國立交通大學

資訊科學與工程研究所

碩 士 論 文

光影變化環境中的背景切割技術

Background Segmentation in dynamic lighting environment

研 究 生：聶政邦

指導教授：傅心家　教授

中 華 民 國 九 十 五 年 七 月

光影變化環境中的背景切割技術
Background Segmentation in dynamic lighting environment

研 究 生：聶政邦　　Student：Jang-Bang Nie

指導教授：傅心家　　Advisor：Hsin-Chia Fu

國 立 交 通 大 學

資 訊 科 學 與 工 程 研 究 所

碩 士 論 文

A Thesis

Submitted to Institute of Computer Science and Engineering

College of Computer Science

National Chiao Tung University

in partial Fulfillment of the Requirements

for the Degree of

Master

in

Computer Science

Juny 2006

Hsinchu, Taiwan, Republic of China

中華民國九十五年七月

光影變化環境中的背景切割技術

學生: 聶政邦　　指導教授: 傅心家 教授

國立交通大學資訊科學與工程研究所

摘　　要

影像背景切割一直爲分析動態影像不可或缺的技術。爲了得到正確的背景切割結果, 目前已有數種背景模型可用來描述每個像素的色彩分佈。然而, 處於亮光或陰影中的背景像素常常會被誤判爲前景。本文提出一個新的背景切割技術, 此方法先利用背景模型得到一個初步的背景切割結果, 其中被誤判爲前景的背景像素會形成數個區塊, 這些區塊的邊緣沒有明顯的色彩變化。因此我們提出一個修正演算法, 使用相鄰像素的資訊將這些錯誤的區塊改正爲背景。實驗結果顯示我們的方法可以讓背景切割正確率提升 $3.5\% \sim 9\%$ 左右, 此結果說明了修正演算法能有效改善亮光與陰影處的背景切割錯誤, 當拍攝環境的明暗度會在瞬間發生變化, 或是光影複雜交錯時, 也能夠提供可靠的背景切割結果。

Background Segmentation in dynamic lighting environment

Student:Jang-Bang Nie Advisors:Prof. Hsin-Chia Fu

Institute of Computer Science
National Chiao Tung University

Abstract

Background segmentation is an inevitable technique for active scene analysis. In order to obtain a correct background segmentation result. There are numerous background models proposed to model background color of each pixel. However, pixels on highlighted background and shadow region are usually segmented as foreground. In this paper, a new background segmentation technique is proposed. At first, the proposed method uses a background model to obtain a preliminary result, where some pixels incorrectly segmented as foreground form several regions without obvious color change on the edge. Then, considering the pixels neighbor relations, the proposed adjustment approach recovers those error foreground regions as background. Experimental result shows that our method can improve segmentation correctness about $3.5\% \sim 9\%$, which indicate that the proposed algorithm can effectively recover segmentation errors on highlighted background and shadow region and provides a robust background segmentation result with sudden illumination changes and complicated lighting condition.

誌　　謝

在此首先要感謝我的指導教授傅心家老師的指導與教誨, 這兩年的碩士生涯無疑是我人生中成長最多的時期, 很高興能有機會在 NNLAB 中學習研究的方法與論文撰寫的技巧, 並親身體驗從事研究工作的生活。另外也要感謝徐永煜學長、陳岳宏學長、曾政龍學長、賴柏伸學長在多方面的協助, 對於一個沒有寫過論文的人來說, 沒有老師與學長們的經驗傳授, 是不可能在如此短的時間內做出此篇論文與口試投影片的。

不論是何種領域, 能在自己的工作崗位上努力不懈的人, 無不令我深受感動並由衷敬佩。能在充滿未知的學術疆界貢獻一己綿薄的見解與構想, 便是身爲研究人員的無上光榮。無論從事研究工作的時間是一年、五年或是十年, 即使未來身在他方, 吾人永將謹守這份研究的熱誠與追求知識的渴望, 並在此勉勵後進學弟玉善, 以及在未來加入實驗室的成員。

在下的研究工作, 從一開始無知的摸索, 歷經匆忙的一年在此劃下句點, 希望此論文能對相關主題有興趣的人提供些許幫助, 對於論文中不夠成熟或完成度不足的地方, 吾人在此表示抱歉與遺憾。

最後, 僅將此論文獻給我親愛的家人, 我的父母與妹妹。

目錄

圖目錄

表目錄

符號目錄

Chapter 1

序論

1.1 動機

隨著文明進步, 人類的生活與科技越來越密不可分。基於人們對安全的需求, 在許多場合建構監視系統都有其必要性, 例如住家的防盜系統、道路上的車輛監控、車站等公共空間過往行人的密度統計等等。

傳統的監視系統只能被動的錄影, 無法依照畫面內容做出即時的反應, 或是需要有人隨時在畫面前監視。但在需要長期監控的地方使用人工監視過於費時費力, 所以自動化視覺監視系統 (visual surveillance system) 一直是電腦視覺 (computer vision) 領域中相當重要的研究課題。

爲了減少不相關的資訊與運算複雜度, 許多高階的視覺辨識工作如移動物體追蹤 (moving object tracking)、人臉辨識 (face recognition) 會先將監控的目標 (通常是移動中的人或車輛) 與背景切割開來, 而切割的好壞也直接影響到後續的辨識效果, 所以將攝影機拍攝畫面中的移動物體 (moving object) 擷取出來是最基本且重要的一個步驟。

1.2 目標

目前的背景切割技術, 使用的資料通常來自於很高的畫面擷取率 [1] 所取得的畫面, 因此相鄰的兩張畫面間隔時間很短, 相鄰的畫面內容具有高度相關性—換句話說, 他們大致上含有相同的移動物體, 而且位置只有少量偏移。由於許多監視系統的架構爲多個攝影機連接至一台電腦, 此電腦往往要同時處理多個畫面, 畫面擷取率過高電腦的運算能力將無法負荷。所以本論文所考慮的畫面擷取率爲十秒

[1] [1]採用11∼13 張/秒, [2]採用30 張/秒, [3]採用20 張/秒。

一張, 相鄰的拍攝畫面之間並無明顯的關連性。

在大部分實際的環境中, 背景都免不了會有明暗變化, 其產生的主要原因有兩種: (1) 背景處於陰影 (shadows) 中或是 (2) 受強光照射而特別亮 (highlighted)。背景物體在此情況下很容易被誤判爲前景物體。一個可靠的背景切割技術應該要能克服背景明暗變化的問題。本論文的目標便在於如何在具有高度光影變動的環境中, 做到良好的背景切割處理。

1.3 章節簡介

在第二章裡, 將會針對背景切割的相關技術作一些簡單的介紹; 第三章則是介紹本論文所使用的背景切割技術; 第四章會說明實驗的設計與實驗的結果和比較; 第五章爲總結與未來展望。

Chapter 2

相關研究

我們將分割背景與前景的工作稱爲背景切割(background segmentation)。在切割背景時, 首先要建構一個能描述背景色彩的模型, 之後將新畫面與背景模型比較, 將判斷爲背景的部分消除。本章將就目前背景色彩模型建立的研究作一概述。

2.1 單色背景模型 (Unimodal Background Model)

此類研究通常只考慮背景畫面爲靜態的狀況, 並且假定每個背景像素只有一種色彩。

由背景像素所組成之畫面稱爲參考畫面 (reference frame), 用於代表背景的色彩。在[4, 5, 6]的方法中只考慮輸入影像爲灰階影像, 測試時計算輸入影像的像素 $I_{x,y}$ 與參考畫面的像素 $B_{x,y}$ 的差

$$D_{x,y} = |I_{x,y} - B_{x,y}|$$

再對 $D_{x,y}$ 取一臨界值(threshold) 區分前景與背景。因此參考畫面直接影響背景切割的結果, 所以有必要建構一個具有代表性的參考畫面。

爲了取得具代表性的背景畫面, Rosin and Ellis [4]提出保留一段過去N 時間的輸入畫面 $I^t_{x,y}$, $t = 1 \sim N$, 取每個像素的中位數組成參考畫面:

$$B_{x,y} = \text{med}_t I^t_{x,y}$$

使用中位數的好處是比較不會受極端值 (比平均值大很多或小很多的樣本) 影響。Yang and Levine [5]提議使用最小方差中位數(LMedS) 做爲參考畫面:

$$B_{x,y} = \min_b \text{med}_t (I^t_{x,y} - b)^2$$

LMedS保有中位數的優點, 而且比中位數更接近樣本最集中的點。參考畫面的像素值越接近樣本集中點, 越能代表大部分時間下的背景色彩 (大部分樣本的像素值)。

除了建立具有代表性的參考畫面外,Rosin and Ellis [6]提出自動化取 $D_{x,y}$ 臨界值的方法, 定義MED 爲 $D_{x,y}$ 的中位數, 假設畫面中的移動物體像素不超過全部像素個數的一半, MAD可以用來代表 $D_{x,y}$ 受雜訊干擾的程度:

$$\text{MED} = \text{med}_{x,y} D_{x,y},$$

$$\text{MAD} = \text{med}_{x,y} |D_{x,y} - \text{MED}|.$$

則臨界值 $T = \text{MED} + 3 \times 1.4826 \times \text{MAD}$ 。

有別於使用參考畫面的作法, Wren et al. [7]使用單高斯模型(UniGaussian model) 來描述一個背景像素的 YUV 色彩, 每個高斯的平均值 (mean) 會根據輸入畫面作簡單的調適 (adaptation)。

Ridder et al. [8]使用卡曼濾波器(Kalman Filter) 估計背景像素色彩。假設每個像素的色彩是由一個隱藏狀態來決定, 此狀態爲一個向量, 會隨時間動態改變, 在任一狀態下所得的觀察會受到一個雜訊干擾, 卡曼濾波器 (Kalman Filter) 可以提供一個遞迴演算法計算下一個狀態的估計值, 而且估計狀態與眞正狀態的誤差 [1] 爲最小。由於卡曼濾波器可以追蹤背景色彩的動態變化, 此模型可容許背景色彩有漸進式的改變, 例如日光的強弱變化。

上述模型對於每個像素只能記住一種背景顏色, 並無法處理背景像素具有多種顏色的情形, 這種情況通常都發生在背景物體會移動的時候, 例如背景有樹枝隨風擺動, 附近的像素會在樹枝、樹葉、天空的顏色間快速切換。

2.2 多色背景模型 (Multimodal Background Model)

爲了考慮每個背景像素有多個顏色分佈。Stauffer and Grimson [1]提出使用高斯混合(mixture of Gaussions) 模型描述背景色彩的方法, 每個高斯成員 (Gaussion Component) 代表背景像素中的一種顏色分佈, 此模型也稱作 Time-Adaptive, Per Pixel, Mixture of Gaussions(TAPPMOG)[3]。

[1]誤差定義爲兩個狀態向量的幾何距離(Euclidean distance)。

模型中的參數是經由online K-mean 訓練而得: 每個新進像素先和現有的高斯成員作配對, 找到符合的高斯成員再依據新資料調整參數, 若找不到則依據新資料建立一個新的高斯成員。在測試時模型的參數也會隨輸入作調適 (adaptation), 然而調適時的學習率 (learning rate) 設定太低會跟不上背景色彩的快速變化; 學習率高則會有將參數往前景色彩調整的風險。

相對於高斯混合模型, Elgammal et al. [2]使用無參數模型(non-parametric model) 來描述背景色彩: 每個位置記錄一組樣本點, 套用 normal function 估計樣本的密度。他們使用兩組模型:

1. short-term model 只取最近的畫面做爲樣本, 用來描述背景色彩的快速變化;

2. long-term model 則是在較長的時段中取樣本, 用來對背景色彩作比較保守的估計。

最後再將兩個模型的結果整合。此方法的缺點在於紀錄樣本點需要佔用大量記憶體, 而且樣本越多, 估計密度的計算時間也越長。

爲了改良前二者,Kim et al. [9]使用codebook model 來描述背景色彩, 每種顏色分佈除了使用一個向量作代表, 還記錄一些額外資訊如亮度和訓練過程的時間特徵, 此模型可以很貼切的描述背景色彩因光線變化所形成的分佈狀況, 而且即使訓練畫面中含有大量前景影像, 並不會影響正確背景資訊的取得。

