascade of Simple Features” , Labs One Cambridge Center 、 Mitsubishi Electric Research
- [14] 林建忠 , 1999 , ”雷射測距技術與研究現況” , 光電科技工業協進會

- [15] 江之川，2012，平價車竟搭載毫米波防撞雷達，CTIMES 網
- [16] AutomotiveIT，2011，”Continental develops stereo camera to improve car safety systems”
- [17] Andreas01，”Tutorial: OpenCV haartraining (Rapid Object Detection With A Cascade of Boosted Classifiers Based on Haar-like Features)” ， Naotoshi Seo(technotes, OpenCv)
- [18] OpenCV，” OpenCV(Open Source Computer Vision)” ， OpenCV homepage
- [19] OpenCV，” Class VideoCapture” ， OpenCV 2.4.6.0 documentation
- [20] Google Project Hosting，2012，”opencv-lane-vehicle-track”，Google
- [21] The Laboratory for Automation and Robotics，2008，” Index of /public/Media/HaarFeatures_RoadFilms/HaarFeaturesTests/CarsRear” ， Department of Mechanical Engineering at the University of Aveiro
- [22] Microsoft，2013，” Multiple Threads” ， Desktop Technologies
- [23] OpenCV，” Cascade Classifier Training” ， OpenCV 2.4.6.0 documentation(OpenCV User Guide)
- [24] OpenCV，”Haar Feature-based Cascade Classifier for Object Detection” ， OpenCV 2.4.6.0 documentation(OpenCV API Reference)
- [25] Paul Viola and Michael J. Jones.，2001，”Rapid Object Detection using a Boosted Cascade of Simple Features” ， IEEE CVPR
- [26] OpenCV，”GPU Module Introduction” ， OpenCV 2.4.6.0 documentation(OpenCV API Reference)

附錄

附錄 1-0 (camera_cal.cpp)

```
// new_camera_cal.cpp : Defines the entry point for the console application.
//

#include "stdafx.h"
#include <iostream>
#include <sstream>
#include <time.h>
#include <stdio.h>
#include <stdlib.h>

#include "cxcore.h"
#include "highgui.h"
#include "cv.h"
#include <opencv2/opencv.hpp>
#include <opencv2/core/core.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <opencv2/imgproc/imgproc.hpp>
#include <opencv2/calib3d/calib3d.hpp>

#define CAPTURE
#define DEMO
#define IMG_RES 1280
#define CAL_D 5000

using namespace cv;
using namespace std;

CvRect box;
bool    drawing_box = false;
int     mouse_x, mouse_y;

void draw_box( Mat img, CvRect rect ){
    rectangle( img, cvPoint(box.x, box.y),
```

```

cvPoint(box.x+box.width,box.y+box.height),
        cvScalar(0xff,0x00,0x00), 3, 8, 0 );
}

void my_mouse_callback( int event, int x, int y, int flags, void *param )
{
    Mat *image = (Mat *) param;

    mouse_x = x;
    mouse_y = y;

    switch( event ) {
        case CV_EVENT_MOUSEMOVE:
            if( drawing_box ){
                box.width = x-box.x;
                box.height = y-box.y;
            }
            break;

        case CV_EVENT_LBUTTONDOWN:
            drawing_box = true;
            box = cvRect( x, y, 0, 0 );
            break;

        case CV_EVENT_LBUTTONUP:
            drawing_box = false;
            if( box.width < 0 ){
                box.x += box.width;
                box.width *= -1;
            }
            if( box.height < 0 ){
                box.y += box.height;
                box.height *= -1;
            }
            draw_box( *image, box );
            break;
    }
}