2.3 背景調適 (Background Adaptation)

背景調適已成爲背景模型不可或缺的一環, 上一節的三種方法都有做背景調適, 因爲在實際的監視環境中, 完全靜態的背景幾乎是不存在的。然而背景調適仍難以解決背景色彩快速變化的問題, 因爲背景調適若要用來調整短時間內的背景色彩波動, 前後兩張畫面必須爲連續畫面, 若是輸入畫面的時間前後間距較長, 背景調適只能用來調整背景畫面長時間緩慢的變化。

2.4 其他的背景切割技術

2.1與2.2中介紹的方法都是將畫面中的每個像素視爲獨立的個體, 所以每個像素個別使用一個模型來描述, 所以又被稱爲 pixel-level 背景模型。Cristani et al. [3]提出整合TAPPMOG 與 region-

level 資訊的技術: 先將背景畫面依照材質相似性切割成多個區域, 前後畫面變化量大的區域, 其涵蓋的像素模型使用較高的學習率 (learning rate) 調整參數。雖然這樣可透過 region-level 資訊加強背景調適的效果, 但在非連續畫面中並不適用。

Javed et al. [10]以高斯混合模型爲基礎, 加入梯度(gradient) 特徵作爲背景切割的一個依據。其方法是先利用高斯混合模型作背景切割, 將得到的結果中前景的部分依照連通性 (connectivity) 集合成數個區塊, 之後針對每個前景區塊作進一步判斷。前景的判斷是依照兩個條件:(1) 一個前景物體的邊緣梯度應該會和背景梯度 [2] 有很大的差異; (2) 前景物體的邊緣在原圖中會有較強的梯度。不符合這兩項條件的前景區塊會被改判爲背景。此方法可以處理背景局部區域明暗突然發生改變, 然而並無法解決陰影的問題, 主要原因是陰影與前景物體通常會連在一起, 所以一個前景區塊常常會同時包含前景與陰影, 以前景區塊作爲切割單位, 是無法將兩者分開的。

[2]其文章中有說明如何使用高斯混合模型[1]計算背景的梯度, 所以在不修改背景模型的情況下也能得到背景的梯度資訊。

Chapter 3

背景切割技術

本論文的背景切割工作可分成兩個步驟,(1) 底層的背景模型提供一個初步的背景切割結果, 這個部分我們使用[9]提出的codebook model, (2) 之後使用修正演算法將錯誤的前景 (突然變亮的背景或投射在背景上的影子) 移除。

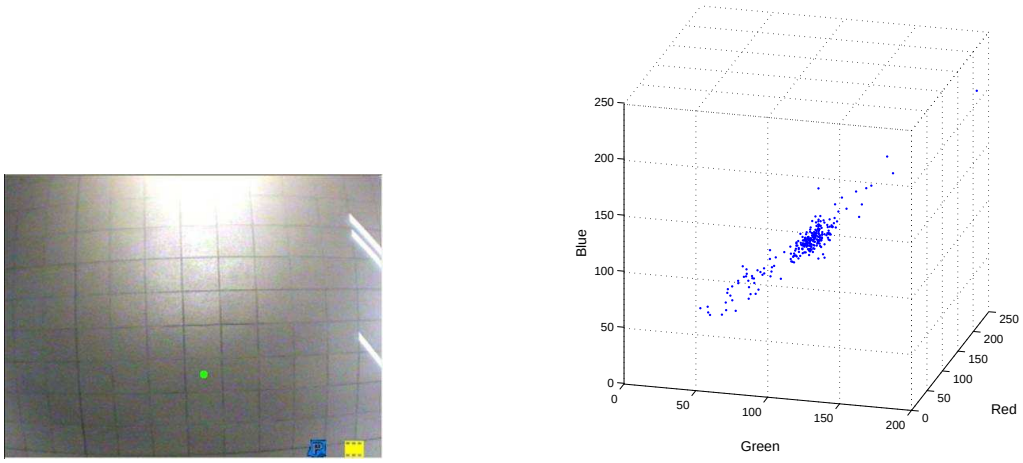
3.1 Codebook Background Model

Codebook一詞源於 Vector Quantization(VQ)。在原來的 VQ 定義中, 考慮一個由向量構成的樣本集合, 將彼此比較接近的樣本歸類爲一群 (cluster), 如此可將所有樣本分爲數個群, 每個群用一個向量代表它, 此向量就被稱爲 codeword, 而這些 codeword 的集合稱爲 codebook。在一張影像中, 每個像素的背景色彩可能是由好幾個群所構成,codebook model 針對每個像素, 個別使用一個 codebook 來描述背景色彩。

3.1.1 Codeword定義

在[9]所設計的codeword 是用來描述一個背景色彩因雜訊或明暗變化造成的分佈範圍。首先觀察某個背景像素的色彩隨著明暗改變的顏色分佈 (圖 3.1), 其近似一個軸線通過原點的的柱狀範圍。圓柱的長度代表背景物體明暗變化的程度, 圓柱的半徑代表背景物體受雜訊干擾的偏移範圍。假設某個像素可從訓練圖檔中取得 N 個樣本[1] $\mathcal{X} = \{x_1, x_2, \ldots, x_N\}$, 描述此像素的 codebook 爲 $\mathcal{C} = \{c_1, c_2, \ldots, c_L\}$, 其中含有 L 個 codeword。一個 codeword c_i 包含一個向量 $v_i = (\bar{R}_i, \bar{G}_i, \bar{B}_i)$ 與六個參數 $aux_i = \langle \check{I}_i, \hat{I}_i, f_i, \lambda_i, p_i, q_i \rangle$, 代表的意義如下:

[1]每個樣本是位於RGB 色彩空間上的顏色向量。



(a) (b)

圖 3.1: (a) 被採樣的像素位置 (綠色圓點); (b) 收集 268 個樣本的色彩分佈圖

v_i: 屬於 c_i 的樣本的平均值 (Mean), 代表此色彩分佈的中心點。

$\hat{I}_i, \check{I}_i$: 在 $\mathcal{X}$ 裡屬於 c_i 的樣本中所能找到的最大與最小的亮度。

f_i: 屬於 c_i 的樣本個數。

λ_i: 在訓練過程中, 沒有像素指定給 c_i 的最長時間間隔。

p_i, q_i: 在訓練過程中, 第一個與最後一個像素指定給 c_i 的時間。

判斷像素 $x_t = (r_t, g_t, b_t)$ 的亮度 $I = \|x_t\| = \sqrt{r_t^2 + g_t^2 + b_t^2}$ 是否落在 c_i 容許範圍的條件爲

$$brightness(I, \langle \check{I}_i, \hat{I}_i \rangle) = \begin{cases} \text{true} & \text{if } I_{low} \leq I \leq I_{hi}, \\ \text{false} & \text{Otherwise.} \end{cases}$$

$[\check{I}_i, \hat{I}_i]$ 是 c_i 現有樣本的亮度範圍, 但在訓練過程中, c_i 的亮度範圍必須能向外擴張, 所以實際上 c_i 容許更大亮度範圍 $[I_{low}, I_{hi}]$ 的資料加入, 上下界的定義爲

$$I_{low} = \alpha \check{I}, I_{hi} = \min\left\{\hat{I}/\alpha, \beta \hat{I}\right\}. \tag{3.1}$$

常數 α 是一個小於 1 的值, α 越小, 亮度範圍擴張的速度越快; β 則是一個大於 1 的常數, 用來進一步限制亮度上界的擴張速度, 因爲背景變暗的情況 (陰影) 通常比變亮的情況多。

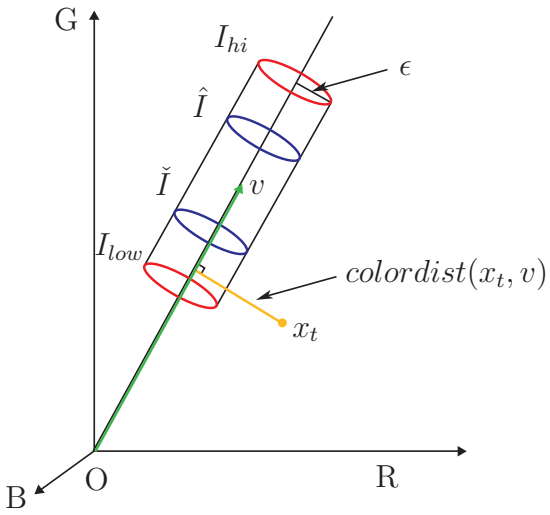


圖 3.2: codeword 代表的範圍

一個像素 x_t 與 codeword c_i(其樣本平均值爲 v_i) 的色彩差異計算方式爲

$$colordist(x_t, v_i) = \sqrt{\|x_t\|^2 - (\frac{x_t \cdot v_i}{\|v_i\|})^2} \tag{3.2}$$

即圖3.2中資料點 x_t 與圓柱軸心的垂直距離。實際上, $colordist(x_t, v_i)$ 是用來衡量兩種色彩在色相 (hue) 與飽和度 (saturation) 的差異。所以 codeword 範圍由以下兩個條件決定:

$$colordist(x_t, v_i) \leq \epsilon_1 \tag{3.3}$$

$$brightness(I, \langle \check{I}_i, \hat{I}_i \rangle) = \text{true} \tag{3.4}$$

所定義的柱狀範圍如圖3.2所示, $[I_{low}, I_{hi}]$ (圓柱的兩端) 代表 codeword 容許物體色彩變亮或變暗的程度, ϵ_1 (圓柱的半徑) 代表 codeword 容許物體色彩因爲雜訊造成偏差的最大值, 在雜訊較強的攝影環境, ϵ_1 需要設定得比較大。

3.1.2 建構 Codebook

假設訓練影像有 N 張, 一開始所有的 codebook 皆爲空集合, 之後逐一讀入訓練影像, 在輸入第 t 張影像時, 其中每個像素 $x_t = (r_t, g_t, b_t), I = \|x_t\|$ 會輸入到相對應的 codebook 中, 並在裡面尋找第一個與新進像素配對成功 (符合公式 (3.3) 與 (3.4)) 的 codeword, 之後分爲以下兩種情況處理:

1. 找到符合的 codeword c_m , 其參數爲

   $v_m = (\bar{R}_m, \bar{G}_m, \bar{B}_m)$ 與 $aux_m = \langle \check{I}_m, \hat{I}_m, f_m, \lambda_m, p_m, q_m \rangle$ 。依照以下的方式更新參數:

   - $v_m \leftarrow \left(\frac{f_m \bar{R}_m + R}{f_m + 1}, \frac{f_m \bar{G}_m + G}{f_m + 1}, \frac{f_m \bar{B}_m + B}{f_m + 1} \right)$
   - $aux_m \leftarrow \langle \min\{I, \check{I}_m\}, \max\{I, \hat{I}_m\}, f_m + 1, \max\{\lambda_m, t - q_m\}, p_m, t \rangle$

2. 沒有找到或 codebook 爲空集合。建立一個新的 codeword c_L , 參數的初始化如下:

   - $v_L \leftarrow (R, G, B)$
   - $aux_L \leftarrow \langle I, I, t - 1, t, t \rangle$

3.1.3 過濾前景資訊

codebook建構完成後, 每個 codebook 同時包含描述前景色彩與描述背景色彩的 codeword, 那些描述前景色彩的 codeword 必須從模型中移除。Kim et al. [9]假設屬於背景的移動物體, 其運動方式通常是週期較短的來回運動(quasi-periodical), [2] 所以在過濾前景時對 λ 取一臨界值 T_λ, 所有 $\lambda_m > T_\lambda$ 的 codeword c_m會被移除。然而當訓練影像中有前景物體在畫面中靜止較長的一段時間時, 被遮蔽的背景像素所屬 codeword, λ 也會跟著增加, 因此有可能因爲 λ 因爲而被移除。取而代之, 我們認爲前景 codeword 樣本數應該比背景少, 所以我們過濾前景的方法是對 f 取一臨界值 T_f , 將所有 $f_m < T_f$ 的codeword c_m 移除。此外, 我們將一個 codebook 中的 codeword 依照 f 由大到小排序, 可以發現 f 最大的 codeword 代表大部分時間時的背景顏色, 其次的 codeword 則描述被陰影覆蓋的背景色彩或是亮度較高的背景色彩, 所以我們可以藉由改變臨界值 T_f 的大小來調整codebook model 容忍背景明暗變化的程度。

3.1.4 背景切割

當過濾掉前景的 codeword 後, 就可以使用只剩背景 codeword 的模型作背景切割測試。一張測試影像中的每個像素 x_{ij} 會被輸入到相對應的 codebook 中測試, 偵測的結果以 C_{ij} 表示, 演算法如

[2]例如樹枝或樹葉的擺動就是典型的例子。

下:

背景切割演算法

I. $x_{ij} = (r_{ij}, g_{ij}, b_{ij}), I \leftarrow \sqrt{r_{ij}^2 + g_{ij}^2 + b_{ij}^2}$

II. 在 codebook 中尋找符合以下兩條件的 codeword c_m

$$colordist(x_{ij}, v_m) \leq \epsilon_2 \tag{3.5}$$

$$brightness(I, \langle \check{I}_m, \hat{I}_m \rangle) = \text{true} \tag{3.6}$$

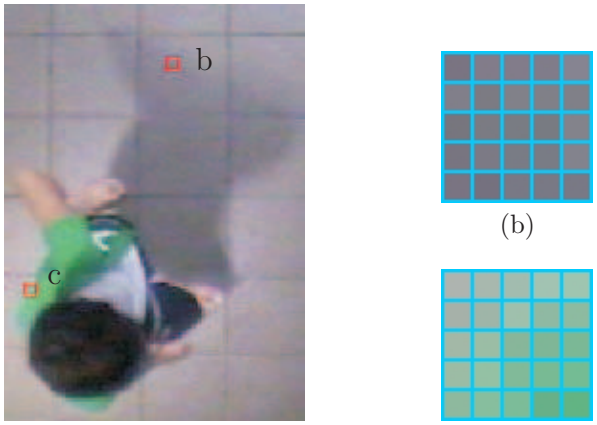
III.

$$C_{ij} = \begin{cases} \text{“BG”} & \text{找到同時符合判斷式(3.5) 與 (3.6) 的 codeword, } x_{ij}\text{是背景。} \\ \text{“FL”} & \text{只有找到符合判斷式(3.5) 的 codeword, } x_{ij}\text{爲前景。} \\ \text{“FC”} & \text{Otherwise, } x_{ij}\text{爲前景。} \end{cases} \tag{3.7}$$

此演算法切割出來的背景, 由於影子與亮光的影響, 所以不夠精確。下節將介紹如何改進背景切割的正確性。

3.2 背景修正演算法

當一張畫面有陰影投射在背景上時, 會有一些背景像素的亮度與平均值差不多, 有些卻低於平均值很多。因此背景模型若要能正確將陰影部分辨識爲背景, 必須要將較暗的背景色彩也納入模型中。但如此會造成深色前景被誤判爲背景的機率 (前景錯誤率) 增加, 這也是背景模型在處理陰影所面對的難題。事實上, 此問題的根源是因爲背景模型只描述了每個像素的色彩分佈, 這些資訊並不足以分辨陰



(a) (b) (c)

圖 3.3: 邊緣放大圖。(a) 同時有前景與影子的影像; (b) 影子的邊緣; (c) 前景 (人的衣服) 的邊緣。

影與深色背景, 若要正確辨識陰影並同時減少前景錯誤率, 引入其他的資訊是必要的。本修正演算法使用了 region-level 的資訊解決上述的問題。

3.2.1 原理

如圖3.3所示, 一個前景物體的邊緣處會有明顯的色彩變化 (圖3.3c), 而影子會有一些邊緣色彩變化非常緩慢 (圖3.3b)。欲對 codebook model 輸出的前景作修正, 可以從前景邊緣往內逐步檢查, 我們將正在檢查的像素稱爲"生長點"; 在一個以生長點爲中心的固定範圍中, 與生長點色彩相近的像素稱爲"參考點"。若此時生長點是在眞正的前景邊緣, 由於背景色彩與前景有差異, 參考點中只會有前景的像素; 相對來說, 若此時生長點在影子的邊緣, 參考點會含有比較多的背景像素。

3.2.2 修正演算法

首先假設 x_{ij} 爲原影像在座標 (i, j) 的像素值, codebook model 的分割結果爲 C_{ij} (定義請見公式 (3.7))。則我們定義8-neighbors 爲與 x_{ij} 相鄰的八個像素:

$$N_8(x_{ij}) = \{x_{pq} | (p, q) \neq (i, j) \text{ and } |p - i| \leq 1 \text{ and } |q - j| \leq 1\}$$

參考視窗是以 x_{ij} 爲中心, $(2r_s + 1) \times (2r_s + 1)$ 的方形區域:

$$W(i, j) = \{x_{pq} | (p, q) \neq (i, j) \text{ and } |p - i| \leq r_s \text{ and } |q - j| \leq r_s\}, \tag{3.8}$$

r_s is spatial radius, 參考視窗的空間半徑。

(a) (b) (c)

圖 3.4: $r_s = 2, r_r = 20$ 時, 參考視窗與參考點範例。(a) 參考視窗在圖中的位置 (綠色方格); (b) 參考視窗放大圖; (c) 參考點 (黃色部分)。

參考點爲參考視窗中, 與中心點色彩相近的像素:

$$R(i,j) = \{x_{pq} | x_{pq} \in W(i,j), \|x_{pq} - x_{ij}\| < r_r\}, \tag{3.9}$$

r_r is range radius, 即色彩差異的允許範圍。

圖 3.4 爲參考視窗與參考點的範例。參考點的數量以 $Size(i,j) = |R(i,j)|$ 表示, 參考點中屬於背景的像素個數爲

$$Bctr(i,j) = |\{x_{pq} | x_{pq} \in R(i,j) \text{ and } C_{ij} = \text{“BG”}\}|$$

在演算法的執行過程中, 影像的每個像素 x_{ij} 會被賦予一個狀態

$$State(i,j) \in \{\text{“未處理”}, \text{“生長點”}, \text{“潛在生長點”}, \text{“非生長點”}\}$$

四種狀態的意義如下:

1. “**未處理**” 演算法尚未處理過的像素。
2. “**生長點**” 演算法目前要處理的像素。
3. “**潛在生長點**” 曾經是生長點, 但還不確定是否要修正, 仍有可能在未來成爲生長點。
4. “**非生長點**” 已經修正過或確定不需要修正的像素, 未來不會再成爲生長點。

一個像素 x_{ij} 要能成爲生長點, 必須符合以下條件:

$$C_{ij} = \text{“FL”} \tag{3.10}$$

$$\|x_{ij}\| \geq I_e \tag{3.11}$$

由於背景被影子投射或照到強光時, 只會造成亮度偏離背景範圍。判斷式 (3.11) 是用來篩選出符合判斷式 (3.5) (與背景色彩的色相與飽和度相近) 但不符合判斷式 (3.6)(與背景色彩的亮度差距較大), 而被判定爲前景的像素, 這類像素事實上有可能爲背景。

修正演算法假設前景物體的邊緣有明顯的色彩變化。當光源被嚴重遮蔽時, 拍攝到的影像中可能會出現亮度很低的區域 (如圖 3.5), 在這樣的區域中各種物體的色彩資訊都會嚴重流失, 背景與前景的交界不會有明顯的色彩變化, 所以使用修正演算法的前提是生長點的亮度必須大於一個預設值 (符合判斷式 (3.11))。

圖 3.5: 光源被嚴重遮蔽的範例影像

所有生長點置於一個佇列 Q 中。決定一個生長點 x_{ij} 要被改判爲背景, 必須符合以下兩條件:

$$Size(i,j) > T_r,\ T_r \text{ is a threshold.} \tag{3.12}$$

$$Bctr(i,j) \geq \kappa\, Size(i,j) \tag{3.13}$$

生長點的修正與否是由參考點決定, 所以參考點的數量必須夠多 (符合判斷式 (3.12)), 其決定才被認爲是可靠的; 使用判斷式 (3.13) 的的意義在於當參考點中屬於背景像素的比例高於一個預設值時, 我們才認爲此生長點應該修正爲背景。κ 是一個小於1的常數。

演算法分成兩個部分, 一是初始化, 二是演算法主體, 詳細內容如下。

Initialize()

For each $x_{ij} \in I$

If $C_{ij} =$ "FL" and $\|x_{ij}\| \geq I_e$

If $\exists\, x_{pq} \in N_8(x_{ij})$ s.t. $C_{pq} =$ "BG"

$State(i,j) \leftarrow$ "生長點"

$Q \leftarrow x_{ij}$

Else

$State(i,j) \leftarrow$ "未處理"

Else

$State(i,j) \leftarrow$ "非生長點"

背景修正演算法

I. Initialize()

II. While $Q \neq \emptyset$

1. $x_{ij} \leftarrow Q$
2. 檢查下列兩條件
   (a) $Size(i,j) > T_r$
   (b) $Bctr(i,j) \geq \kappa\, Size(i,j)$
3. If (a)(b) 都成立
   - $C_{ij} \leftarrow$ "BG"
   - $State(i,j) \leftarrow$ "非生長點"
   - Wake Up:
     For each $x_{pq} \in N_8(x_{ij})$ s.t. $State(p,q) =$ "未處理",
     $State(p,q) \leftarrow$ "生長點"
     $Q \leftarrow x_{pq}$
   - Update:
     For each $x_{pq} \in R(i,j)$ s.t. $State(p,q) =$ "潛在生長點",
     $Bctr(p,q) \leftarrow Bctr(p,q) + 1$
     If $Bctr(p,q) \geq \kappa\, Size(p,q)$
     $State(p,q) \leftarrow$ "生長點"
     $Q \leftarrow x_{pq}$
4. If (a) 成立,(b) 不成立, 且 $\exists\, x_{pq} \in R(i,j)$ s.t. $State(p,q) \neq$ "非生長點"
   $State(p,q) \leftarrow$ "潛在生長點"
5. Otherwise
   $State(p,q) \leftarrow$ "非生長點"

3.2.3 時間複雜度

背景修正演算法的時間複雜度計算如下。定義 N 爲影像的像素個數, N_f 爲影像中前景部分的像素個數, 初始化時每個像素只會處理一次, 每次花費時間爲常數, 所以初始化的時間爲 $\mathrm{O}(N)$ 。

只有前景像素能成爲生長點, 而一個像素最多只能成爲生長點兩次, 所以進入佇列 Q 的生長點最多只有 $\mathrm{O}(2N_f) = \mathrm{O}(N)$ 個。

每個生長點計算 $Size(i,j)$ 與 $Bctr(i,j)$的時間爲 $\mathrm{O}(r_s^2)$, update 需要 $\mathrm{O}(r_s^2)$, 其餘的動作只需要 $\mathrm{O}(1)$ 的時間, 所以一個生長點的處理時間爲 $\mathrm{O}(2r_s^2+1)=\mathrm{O}(r_s^2)$ 。

所以全部花費的時間爲 $\mathrm{O}(N+N\times r_s^2)=\mathrm{O}(Nr_s^2)$

Chapter 4

實驗

4.1 效能評估方法

背景切割程式可以被視爲一個分類器 (classifier), 其目的是要預測一張影像的各個像素爲前景或背景, 如果可以得知畫面中眞正的背景與前景爲哪些像素,[1] 則預測結果可分爲下列四種情況:

true positive(TP): 爲一張圖中被預測爲前景, 實際上也是前景的像素個數;

false positive(FP): 爲一張圖中被預測爲前景, 實際上卻是背景的像素個數;

flase negative(FN): 爲一張圖中被預測爲背景, 實際上卻是前景的像素個數;

true negative(TN): 爲一張圖中被預測爲背景, 實際上也是背景的像素個數。

這四類結果如表 4.1 所示

表 4.1: confusion matrix

		預測結果	
		前景	背景
眞實結果	前景	true positive	false negative
	背景	false positive	true negative