```

```

}

int _tmain(int argc, _TCHAR* argv[])
{
    Mat frameL_grey, frameR_grey;
    Mat frameL, frameR;

    cvNamedWindow("WebcamL",1);
    cvNamedWindow("WebcamR",1);

#ifdef CAPTURE-
    VideoCapture capL(2);
    VideoCapture capR(1);

    capL.set(CV_CAP_PROP_FRAME_WIDTH, IMG_RES);
    capR.set(CV_CAP_PROP_FRAME_WIDTH, IMG_RES);

    if(IMG_RES == 640){
        capL.set(CV_CAP_PROP_FRAME_HEIGHT, 480);
        capR.set(CV_CAP_PROP_FRAME_HEIGHT, 480);
    }
    else{
        capL.set(CV_CAP_PROP_FRAME_HEIGHT, IMG_RES*0.5625);
        capR.set(CV_CAP_PROP_FRAME_HEIGHT, IMG_RES*0.5625);
    }

    while(1) {
        capL.read(frameL);
        capR.read(frameR);
        imshow("WebcamL", frameL);
        imshow("WebcamR", frameR);
        if(waitKey(0) == 27) break;
    }
#else
    char fullnameL[100], fullnameR[100];
    sprintf(fullnameL, "c:/temp/img/imageL050-1280-14-20-19-746.jpg");

```

```

sprintf(fullnameR, "c:/temp/img/imageR050-1280-14-20-19-746.jpg");
frameL = imread(fullnameL, 1);
frameR = imread(fullnameR, 1);
imshow("WebcamL", frameL);
imshow("WebcamR", frameR);

waitKey(0);
#endif

cvtColor(frameL, frameL_grey, CV_BGR2GRAY);
cvtColor(frameR, frameR_grey, CV_BGR2GRAY);

cvSetMouseCallback("WebcamL", my_mouse_callback, &frameL);

Mat temp;
char dis_str[100];
while(1) {
    frameL.copyTo(temp);
    if( drawing_box ) draw_box( temp, box );
    sprintf(dis_str, "x=%d y=%d", mouse_x, mouse_y);
    putText(temp, dis_str, cvPoint(320, 100), FONT_HERSHEY_COMPLEX_SMALL,
1.2, cvScalar( 0, 0, 255), 1, CV_AA);
    imshow("WebcamL", temp);
    if( cvWaitKey(50)==27 )
        break;
}

Rect cRectR1;
Rect cRectL1;

cRectL1.x = box.x * 1.0;
cRectL1.y = box.y * 1.0;
cRectL1.width = box.width * 1.0;
cRectL1.height = box.height * 1.0;
Mat L = frameL_grey(box);

cRectR1.x = box.x - 6*box.width; if(cRectR1.x < 0) cRectR1.x = 0;
cRectR1.y = box.y - box.height * 1.0; if(cRectR1.y < 0) cRectR1.y = 0;

```

```

cRectR1.width = box.width * 8.0;
cRectR1.height = box.height * 2.0;
Mat R = frameR_grey(cRectR1);

Mat result;
int match_method = CV_TM_CCORR_NORMED;

matchTemplate( frameR_grey, L, result, match_method );
normalize( result, result, 0, 1, NORM_MINMAX, -1, Mat() );

/// Localizing the best match with minMaxLoc
double minVal; double maxVal; Point minLoc; Point maxLoc;
Point matchLoc;

minMaxLoc( result, &minVal, &maxVal, &minLoc, &maxLoc, Mat() );
matchLoc = maxLoc;

Rect R_box;
R_box.x = matchLoc.x;
R_box.y = matchLoc.y;
R_box.width = box.width ;
R_box.height = box.height;

rectangle(frameR, R_box, Scalar( 0, 255, 255 ), 2, 8, 0);
imshow("WebcamR", frameR);
char idx[100];

#ifdef DEMO

int base = 9;
double ideal_distance = CAL_D / 13.986014;
sprintf(idx, "demo");

#else

int base = 90;
double ideal_distance = CAL_D;
sprintf(idx, "car");