我們使用三個度量單位評估背景切割的好壞, 定義如下:

[1] 實際上我們是透過人工切割的結果得知。

- 背景錯誤率 (e_B), 即 FP 佔總畫面的比例, 此類錯誤主要發生在陰影與高亮度背景的部分, 其定義爲

$$e_B = \frac{\text{FP}}{\text{TP} + \text{FP} + \text{FN} + \text{TN}}$$

- 前景錯誤率 (e_F), 即 FN 佔總畫面的比例, 此類錯誤主要出現在前景顏色與背景很接近時, 其定義爲

$$e_F = \frac{\text{FN}}{\text{TP} + \text{FP} + \text{FN} + \text{TN}}$$

- 成功率 (success rate, $R_{success}$), 預測正確的像素 (TP+TN) 佔總畫面的比例, 定義爲

$$R_{success} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{FP} + \text{FN} + \text{TN}}$$

實驗的目的是爲了顯示使用修正演算法後, 背景切割的效果有何改善之處。所以測試時, 對照組只用 codebook model 切割背景; 實驗組使用 codebook model+ 修正演算法, 比較兩者在所有測試畫面中的平均成功率、平均背景錯誤率、最差背景錯誤率、平均前景錯誤率與最差前景錯誤率。

4.2 實驗環境

我們在一個地下鐵車站選定四個地點架設照度0.05Lux, 焦距1.4mm 的 CCD 攝影機, 每台攝影機都嵌在天花板中垂直向下拍攝, 鏡頭高度三公尺。另外設有一台電腦負責擷取四台攝影機拍下的畫面, 在車站的營運時段 (6:00AM $\sim$ 12:00PM) 以十秒一張的速度擷取畫面, 每張畫面的解析度爲 512×384 。

四台攝影機的拍攝畫面如圖 4.1。Camera 1 的拍攝地點在車站較偏僻處, 前景物體出現頻率較少, 因爲光源不足導致畫面偏暗; Camera 2 所在地的人潮最爲擁擠, 但也同樣有光源不足的情況; Camera 3 的光源最充足, 拍攝畫面色彩最眞實; Camera 4 畫面中有一個很強的點光源, 靠近光源中心點的物體會趨近白色。

(a) (b) (c) (d)

圖 4.1: 拍攝樣本畫面。(a)camera 1; (b)camera 2; (c)camera 3; (a)camera 4。

4.3 訓練資料與參數設定

四個攝影機隨機取1000張畫面 ($N = 1000$) 作爲訓練資料, 每個攝影機各自建構一個 codebook model , codebook model 需要人爲調整的參數爲

ϵ_1, ϵ_2: 前者爲建構 codebook 時 colordist 的最大臨界值, 用來決定判斷式 (3.3)。後者爲 codebook model 在測試時 colordist 的最大臨界值, 用來決定判斷式 (3.5)。此二參數與攝影機拍攝的雜訊強度有關, 通常設定範圍在10到20之間, 範圍內設定不同的值對成功率沒有顯著的影響, 然而 ϵ_1 若太小會造成 codeword 數量太多, 因而使記憶體使用量和運算花費過高。

α, β: 決定 $[I_{low}, I_{hi}]$ 的參數, 用來計算公式 (3.1), 這組參數與 T_f 一同決定 codebook model 對光影變化的容忍程度, 現在爲了統一由 T_f 決定此特性, 我們使用較大的 α 值, β則取接近 α 倒數的值。

四個攝影機的模型使用相同參數如表4.2。

表 4.2: codebook 參數

ϵ_1	ϵ_2	α	β
20	20	0.7	1.2

我們列出建構完成的 codebook model 中, 前三個 codeword 的樣本個數 (f) 平均值於表4.3

。

表 4.3: 前三個 codeword 的平均樣本數 $\bar{f}$

	camera 1	camera 2	camera 3	camera 4
first codeword	869.147	489.938	540.015	749.309
second codeword	65.7674	182.365	182.66	108.45
third codeword	20.5115	105.933	76.2844	39.7935

建構完的模型還需要過濾前景, 此時使用不同臨界值 T_f , 可得到多個對明暗變化容忍度不同的 codebook model。在實作上, T_f 是透過參數 b_f 決定, $T_f = N/b_f = 1000/b_f, b_f = 2, 4, 6, \ldots, 40$, 所以每個攝影機各可得20個不同 b_f 的模型。[2] 此時可順便觀察不同 b_f 時, 平均每個 codebook 含有的 codeword 個數 (n_c), 表4.4列舉 $b_f = 2, 4, \ldots, 40$, n_c 的變化情況。可以看出 b_f 越小,codeword 個數越少, 當 $b_f = 2$ 時, 每個 codebook 只剩下一個 codeword, 已經無法再更少。在實作 codebook model 時, 我們將 codebook 實作成一個 sorted list: codebook 中的 codeword 排序方式爲 f 越大的排越前面。所以第一個之外的前幾個 codeword 通常都在描述影子或亮光中的背景色彩, codeword越少, 影子或亮光中的背景越容易被判定爲前景, 所以 b_f 越小,codebook model 辨識陰影或亮光中背景的能力越差。

[2]經過實驗後發現 camera 1 的 $R_{success}$ 最大值落在更大的範圍, 所以我們針對 camera 1 在 $b_f = 2 \sim 270$ 間一共訓練了66個模型。

表 4.4: 前景過濾後的 codebook model

camera 1		camera 2		camera 3		camera 4	
b_f	n_c	b_f	n_c	b_f	n_c	b_f	n_c
2	1	2	1	2	1	2	1
4	1.01855	4	1.22781	4	1.22275	4	1.09657
6	1.05882	6	1.5674	6	1.5521	6	1.24191
8	1.11358	8	1.93341	8	1.82348	8	1.34205
10	1.1834	10	2.4967	10	2.05502	10	1.42967
12	1.26247	12	3.09029	12	2.27401	12	1.52372
14	1.34065	14	3.6091	14	2.48846	14	1.62144
16	1.41279	16	4.01378	16	2.70374	16	1.72824
18	1.47778	18	4.31952	18	2.92684	18	1.8478
20	1.52995	20	4.54156	20	3.1256	20	1.96271
22	1.58987	22	4.77216	22	3.36038	22	2.10918
24	1.64501	24	4.96758	24	3.57155	24	2.26265
26	1.69434	26	5.12386	26	3.74557	26	2.40547
28	1.754	28	5.28818	28	3.94289	28	2.57934
30	1.80118	30	5.40347	30	4.08881	30	2.71217
32	1.85536	32	5.52581	32	4.24972	32	2.85448
34	1.91914	34	5.65342	34	4.43345	34	3.00953
36	1.99468	36	5.78894	36	4.64564	36	3.17724
38	2.03727	38	5.86056	38	4.76439	38	3.26659
40	2.0841	40	5.93333	40	4.89153	40	3.35815

n_c = 平均每個 codebook 含有的 codeword 個數。

背景修正演算法需要設定以下五個參數:

r_s: 此參數決定參考視窗的大小, 用於式 (3.8), 此參數與拍攝畫面的解析度相關, 以我們的實驗數據作測試, 當 $r_s < 2$ 時修正演算法的效果明顯變差, r_s 在2以上的效果差別不大, 但運算時間會逐漸增加。

r_r: 從參考視窗中骰選出參考點時使用的色彩差異臨界值, 通常設定在15到25之間, 適當的數值需要比較演算法的效果來決定, 用於式 (3.9)。

I_e: 此參數決定生長點的亮度下限, 用於判斷式 (3.11), 先用人工方式找出一些光源被遮蔽的畫面, 計算黑暗區域的亮度便可得到此參數的參考值。

T_r: 參考點數量的臨界值, 用於判斷式 (3.12), 此參數與 r_s 有關, 當 $r_s = 2$ 時適合的設定值在0到10之間。

κ: 參考點中背景像素的比例下限, 用於判斷式 (3.13), $\kappa = 0.5$ 代表參考點的背景多於前景, 但是考慮前景邊界有可能是內凹, 此數值應設定略小於0.5。

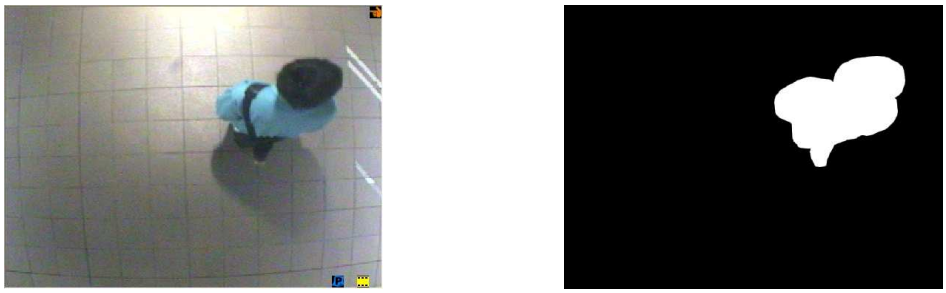
這些參數在實驗中的設定值如表4.5。

表 4.5: 背景修正演算法參數

r_s	r_r	I_e	T_r	κ
2	15	70	5	0.4

4.4 測試資料

我們從四個攝影機未用作訓練資料的畫面中, 挑選前景物體較多的畫面作爲測試資料。爲了能夠量化地評估背景切割結果好壞, 每張測試畫面都以人工方式做出相對應的 ground truth, 屬於前景的部分爲255, 背景的部分爲0, 如圖4.2, 這些 ground truth 記錄了測試畫面中眞正的背景像素與前景像素, 與背景切割的結果對照便能夠計算 TN、TP、FN 與 FP。



(a) (b)

圖 4.2: 測試資料範例。(a) 原圖; (b)ground truth。

測試資料的相關資訊如表4.6, 其中 camera 2的前景比例明顯比較高, 比較能夠代表車站中人潮擁擠時的狀況。

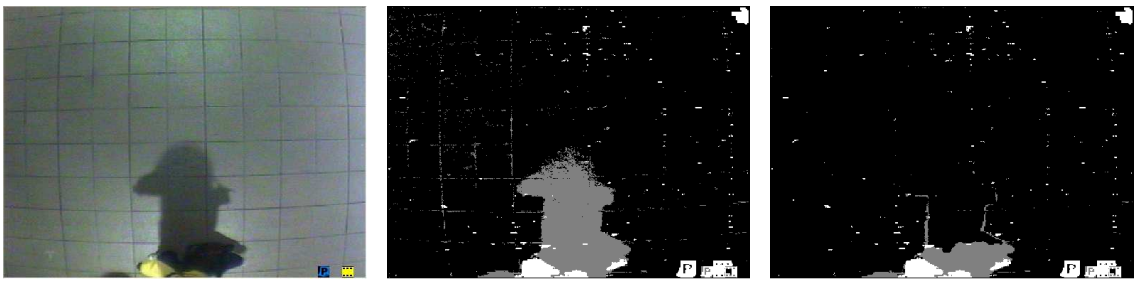
表 4.6: 測試資料

	測試影像張數	畫面中平均前景像素佔有比例 (%)
Camera 1	200	14.0615
Camera 2	209	32.1498
Camera 3	264	15.4574
Camera 4	218	13.3836

4.5 實驗結果

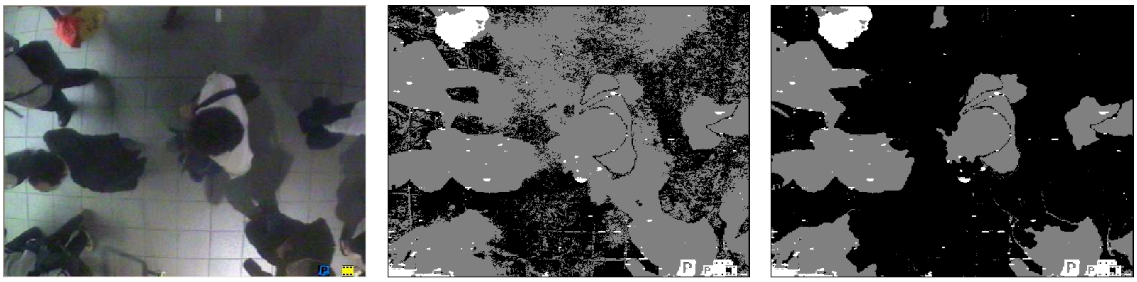
4.5.1 背景修正的效果

圖4.3∼4.6展示四個攝影機的背景修正演算法的效果, 背景變亮造成的錯誤和陰影部分都被正確改回背景。



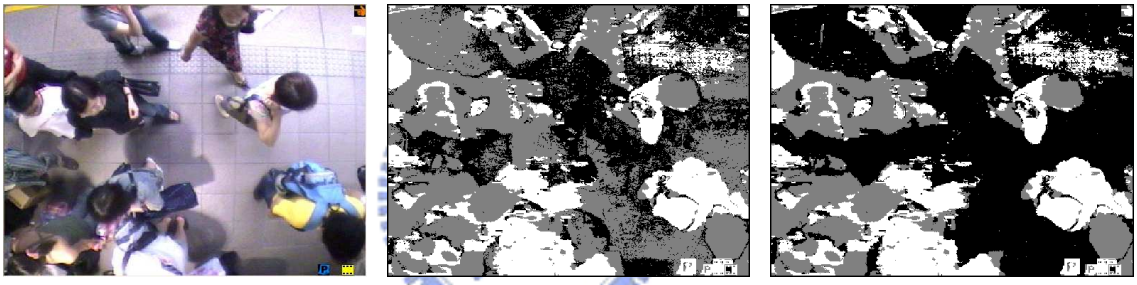
(a) (b) (c)

圖 4.3: (a)Camera 1 測試影像; (b) $b_f = 40$ 時 codebook model 背景切割結果; (c) $b_f = 40$,codebook+ 背景修正結果



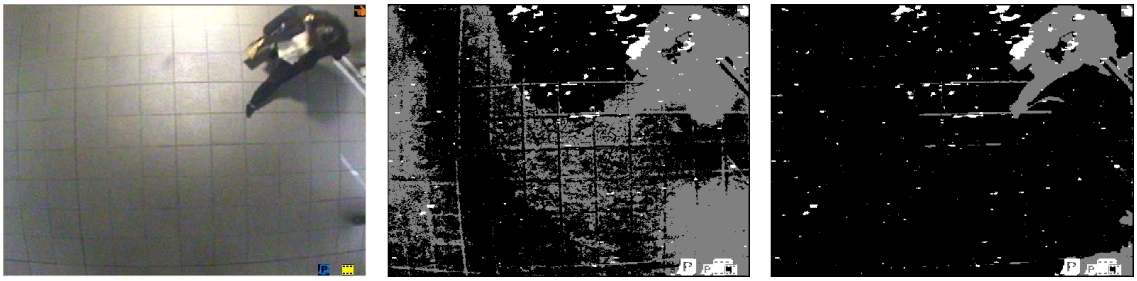
(a) (b) (c)

圖 4.4: (a)Camera 2 測試影像; (b) $b_f = 2$ 時 codebook model 背景切割結果; (c) $b_f = 2$,codebook+背景修正結果



(a) (b) (c)

圖 4.5: (a)Camera 3 測試影像; (b) $b_f = 6$ 時 codebook model 背景切割結果; (c) $b_f = 6$,codebook+背景修正結果



(a) (b) (c)

圖 4.6: (a)Camera 4 測試影像; (b) $b_f = 2$ 時 codebook model 背景切割結果; (c) $b_f = 2$,codebook+背景修正結果

4.5.2 實驗數據

透過實驗我們發現 camera 2∼4的成功率在 $b_f = 30$ 時趨於收斂, $R_{success}$ 極大值出現在 $b_f = 2 \sim 32$ 之間, 所以這三個實驗只列出 $b_f = 2, 4, \ldots, 38$ 的結果。

camera 1 的情況較爲特別, 由於該攝影機架設在車站中比較偏僻的角落, 拍攝畫面在大部分時間都只有背景, 所以訓練畫面中前景物體的出現量遠比其他攝影機少。陰影通常都伴隨著前景物體出現, 前景物體減少也意味著訓練畫面中陰影像素變少, 表4.3中的數據也反映了此現象 — camera 1 建構的 codebook model 樣本幾乎都集中在第一個 codeword , 所以從第二個 codeword 開始樣本數量遠比其他攝影機少, 陰影資料的減少造成使用 b_f 調整明暗容忍度的幅度較爲平緩。此推論可以從表4.4獲得證實 — 隨著 b_f 增加, camera 1 模型的 codeword 個數的增加速度明顯比其他三個攝影機慢。因此, 在 camera 1 的實驗中 $R_{success}$ 極大值出現位置比其他攝影機高出很多, 使用修正演算法時的 $R_{success}$ 極大值出現在 $b_f = 40$ 時; 未修正時的 $R_{success}$ 極大值出現在 $b_f = 126 \sim 142$ 之間, 所以我們只列出 $b_f = 2 \sim 254$ 之間20筆數據, 在 $R_{success}$ 極大值附近的數據比較密集一些。

以下用圖表列舉四個攝影機的實驗組 (codebook+ 修正演算法) 與對照組 (codebook), 在不同參數 b_f 下的成功率 ($R_{success}$)、平均背景錯誤率 (Average e_B)、最差背景錯誤率 (Worst e_B)、平均前景錯誤率 (Average e_F)、最差前景錯誤率 (Worst e_F)。

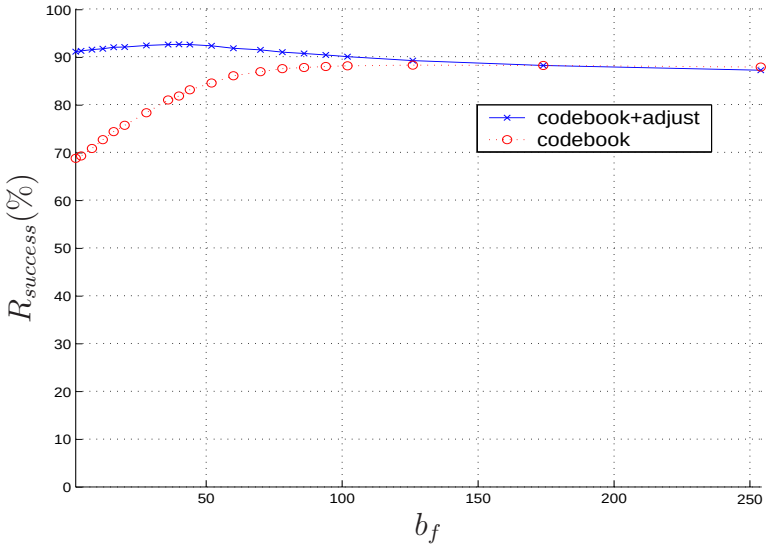
表 4.7: Camera 1 在 $b_f = 2 \sim 254$ 的實驗結果

codebook

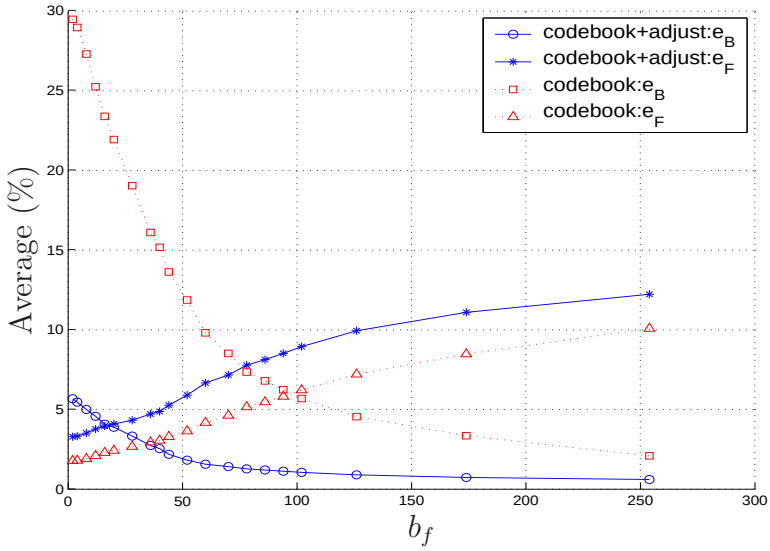
b_f	$R_{success}$ (%)	Average e_B (%)	Worst e_B (%)	Average e_F (%)	Worst e_F (%)
2	68.733299	29.488400	67.285103	1.778180	8.428300
4	69.225899	28.972300	67.188400	1.801860	8.501800
8	70.787598	27.303301	66.759300	1.909120	8.866150
12	72.644897	25.255199	66.250603	2.099850	9.401300
16	74.316101	23.397800	65.471603	2.286160	10.038900
20	75.636894	21.938101	64.412201	2.425020	10.626900
28	78.281494	19.045300	60.461800	2.673110	11.704900
36	80.945595	16.109699	53.800900	2.944660	12.673300
40	81.765800	15.179700	51.324902	3.054440	13.052100
44	83.079193	13.632900	47.050499	3.287850	13.746100
52	84.485199	11.871300	41.843399	3.643400	14.573200
60	86.005661	9.814550	35.449600	4.179760	15.991800
70	86.857521	8.519030	31.473801	4.623500	17.587400
78	87.496284	7.349520	28.159401	5.154130	19.236799
86	87.738190	6.796330	26.470100	5.465530	20.133699
94	87.937943	6.243040	24.836201	5.818970	21.118099
102	88.090210	5.689820	23.821800	6.220010	22.262400
126	88.235947	4.553690	21.499001	7.210380	24.827900
174	88.167191	3.355310	17.588400	8.477440	27.909401
254	87.827873	2.091840	10.901700	10.080300	32.428699

codebook+ 修正演算法

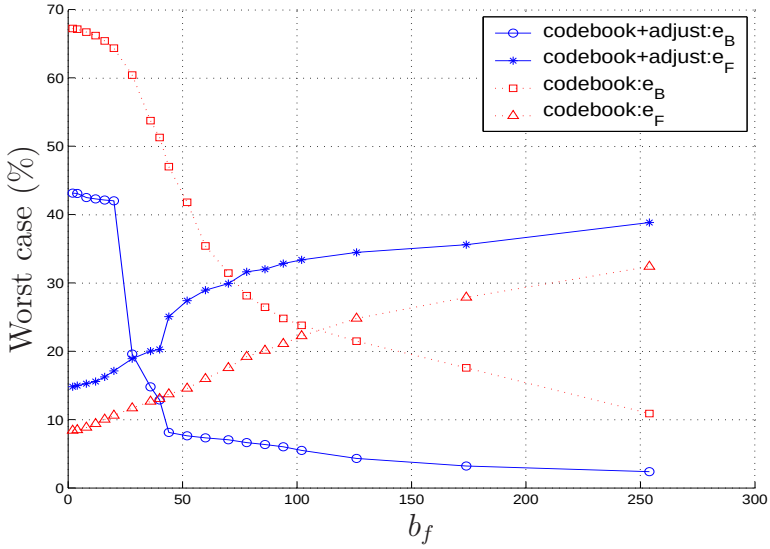
b_f	$R_{success}$ (%)	Average e_B (%)	Worst e_B (%)	Average e_F (%)	Worst e_F (%)
2	91.037399	5.676440	43.181801	3.286160	14.836100
4	91.212997	5.459600	43.134201	3.327440	14.997100
8	91.479698	5.002100	42.534901	3.518250	15.277600
12	91.658096	4.572650	42.340302	3.769230	15.591200
16	91.956696	4.079290	42.173100	3.964020	16.269199
20	92.019951	3.888850	42.050900	4.091170	17.141800
28	92.344185	3.332810	19.618299	4.323040	18.943899
36	92.538368	2.743590	14.815900	4.718060	20.054001
40	92.585747	2.545390	12.897900	4.868870	20.293699
44	92.540985	2.190800	8.142620	5.268240	25.088800
52	92.280190	1.820910	7.637490	5.898930	27.437901
60	91.762909	1.566250	7.351800	6.670900	28.965200
70	91.417206	1.416880	7.057830	7.165910	29.929899
78	90.946182	1.275080	6.642240	7.778750	31.660601
86	90.665443	1.202810	6.374150	8.131780	32.032700
94	90.354858	1.127800	6.029980	8.517360	32.863899
102	89.995567	1.056080	5.502080	8.948350	33.410400
126	89.166092	0.897403	4.324130	9.936530	34.506599
174	88.167213	0.736088	3.220200	11.096700	35.629700
254	87.160873	0.607415	2.390560	12.231800	38.872700