```

```

#endif

double xR = matchLoc.x;
double xL = box.x;
double fL = 949.19078736281699/1280*IMG_RES;
double fR = 902.18669057603711/1280*IMG_RES;
double B_MUL_F_constant = ((fL + fR) / 2) * base;

double x2R = xR - IMG_RES/2;
double x2L = xL - IMG_RES/2;
double x1L = ( B_MUL_F_constant / ideal_distance) + x2R;
double x1R = x2R;

double x1_test = x1L - x1R;

double tan_theta_xR = (fR * (x1R-x2R)) / (x2R*x1R + fR*fR);
double tan_theta_xL = (fL * (x1L-x2L)) / (x2L*x1L + fL*fL);

printf("tan_theta_xR = %f\n", tan_theta_xR);
printf("tan_theta_xL = %f\n", tan_theta_xL);

waitKey(0);

char filename[100];
sprintf(filename, "C:/Temp/CAL_file/calibration_file_%d_%d_%s.xml" , CAL_D,
IMG_RES, idx);
FileStorage fs;
fs.open(filename, FileStorage::WRITE);

fs << "tan_theta_xR" << tan_theta_xR;
fs << "tan_theta_xL" << tan_theta_xL;
fs << "fL" << fL;
fs << "fR" << fR;

fs.release();
cout << "Write done" << endl;

```

```
waitKey(0);  
  
return 0;  
}
```

附錄 2-0 (car_warning_radar_system.cpp)

```
// car_warning_radar_system.cpp : Defines the entry point for the console
application.
//
#include "stdafx.h"
#include <iostream>
#include <sstream>
#include <time.h>
#include <stdio.h>
#include <stdlib.h>
#include <vector>
#include <string>
#include <algorithm>
#include <ctype.h>
#include <process.h>
#include "sys/timeb.h"
#include <Windows.h>
#include <MMSystem.h>

#include "cxcore.h"
#include "highgui.h"
#include "cv.h"
#include <opencv2/opencv.hpp>
#include <opencv2/core/core.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <opencv2/imgproc/imgproc.hpp>
#include <opencv2/calib3d/calib3d.hpp>
#include <opencv2/gpu/gpu.hpp>

#define CAPTURE
#define OPTICAL_AXIS_CALIBRATION
#define DEMO
#define CAL_NO 1500
#define IMG_RES_W 1280
#define IMG_RES_H 720
#define OBJECT_SIZE 60
```

```

#define ROI_X 400
#define ROI_Y 200
#define ROI_W 700
#define ROI_H 500
#define DETECT_AREA_X_OFFSET1 10
#define DETECT_AREA_X_OFFSET2 40

#define ALARM false

using namespace cv;
using namespace std;
using namespace cv::gpu;

String car_cascade_name = "c:/temp/training/cars3.xml";
String mot_cascade_name = "c:/temp/training/haarcascade_motor01121559.xml";

CascadeClassifier_GPU car_cascade_gpu;
CascadeClassifier_GPU mot_cascade_gpu;

volatile bool cameraL_enable_capture;
volatile bool cameraR_enable_capture;
volatile bool cameraL_capture_done;
volatile bool cameraR_capture_done;
volatile bool alarm_fired;

Mat cameraL_frame;
Mat cameraR_frame;
Mat cameraL_frame_grey;
Mat cameraR_frame_grey;

double fL, fR;
double tan_theta_R, tan_theta_L;
double B_MUL_F_constant;

Scalar color[10];
Rect msg_box[10];
Point msg_loc[10];

```

```

int    H, W;
int    danger_cnt = 0;
double prev_distance;

int     detect_types[300];

char *get_detail_currenttime()
{
    char    st1[100];
    char    st2[100];
    struct timeb detail_time;

    ftime(&detail_time);
    strcpy(st1, ctime(&detail_time.time));

    st1[7] = '-';
    st1[19] = '\\0';
    strcpy(st2, &st1[4]);

    sprintf(st1, "%s.%03d", st2, detail_time.millitm);

    return(&st1[0]);
}

int gpu_detectAndDisplay(Mat frame, Rect *cRect_obj)
{
    GpuMat  car_objects;
    GpuMat  mot_objects;
    GpuMat  features;
    Rect    cRect_detect;
    Rect    cRect_obj1[100];
    Rect    cRect_all_obj[300];
    Point2f detect_obj_center[100];
    int     detect_obj_types[300];
    int     detect_obj1_types[300];
    int     W1=frame.cols, H1=frame.rows;
    int     i;
    int     x_lower, x_upper;