(a) 成功率



(b) 平均背景錯誤率 (e_B) 與平均前景錯誤率 (e_F)



(c) 最差背景錯誤率 (e_B) 與最差前景錯誤率 (e_F)

圖 4.7: camera 1在 $b_f = 2, 4, \ldots, 38$ 的實驗結果

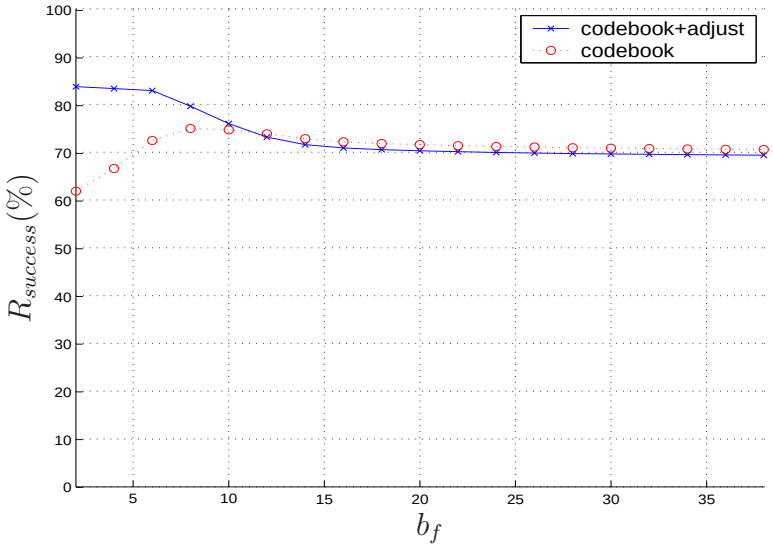
表 4.8: Camera 2 在 $b_f = 2, 4, \ldots, 38$ 的實驗結果

codebook

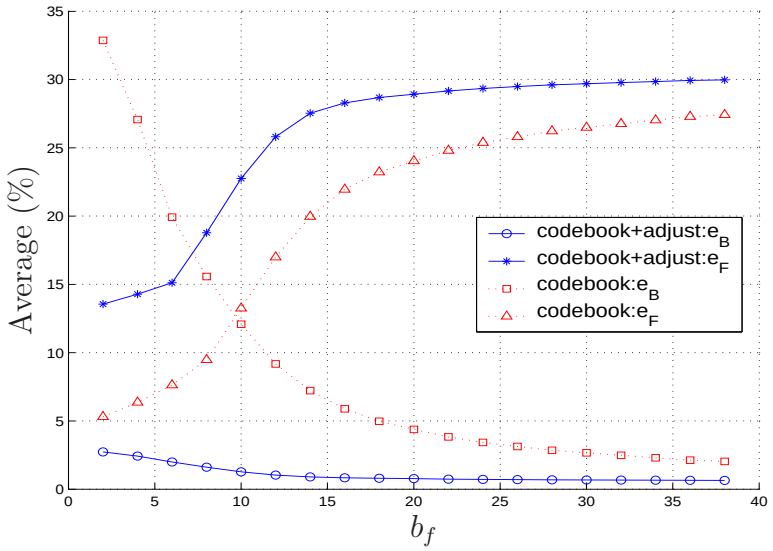
b_f	$R_{success}$ (%)	Average e_B (%)	Worst e_B (%)	Average e_F (%)	Worst e_F (%)
2	61.841499	32.856098	55.526901	5.302440	16.537300
4	66.581604	27.066200	53.355801	6.352250	20.670401
6	72.453499	19.920799	45.440899	7.625680	24.939199
8	74.948898	15.574600	36.214001	9.476470	29.259199
10	74.683800	12.073500	30.625500	13.242600	38.087002
12	73.846405	9.168040	26.076200	16.985600	46.219299
14	72.822098	7.214910	24.540199	19.962999	51.464699
16	72.167000	5.885800	23.487499	21.947201	54.646599
18	71.816658	4.969670	22.591600	23.213699	56.571800
20	71.591064	4.376090	21.804899	24.032900	57.736900
22	71.374176	3.832320	20.944700	24.793501	58.706699
24	71.206406	3.420110	20.103201	25.373501	59.437000
26	71.074791	3.124860	19.386400	25.800301	59.955101
28	70.936256	2.847880	18.604401	26.215900	60.477299
30	70.850876	2.663890	17.969299	26.485201	60.802799
32	70.763145	2.481600	17.262400	26.755301	61.123699
34	70.681030	2.299490	16.516100	27.019501	61.446098
36	70.594360	2.123140	15.735100	27.282499	61.771198
38	70.549484	2.036480	15.307100	27.414101	61.942501

codebook+ 修正演算法

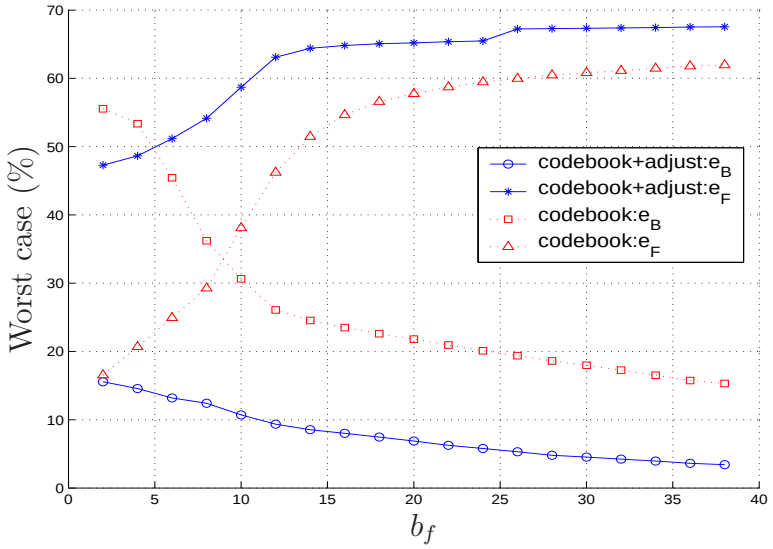
b_f	$R_{success}$ (%)	Average e_B (%)	Worst e_B (%)	Average e_F (%)	Worst e_F (%)
2	83.721802	2.729010	15.550300	13.549100	47.272499
4	83.300499	2.424100	14.564400	14.275300	48.638302
6	82.893196	1.991730	13.197500	15.115000	51.168098
8	79.614502	1.613360	12.408800	18.772200	54.136799
10	75.985695	1.268330	10.711200	22.746000	58.693298
12	73.156929	1.034240	9.349540	25.808800	63.081600
14	71.578812	0.894933	8.536470	27.526199	64.390999
16	70.889305	0.833694	8.013230	28.277000	64.803497
18	70.529663	0.795653	7.462040	28.674700	65.055000
20	70.309570	0.771521	6.874620	28.918900	65.189102
22	70.104393	0.736808	6.257700	29.158800	65.360901
24	69.945671	0.713171	5.788280	29.341200	65.469101
26	69.817772	0.699488	5.305920	29.482700	67.228203
28	69.706238	0.686616	4.796650	29.607100	67.284103
30	69.631386	0.678070	4.522870	29.690599	67.324997
32	69.562660	0.667642	4.238740	29.769699	67.375198
34	69.493950	0.657115	3.948390	29.848900	67.430099
36	69.422455	0.646524	3.618190	29.931000	67.510300
38	69.386902	0.640566	3.414280	29.972601	67.545998



(a) 成功率



(b) 平均背景錯誤率 (e_B) 與平均前景錯誤率 (e_F)



(c) 最差背景錯誤率 (e_B) 與最差前景錯誤率 (e_F)

圖 4.8: camera 2在 $b_f = 2, 4, \ldots, 38$ 的實驗結果

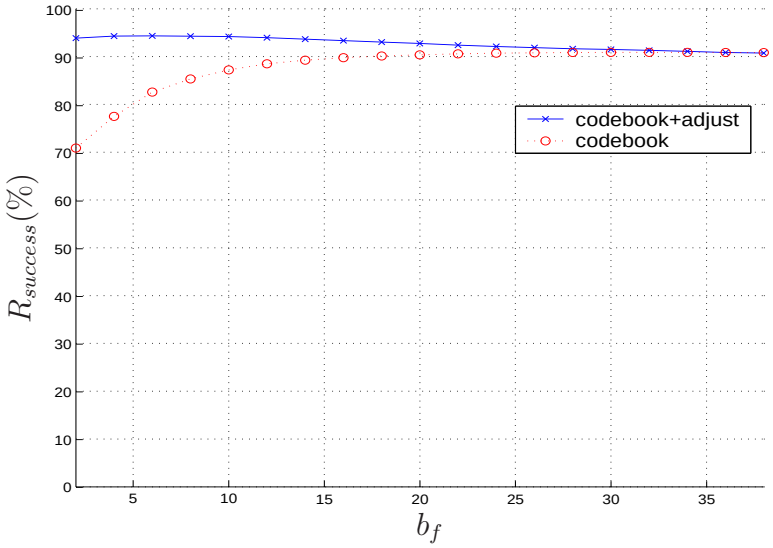
表 4.9: Camera 3 在 $b_f = 2, 4, \ldots, 38$ 的實驗結果

codebook

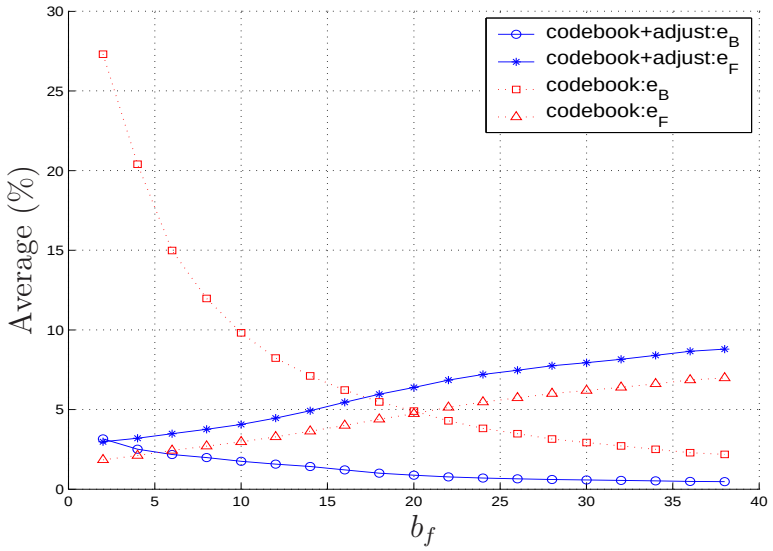
b_f	$R_{success}$ (%)	Average e_B (%)	Worst e_B (%)	Average e_F (%)	Worst e_F (%)
2	70.861000	27.297100	51.112202	1.841910	8.873400
4	77.493896	20.393999	44.122700	2.112050	10.829700
6	82.583702	14.982000	28.874599	2.434290	13.870900
8	85.328499	11.968900	23.406200	2.702700	16.002100
10	87.217499	9.810870	19.165899	2.971650	17.153200
12	88.489204	8.226650	17.954300	3.284160	18.333700
14	89.252602	7.108930	17.114300	3.638470	19.719200
16	89.775299	6.222120	16.344200	4.002590	21.153799
18	90.133003	5.474620	15.513000	4.392420	22.760799
20	90.363998	4.897150	14.724300	4.738820	24.262800
22	90.577301	4.288100	13.771000	5.134560	26.057100
24	90.714783	3.814870	12.944400	5.470300	27.568300
26	90.792892	3.477830	12.271600	5.729270	28.684700
28	90.851593	3.145150	11.519600	6.003280	29.834700
30	90.880051	2.925790	10.979800	6.194130	30.680901
32	90.891556	2.714220	10.331900	6.394320	31.669399
34	90.886322	2.501570	9.642480	6.612050	32.816299
36	90.861191	2.286660	8.856840	6.852180	34.103401
38	90.834709	2.183750	8.408640	6.981560	34.833199

codebook+ 修正演算法

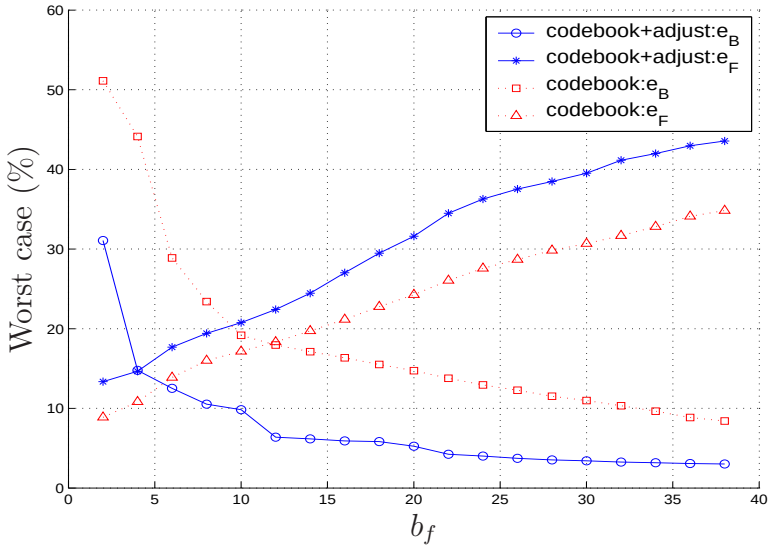
b_f	$R_{success}$ (%)	Average e_B (%)	Worst e_B (%)	Average e_F (%)	Worst e_F (%)
2	93.872200	3.151210	31.070601	2.976600	13.350700
4	94.293495	2.511640	14.770900	3.194920	14.662200
6	94.344803	2.177170	12.515900	3.477980	17.686199
8	94.265198	1.980340	10.519200	3.754490	19.410200
10	94.190201	1.749030	9.824650	4.060810	20.762600
12	93.964600	1.573110	6.376740	4.462320	22.409401
14	93.649902	1.425220	6.165060	4.924890	24.447500
16	93.335304	1.213570	5.917670	5.451080	27.018200
18	93.039055	1.005480	5.823470	5.955420	29.479700
20	92.740829	0.877710	5.240710	6.381450	31.598000
22	92.383957	0.771158	4.243910	6.844920	34.487999
24	92.103401	0.698722	4.006360	7.197880	36.268902
26	91.883301	0.651040	3.719120	7.465680	37.525002
28	91.643616	0.609312	3.531240	7.747060	38.473099
30	91.488739	0.576412	3.417900	7.934840	39.507198
32	91.297661	0.548062	3.257460	8.154220	41.144199
34	91.084045	0.519107	3.175690	8.396930	41.980598
36	90.857750	0.488575	3.079940	8.653630	42.957699
38	90.737831	0.473427	3.016800	8.788750	43.550800



(a) 成功率



(b) 平均背景錯誤率 (e_B) 與平均前景錯誤率 (e_F)



(c) 最差背景錯誤率 (e_B) 與最差前景錯誤率 (e_F)

圖 4.9: camera 3在 $b_f = 2, 4, \ldots, 38$ 的實驗結果

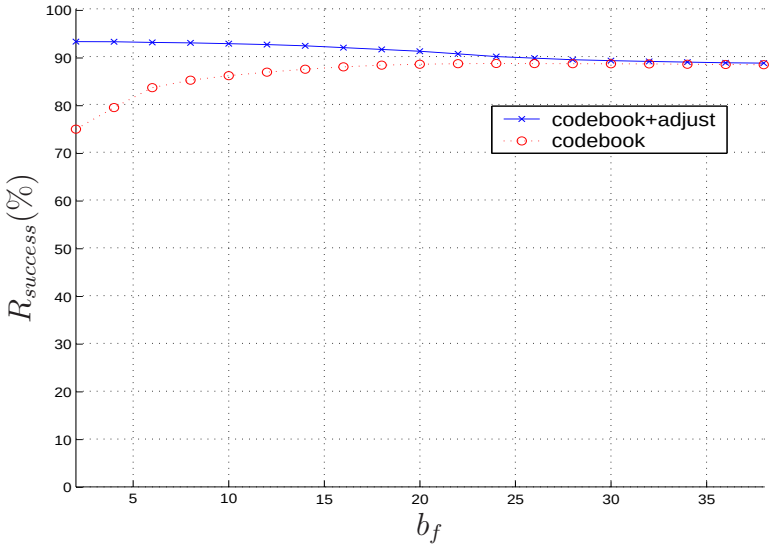
表 4.10: Camera 4 在 $b_f = 2, 4, \ldots, 38$ 的實驗結果

codebook

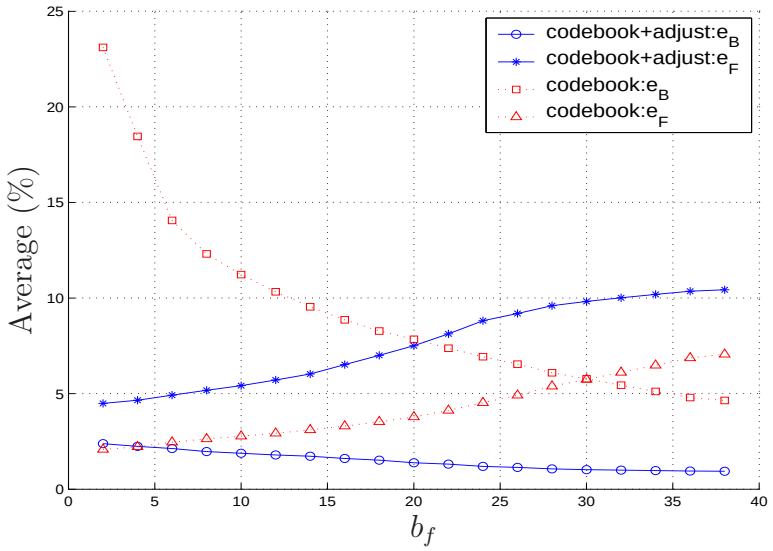
b_f	$R_{success}$ (%)	Average e_B (%)	Worst e_B (%)	Average e_F (%)	Worst e_F (%)
2	74.816002	23.103500	52.051601	2.080490	12.977600
4	79.334396	18.448200	49.318401	2.217470	13.454800
6	83.490494	14.052600	46.957802	2.456960	16.258801
8	85.069099	12.304400	45.344101	2.626430	17.717300
10	86.010803	11.220800	43.553398	2.768330	18.731199
12	86.751801	10.321700	41.021000	2.926450	19.951000
14	87.360794	9.539680	38.014599	3.099560	21.232000
16	87.850197	8.855170	34.717300	3.294630	22.512899
18	88.198189	8.272650	31.280701	3.529140	23.684601
20	88.387268	7.837100	28.584299	3.775660	24.526199
22	88.507217	7.371190	25.958200	4.121610	25.445900
24	88.550331	6.930090	23.822300	4.519600	26.262600
26	88.548912	6.540620	22.222000	4.910420	26.896601
28	88.523376	6.090110	20.637300	5.386550	27.593201
30	88.491501	5.766640	19.551500	5.741830	28.095200
32	88.457085	5.438720	18.534500	6.104140	28.631399
34	88.404556	5.114410	17.566200	6.480970	29.220400
36	88.343117	4.794430	16.632500	6.862410	29.747200
38	88.306740	4.638380	16.181700	7.054890	30.027700

codebook+ 修正演算法

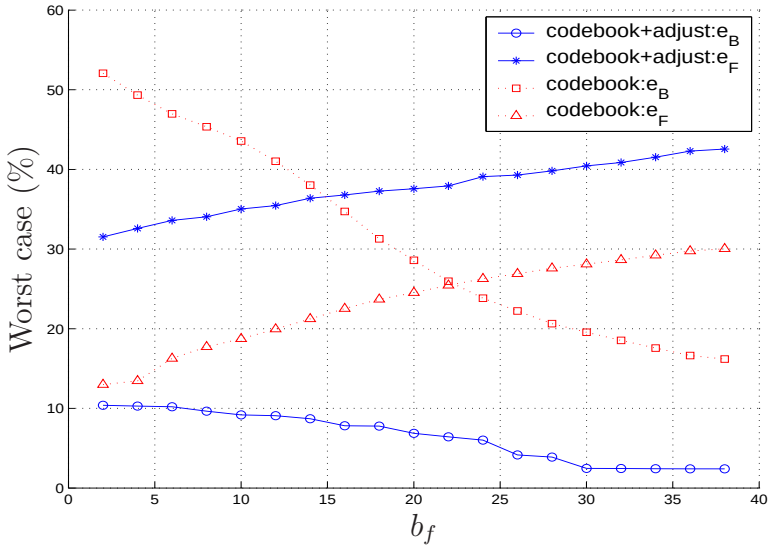
b_f	$R_{success}$ (%)	Average e_B (%)	Worst e_B (%)	Average e_F (%)	Worst e_F (%)
2	93.135391	2.379120	10.377400	4.485440	31.513100
4	93.106567	2.240530	10.287300	4.652860	32.576698
6	92.954269	2.126760	10.203500	4.918960	33.586399
8	92.861610	1.965660	9.645580	5.172720	34.049099
10	92.707893	1.878970	9.172020	5.413130	35.023701
12	92.502014	1.786730	9.080930	5.711220	35.449600
14	92.251938	1.724250	8.688630	6.023790	36.375999
16	91.881401	1.604710	7.815010	6.513930	36.787498
18	91.483902	1.519200	7.770500	6.996930	37.270302
20	91.115425	1.382920	6.857540	7.501700	37.567902
22	90.566734	1.310000	6.421240	8.123300	37.923000
24	90.003288	1.190040	6.008240	8.806640	39.083302
26	89.675323	1.136560	4.147130	9.188100	39.288799
28	89.341820	1.062610	3.883180	9.595610	39.810501
30	89.156570	1.022260	2.464060	9.821160	40.438801
32	88.995621	0.995899	2.448010	10.008500	40.863201
34	88.838837	0.973016	2.425760	10.188100	41.528702
36	88.699379	0.949453	2.407130	10.351200	42.297298
38	88.636230	0.935244	2.399360	10.428500	42.547798



(a) 成功率



(b) 平均背景錯誤率 (e_B) 與平均前景錯誤率 (e_F)



(c) 最差背景錯誤率 (e_B) 與最差前景錯誤率 (e_F)

圖 4.10: camera 4在 $b_f = 2, 4, \ldots, 38$ 的實驗結果

4.5.3 背景修正演算法執行效率

在實驗中用來測試背景切割的電腦配備爲 Intel pentium 4 3.0 GHz, 2048 MB DDR RAM。表 4.11列舉在 Camera 4 的測試畫面使用背景修正演算法的花費時間。

表 4.11: 背景修正演算法在 Camera 4 每張測試畫面的平均花費時間

b_f	process time (sec)
2	0.479
4	0.394
6	0.318
8	0.290
10	0.274
12	0.261
14	0.250
16	0.244
18	0.239
20	0.237
22	0.236
24	0.235
26	0.227
28	0.218
30	0.210
32	0.201
34	0.191
36	0.181
38	0.176

4.6 討論

首先列出四組攝影機 $R_{success}$ 最高的數據如表4.12。

由實驗結果可歸納出以下幾個結論:

I. 若希望陰影與亮光部分有較好的背景切割結果, 其代價爲辨識前景像素的錯誤率提高, 所以背景切割方法必須在前景錯誤率與背景錯誤率之間作取捨 (trade-off)。此現象在圖4.7, 4.8, 4.9, 4.10中的 (b)(c) 可以明顯看出: 背景錯誤率與 b_f 成反比; 前景錯誤率與 b_f 成正比。