```

```

int    object_cnt = 0;

double t = (double)getTickCount();

GpuMat gray_converted;
GpuMat gray_gpu(frame);
gpu::equalizeHist(gray_gpu, gray_converted);

int car_detect_num = car_cascade_gpu.detectMultiScale(gray_converted,
car_objects, Size(200, 200), Size(50, 50), 1.3, 2);
int mot_detect_num = mot_cascade_gpu.detectMultiScale(gray_converted,
mot_objects, Size(250, 250), Size(50, 50), 1.3, 4);
printf("gpu_detectAndDisplay: car_detect_num=%d\n", car_detect_num);
printf("gpu_detectAndDisplay: mot_detect_num=%d\n", mot_detect_num);

Mat obj_host;

car_objects.colRange(0, car_detect_num).download(obj_host);
Rect* cobjects = obj_host.ptr<Rect>();
for(i=0;i<car_detect_num;i++) {
    detect_obj_types[object_cnt] = 1;
    cRect_all_obj[object_cnt++] = cobjects[i];
}

mot_objects.colRange(0, mot_detect_num).download(obj_host);
cobjects = obj_host.ptr<Rect>();
for(i=0;i<mot_detect_num;i++) {
    detect_obj_types[object_cnt] = 2;
    cRect_all_obj[object_cnt++] = cobjects[i];
}

int detect_object_center_y;
int detect_object_center_x;

int object_cnt0 = 0;
for(i=0;i<object_cnt;i++) {
    cRect_detect = cRect_all_obj[i];
    detect_object_center_y = cRect_detect.y + cRect_detect.height/2;

```

```

        detect_object_center_x = cRect_detect.x + cRect_detect.width/2;
        x_lower=
(W1/2+DETECT_AREA_X_OFFSET1)*(1.0-((double)((double)detect_object_center_y/(d
ouble)H1)));
        x_upper=
(W1/2+DETECT_AREA_X_OFFSET2)*(1.0+((double)((double)detect_object_center_y/(d
ouble)H1)));
        if(detect_object_center_x >= x_lower && detect_object_center_x <= x_upper)
{
            cRect_obj1[object_cnt0] = cRect_detect;
            detect_obj1_types[object_cnt0] = detect_obj_types[i];
            detect_obj_center[object_cnt0].x = detect_object_center_x;
            detect_obj_center[object_cnt0].y = detect_object_center_y;
            object_cnt0++;
        }
    }

    if(object_cnt0 == 0) {
        t = ((double)getTickCount() - t)/getTickFrequency();
        cout << "Harr: Times passed in seconds: " << t << endl;
        return 0;
    }

    int object_cnt1;

    cRect_obj[0] = cRect_obj1[0];
    detect_types[0] = detect_obj1_types[0];
    object_cnt1 = 1;
    if(object_cnt0 == 1) {
        t = ((double)getTickCount() - t)/getTickFrequency();
        cout << "Harr: Times passed in seconds: " << t << endl;
        return(object_cnt1);
    }

    // check whether is same object?
    bool same_object;
    int k;
    for(i=1;i<object_cnt0;i++) {

```

```

        same_object = false;
        for(k=0; k<object_cnt1; k++) {

            if(abs(detect_obj_center[i].x-(cRect_obj1[k].x+cRect_obj1[k].width/2)) <
cRect_obj1[k].width &&

abs(detect_obj_center[i].y-(cRect_obj1[k].y+cRect_obj1[k].height/2)) <
cRect_obj1[k].height ) {
                same_object = true;
                break;
            }
        }
        if(same_object == false) {
            cRect_obj[object_cnt1] = cRect_obj1[i];
            detect_types[object_cnt1] = detect_obj1_types[i];
            object_cnt1++;
        }
    }

    t = ((double)getTickCount() - t)/getTickFrequency();
    cout << "Times passed in seconds: " << t << endl;

    return(object_cnt1);
}

void capture_camera_L(void *dummy)
{
    VideoCapture capL(2);
    capL.set(CV_CAP_PROP_FRAME_WIDTH, IMG_RES_W);
    capL.set(CV_CAP_PROP_FRAME_HEIGHT, IMG_RES_H);

    while(1) {
        if(cameraL_enable_capture) {
            cameraL_capture_done = false;
            capL.retrieve(cameraL_frame); // read a new frame from video
            cvtColor(cameraL_frame, cameraL_frame_grey, CV_BGR2GRAY);
            cameraL_capture_done = true;
            cameraL_enable_capture = false;
        }
    }
}

```

```

    }
}

void capture_camera_R(void *dummy)
{
    VideoCapture capR(1);
    capR.set(CV_CAP_PROP_FRAME_WIDTH, IMG_RES_W);
    capR.set(CV_CAP_PROP_FRAME_HEIGHT, IMG_RES_H);

    while(1) {
        if(cameraR_enable_capture) {
            cameraR_capture_done = false;
            capR.retrieve(cameraR_frame); // read a new frame from video

            cvtColor(cameraR_frame, cameraR_frame_grey, CV_BGR2GRAY);
            cameraR_capture_done = true;
            cameraR_enable_capture = false;
        }
    }
}

void play_alarm(void *dummy)
{
    while(1) {
        if(alarm_fired && ALARM) {
            alarm_fired = false;
            PlaySound(TEXT("alarm.wav"), NULL, SND_SYNC);
        }
    }
}

void init()
{
    char filename[100];

    cvNamedWindow("WebcamL",1);

```

```

cameraL_enable_capture = false;
cameraR_enable_capture = false;
cameraL_capture_done   = false;
cameraR_capture_done   = false;
alarm_fired            = false;

int gpuCnt = getCudaEnabledDeviceCount(); // gpuCnt >0 if CUDA device
detected
if(gpuCnt==0) exit(-1);
if(!car_cascade_gpu.load(car_cascade_name)) exit(-1);
if(!mot_cascade_gpu.load(mot_cascade_name)) exit(-1);

int base = 9;

FileStorage fs;
sprintf(filename, "C:/Temp/CAL_file/calibration_file_%d_%d_demo.xml",
CAL_NO, IMG_RES_W);
fs.open(filename, FileStorage::READ);

fs["tan_theta_xR"] >> tan_theta_R;
fs["tan_theta_xL"] >> tan_theta_L;
fs["fL"] >> fL;
fs["fR"] >> fR;
B_MUL_F_constant = ((fL + fR)/2) * base;

H = IMG_RES_H;
W = IMG_RES_W;

color[0] = Scalar(0 , 255, 255);
color[1] = Scalar(255, 0 , 255);
color[2] = Scalar(255, 255, 0 );
color[3] = Scalar(0 , 0 , 255);
color[4] = Scalar(0 , 255, 0 );
color[5] = Scalar(255, 0 , 0 );
color[6] = Scalar(255, 255, 255);

int i;
for(i=0; i<6; i++) {

```

```

        msg_box[i].x      = W/2 - 150;
        msg_box[i].y      = i*120 + 50;
        msg_box[i].width  = 300;
        msg_box[i].height = 100;
        msg_loc[i].x      = msg_box[i].x + 20;
        msg_loc[i].y      = msg_box[i].y + 60;
    }

    prev_distance = 0.0;
    danger_cnt = 0;
}

double calc_distance(Mat frame, Mat obj_img, Rect frame_rect, Rect obj_rect,
int *disparity)
{
    Mat result;
    int match_method = CV_TM_CCORR_NORMED;
    Rect rect_found = obj_rect;

    /// Localizing the best match with minMaxLoc
    double minVal; double maxVal; Point minLoc; Point maxLoc;
    Point matchLoc;

    double t = (double)getTickCount();

    matchTemplate( frame, obj_img, result, match_method );
    normalize( result, result, 0, 1, NORM_MINMAX, -1, Mat() );
    minMaxLoc( result, &minVal, &maxVal, &minLoc, &maxLoc, Mat() );

    matchLoc = maxLoc;

    rect_found.x = matchLoc.x;
    rect_found.y = matchLoc.y;

    double xR_test = matchLoc.x + frame_rect.x;
    double xL_test = obj_rect.x;
    double x2R_test = ROI_X + xR_test - W/2;
    double x2L_test = ROI_X + xL_test - W/2;
    double x1R_test = fR * ((x2R_test + fR * tan_theta_R) / (fR - x2R_test *