II. 背景模型的切割結果在使用修正演算法後有明顯改進, 改善的部分可以分爲三個方面:

表 4.12: 四組實驗的最佳結果

	b_f	$R_{success}$ (%)	Average e_B (%)	Worst e_B (%)	Average e_F (%)	Worst e_F (%)
camera 1						
codebook	126	88.235947	4.553690	21.499001	7.210380	24.827900
codebook+修正演算法	40	92.585747	2.545390	12.897900	4.868870	20.293699
camera 2						
codebook	8	74.948898	15.574600	36.214001	9.476470	29.259199
codebook+修正演算法	2	83.721802	2.729010	15.550300	13.549100	47.272499
camera 3						
codebook	32	90.891556	2.714220	10.331900	6.394320	31.669399
codebook+修正演算法	6	94.344803	2.177170	12.515900	3.477980	17.686199
camera 4						
codebook	24	88.550331	6.930090	23.822300	4.519600	26.262600
codebook+修正演算法	2	93.135391	2.379120	10.377400	4.485440	31.513100

1. **正確性** 由表4.12可看出使用修正演算法後成功率可以提升 3.5% ∼ 9% 左右。

2. **光影變化的容忍性** 在表4.12中使用修正演算法的平均背景錯誤率都低於3%, 最差背景錯誤率也只有15.5503%; 相較之下, 沒使用修正演算法的平均背景錯誤率在 2% ∼ 16% 之間, 最差背景錯誤率高達 10% ∼ 37% 。可見雖然藉由增加 b_f 可提升 codebook model 辨識陰影與亮光背景的能力, 使用修正演算法的效果卻更好。

3. **可靠性** 使用修正演算法後, 較不容易發生特別差的切割結果。觀察表4.12的最差前景錯誤率與最差背景錯誤率, 雖然修正後的前景錯誤率有部分會增加 (camera 2 與 camera 4) , 但背景錯誤率也大幅減少 (只有 camera 3 小幅增加)。整體來說, 最差的情況有比原來改善。

III. 與修正演算法搭配的背景模型, 對光影變化的容忍應該要設定得低一點, 換句話說, 辨識陰影與亮光背景的工作, 應該儘量交由修正演算法處理。其原因可參考表4.12, 使用修正演算法時 $R_{success}$ 極大值都出現在 b_f 較低處。[3]

[3]唯一的例外是 camera 1 的 $R_{success}$ 極大值出現在 b_f 較高 (=40) 的情況, 不過 $b_f = 2$ 的 $R_{success}$ 與極大值相差並不大 (約1.5%左右)。

Chapter 5

結論與未來展望

5.1 結論

本論文提出了一個有效的背景切割改良技術, 現有的方法主要是使用背景模型做背景切割, 一個好的背景模型可以描述多個背景色彩, 允許背景物體少量的來回移動, 容許背景物體一定程度的明暗變化,codebook model 即爲其中之一。然而現有的背景模型技術只能描述 pixel-wise 的背景色彩資訊, 對陰影與亮光中的背景辨識能力仍然有限。在前景物體較多、陰影與亮光頻繁出現的攝影地點, 背景模型的切割結果仍有改良的空間。此外在畫面擷取時間間隔較長、相鄰畫面無連續性的情況下, 背景模型難以透過背景調適改善背景切割的結果。我們提出的背景修正演算法可以正確地辨識陰影與亮光處的背景, 不需透過背景調適也能在突發的背景明暗變化中得到良好的背景切割結果。

事實上, 我們提出的背景切割技術底層使用的背景模型並不限定只能用 codebook model, 背景修正演算法可以和現有的各種背景模型搭配使用, 整合的方法分爲兩個步驟:

1. **建構參考畫面**

   有些背景模型本身即爲一張參考畫面, 如[4, 5, 6]的方法, 在[8]使用卡曼濾波器的預測結果可視爲一張參考畫面, 此時可略過本步驟。如果使用的模型是 UniGaussian model[7], 可以用 Gaussian 的平均值(mean) 做爲參考畫面。

   若是使用的背景模型類別屬於多色背景模型, 建構參考畫面又細分爲兩個子步驟:

   (a) 針對每個像素座標 (i, j) 建構中心點集合 (S_{ij}), 這裡的中心點是用來代表一個色彩分

佈的點。[1] 若是使用高斯混合模型[1], 每個高斯成員的平均值可以組成一個中心點集合。在無參數模型中, 可以藉由 mean shift-based mode detection[11]找出眾數(mode), 用這些眾數組成中心點集合。

(b) 假設測試畫面的像素爲 x_{ij} , 對於中心點集合中的每個中心點爲$s_{ij}^k \in S_{ij}$, 套用公式 (3.2) 計算色差 $colordist(x_{ij}, s_{ij}^k)$, 以色差最小的中心點做爲參考畫面的像素。

2. **重新標記輸出結果**

假設M爲現在所使用的背景模型, 其參考畫面在座標 (i, j) 的像素值爲 B_{ij}^M , 使用 M 對測試畫面中的像素 x_{ij} 作背景切割的結果爲

$$\hat{C}_{ij} = \begin{cases} \text{“BG”} & x_{ij} \text{ 是背景。} \\ \text{“F”} & x_{ij} \text{ 是前景。} \end{cases}$$

現在建立一個新的標記如下:

$$C_{ij} = \begin{cases} \text{“BG”} & \text{If } \hat{C}_{ij} = \text{“BG”} \\ \text{“FL”} & \text{If } \hat{C}_{ij} = \text{“F”} \text{ and } colordist(x_{ij}, B_{ij}^M) \le \epsilon \\ \text{“FC”} & \text{If } \hat{C}_{ij} = \text{“F”} \text{ and } colordist(x_{ij}, B_{ij}^M) > \epsilon \end{cases}$$

ϵ is a threshold.

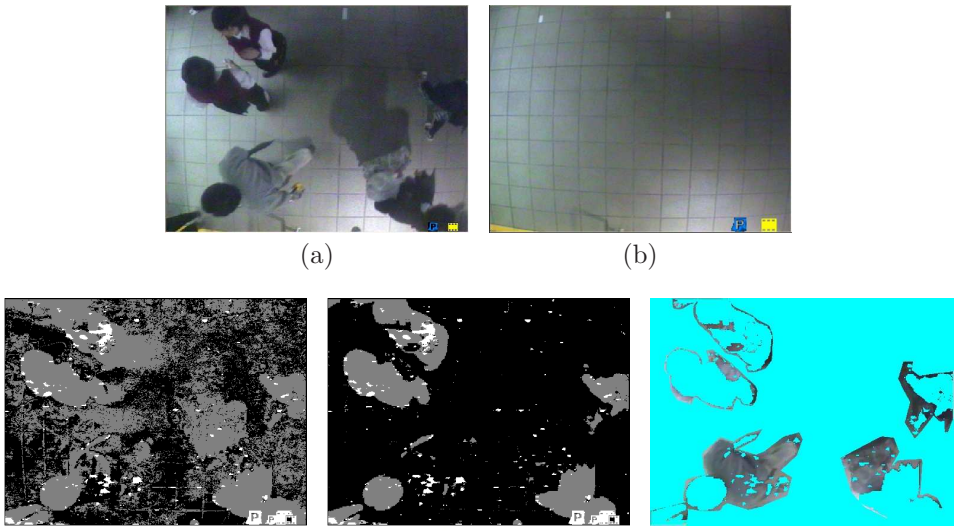
有了新的標記 C 與原來的測試畫面 x , 便可以套用背景修正演算法改良原來的背景切割結果。

5.2 未來展望

雖然背景修正演算法可以解決背景光影變化的問題, 前景部分的錯誤仍然有待解決, 由實驗結果表4.12可以看出, 使用修正演算法後的前景錯誤率仍然偏高 (平均3%∼14%, 最差17%∼48%), 前景物體往往是監視系統所要觀察的目標, 所以前景判斷錯誤比背景錯誤更加嚴重。當背景模型的切割結

[1]若是從 Vector Quantization 的觀點來看, 中心點即爲 codeword , 中心點集合就是指 codebook , 不過爲了避免與 codebook model[9] 中使用的 codeword 和 codebook 名詞混淆, 所以這裡暫稱中心點與中心點集合。

果在前景部分出現嚴重錯誤時, 修正演算法可能會使錯誤更嚴重, 因爲有大量前景像素被誤判爲背景時, 附近的前景像素可能會被演算法視爲陰影或亮光中的背景, 導致錯誤的範圍擴大, 我們在圖 5.1 中以實例說明前景錯誤的問題。



(a) (b) (c) (d) (e)

圖 5.1: (a)Camera 2 其中一張測試畫面; (b)Camera 2 只有背景時的樣本畫面; (c) 未使用修正演算法, $b_f = 2$ 的背景切割結果: 黑色代表背景, 灰色代表前景且 C_{ij} = "FL", 白色代表前景且 C_{ij} = "FC"; (d) 使用修正演算法, $b_f = 2$ 的背景切割結果; (e) 修正演算法切割結果的前景錯誤 (FN), 非藍色的部分為發生前景錯誤的像素。可發現畫面左下的人穿的衣服、畫面右下的人手上拿的白色物體與背景色彩非常接近, 亮度也與 (b) 中的背景相差不遠, 這些像素大部分在 (c) 中已經被切割為背景, 剩下的少部分對於修正演算法來說, 與陰影或亮光中的背景並無差別, 所以也被誤改為背景。

上述問題的根本原因在於有些前景物體的色彩與背景過於接近, 單靠色彩資訊不足以正確地切割這些前景。尋找可以有效切割這類前景的特徵 (feature), 是未來值得努力的研究方向。

參考文獻

[1] C. Stauffer and W. E. L. Grimson, "Adaptive background mixture models for real-time tracking," in *Proceedings of the IEEE Computer Science Conference on Computer Vision and Pattern Recognition(CVPR-99)*, Los Alamitos, June 23–25 1999, pp. 246–252, IEEE.

[2] A. M. Elgammal, D. Harwood, and L. S. Davis, "Non-parametric model for background subtraction," in *European Conference on Computer Vision*, 2000, pp. II: 751–767.

[3] M. Cristani, M. Bicego, and V. Murino, "Integrated region- and pixel-based approach to background modelling," in *IEEE Workshop on Motion and Video Computing*, 2002, pp. 3–8.

[4] Paul L. Rosin and Tim Ellis, "Detecting and classifying intruders in image sequences," in *British Machine Vision Conference*, 1991, pp. 293–300.

[5] Y. H. Yang and M. D. Levine, "The background primal sketch: An approach for tracking moving objects," *Machine Vision and Applications*, vol. 5, pp. 17–34, 1992.

[6] P. L. Rosin and T. Ellis, "Image difference threshold strategies and shadow detection," in *British Machine Vision Conference*, 1995, pp. 347–356.

[7] C. R. Wren, A. Azarbayejani, T. J. Darrell, and A. P. Pentland, "Pfinder: Real-time tracking of the human body," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 19, no. 7, pp. 780–785, July 1997.

[8] Christof Ridder, Olaf Munkelt, and Harald Kirchner, "Adaptive background estimation and foreground detection using kalman-filtering," in *Proceedings of International Conference on Recent Advances in Mechatronics, ICRAM'95*, Aug. 14-16 1995, vol. I, pp. 193–199.

[9] Kyungnam Kim, Thanarat H. Chalidabhongse, David Harwood, and Larry S. Davis, "Real-time foreground-background segmentation using codebook model," *Real-Time Imaging*, vol. 11, no. 3, pp. 172–185, 2005.

[10] O. Javed, K. Shafique, and M. Shah, "A hierarchical approach to robust background subtraction using color and gradient information," in *IEEE Workshop on Motion and Video Computing*, 2002, pp. 22–27.

[11] D. Comaniciu and P. Meer, "Mean shift: A robust approach toward feature space analysis," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 24, no. 5, pp. 603–619, May 2002.