```

```

tan_theta_R));
    double x1L_test = fL * ((x2L_test + fL * tan_theta_L) / (fL - x2L_test *
tan_theta_L));

    double x1_test = x1L_test - x1R_test;
    if(x1_test < 8) {
        return -1;
    }

    double distance = 14.0*B_MUL_F_constant*0.01 / x1_test;

    *disparity= (int)x1_test;

    t = ((double)getTickCount() - t)/getTickFrequency();
    cout << "TemplateMatch: Times passed in seconds: " << t << endl;

    return distance;
}

int _tmain(int argc, _TCHAR* argv[])
{
    Mat frameL_grey, frameR_grey;
    Mat frameL, frameR;
    Mat frameL1_grey, frameR1_grey;
    Mat frameL1, frameR1;
    Mat L, R;
    Rect cRectL_all[100];
    Rect Roi;
    Point matchLoc;
    int detect_num;
    int k;
    char timestr[100];
    bool show_lane = true;
    int idx;
    long image_seq_no = 0;
    int danger_start = 0;

    init();

```

```

_beginthread(capture_camera_L, 0, (void *)NULL);
_beginthread(capture_camera_R, 0, (void *)NULL);
_beginthread(play_alarm, 0, (void *)NULL);

// first initial to enable camera for image capturing
cameraL_enable_capture = true;
cameraR_enable_capture = true;

// main threading
for(;;) {

    // wait for image ready
    while(!cameraL_capture_done || !cameraR_capture_done) {
        if(waitKey(1) == 27) break;
    }

    image_seq_no++;
    // copy image from threading
    frameL = cameraL_frame.clone();
    frameR = cameraR_frame.clone();
    frameL_grey = cameraL_frame_grey.clone();
    frameR_grey = cameraR_frame_grey.clone();
    cameraL_enable_capture = true;
    cameraR_enable_capture = true;

    Roi = Rect(ROI_X, ROI_Y, ROI_W, ROI_H);

    frameL1 = frameL(Roi).clone();
    cvtColor(frameL1, frameL1_grey, CV_BGR2GRAY);

    if(show_lane) {
        rectangle(frameL, Roi, Scalar(255,255,0), 2, 8, 0);
        line(frameL, Point(ROI_X, ROI_Y+ROI_H),
Point(ROI_X+ROI_W/2+DETECT_AREA_X_OFFSET1, ROI_Y), Scalar(255,0,255), 2, 8, 0);
        line(frameL, Point(ROI_X+ROI_W, ROI_Y+ROI_H),
Point(ROI_X+ROI_W/2+DETECT_AREA_X_OFFSET2, ROI_Y), Scalar(255,0,255), 2, 8, 0);
        imshow("WebcamL", frameL);
    }
}

```

```

}

detect_num = gpu_detectAndDisplay(frameL1_grey, &cRectL_all[0]);
printf("%d", detect_num);

idx = 0;
for(k=0; k<detect_num; k++) {
    Rect  cRectR1;
    Rect  cRectL1;
    Rect  cRectR2;
    Rect  cRectL2;

    cRectL1 = cRectL_all[k];
    if(cRectL1.width > OBJECT_SIZE && cRectL1.height > OBJECT_SIZE) {
        cRectL2.x = cRectL1.x + (cRectL1.width - OBJECT_SIZE)/2;
        cRectL2.y = cRectL1.y + (cRectL1.height - OBJECT_SIZE)/2;
        cRectL2.width = OBJECT_SIZE;
        cRectL2.height = OBJECT_SIZE;
    }
    else {
        cRectL2 = cRectL1;
    }
    L      = frameL1_grey(cRectL2).clone();

    frameR1 = frameR(Roi).clone();
    cvtColor(frameR1, frameR1_grey, CV_BGR2GRAY);
    cRectR1.x = cRectL2.x - 4*cRectL2.width; if(cRectR1.x < 0) cRectR1.x
= 0;
    cRectR1.y = cRectL2.y - cRectL2.height; if(cRectR1.y < 0) cRectR1.y
= 0;
    cRectR1.width = cRectL2.width * 4.0; if(cRectR1.width > ROI_W)
cRectR1.width = ROI_W;
    cRectR1.height = cRectL2.height * 3.0; if(cRectR1.height > ROI_H)
cRectR1.height = ROI_H;
    if(cRectR1.y+cRectR1.height > ROI_H) cRectR1.y = cRectR1.y
-(cRectR1.y+cRectR1.height- ROI_H);
    if(cRectR1.x+cRectR1.width > ROI_W) cRectR1.x = cRectR1.x
-(cRectR1.x+cRectR1.width- ROI_W);

```

```

R                = frameR1_grey(cRectR1).clone();
int    disparity;
double distance = calc_distance(R, L, cRectR1, cRectL2, &disparity);

if(distance > 0) {
    if(distance < prev_distance) {
        if(image_seq_no-danger_start > 20) danger_cnt = 0; else
danger_cnt++;
        danger_start = image_seq_no;
    }
    else {
        danger_cnt = 0;
    }
    bool had_fired = false;
    if(prev_distance-distance > 2.0) {alarm_fired = true; had_fired
= true;}
    if(distance <= 10 && had_fired == false) {alarm_fired = true;
had_fired = true;}
    if(danger_cnt > 3 && had_fired == false) alarm_fired = true;
    prev_distance = distance;

    cRectL1.x = cRectL1.x + ROI_X;
    cRectL1.y = cRectL1.y + ROI_Y;

    if(had_fired) rectangle(frameL, cRectL1, Scalar(0, 0, 255), 4,
8, 0);
    else
        rectangle(frameL, cRectL1, color[idx], 2, 8,
0);

    imshow("WebcamL", frameL);

    char dis_str[100];
    sprintf(dis_str, " %5.2f m", distance);
    if(had_fired) rectangle(frameL, msg_box[idx], Scalar(0, 0, 255),
4, 8, 0);
    else
        rectangle(frameL, msg_box[idx], color[idx], 2,
8, 0);

    putText(frameL, dis_str, msg_loc[idx] ,
FONT_HERSHEY_COMPLEX_SMALL, 2.0, color[idx], 2, CV_AA);

```

```

        imshow("WebcamL", frameL);

        Point msg_loc1;
        msg_loc1.x = msg_box[idx].x + msg_box[idx].width + 20;
        msg_loc1.y = msg_box[idx].y + 60;
        if(detect_types[k] == 2) putText(frameL, "WATCHOUT !!!",
msg_loc1 , FONT_HERSHEY_COMPLEX_SMALL, 2.0, Scalar(0, 0, 255), 2, CV_AA);
        imshow("WebcamL", frameL);
        idx++;
    }
} // for detect_num

imshow("WebcamL", frameL);
strcpy(timestr, get_detail_currenttime()); printf("time=%s\n",
timestr);
waitKey(10);
} //for loop

return 0;
}

```

評語

1. 研究主題具實用性，且為重要的、持續發展中的主題。
2. 研究完整性高，實驗數據能用以佐證研究結論。
3. 建議可加強影像處理方法的深度以提昇辨識效能。亦可適度的放寬研究限制，使研究問題能貼近一般行車的狀況。