國立臺北大學
National Taipei University

專題製作

智慧型影像監控系統

期末成果報告

國立臺北大學 電機工程學系

指導老師：鄭穎仁

專題學生：李尚儒、蔡涵淳、孫子涵、王奕凱

目錄

I. 緒論

在科技普及的現今，從攝影機拍攝與擷取影像再也不是難事，而且幾乎零成本，然而，當這些影像需要花費人力去管理監控時，又是另一筆可觀的開銷了。因此，智慧影像監控(Intelligent video surveillance，IVS)系統的發展，不管是應用在商業管理部門，亦或是其他相關的研究領域，已經越來越受矚目[1]-[2]。

智慧影像監控系統意即在特定的環境下，即時監控畫面中不變的物體，或是追蹤短暫出現的物體，利用影像分析(VA，Video analytics)之技術，應用軟體演算法，自動追蹤感興趣的物件，並且分析出其特徵與行為。為了找到感興趣的物件，使用背景相減針對每幅畫面的比較找出差異，偵測出物件的移動或改變來定義物件存在，在此高斯混和模型(Gaussian mixture model)即是有效建立背景的演算法[3]-[4]。在感興趣的物件被定義出來後，智慧影像監控系統可以藉由追蹤感興趣的物件，加以分析物件之行為，在此粒子濾波器(Particle filter)已經很成功地被應用在物件追蹤上[5]。在追蹤的過程中，如果物件被遮蔽，就有可能導致追蹤失敗，在此卡曼濾波器(Kalman filter)提供了一個很好的解決方案[6]。

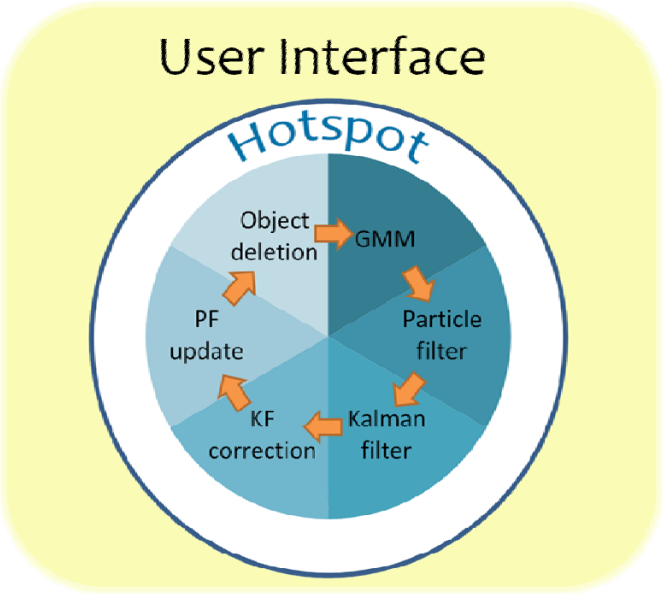
本計畫之目標為設計一智慧型影像監控系統，藉由追蹤人物，並分析人物行經之軌跡，加以獲得某些重要資訊，例如：目前在畫面中人物之總數，曾經出現在畫面中人物之總數，最多人停留之區域(熱點分析)等等。在此研究中我們使用高斯混合模型建立背景，再利用背景相減之方法，找出畫面中移動之物件，並將其定義為人物。在找出人物後，我們將利用粒子濾波器追蹤他們，並利用卡曼濾波器，預測人物走向之特性，解決物件重疊或被遮蔽之問題，以達成更良好之追蹤效果。最後，再藉由分析人物行經之軌跡，獲得重要資訊，並將這些資訊藉由使用者介面，提供給系統的使用者。此計畫的期程規劃如圖一。

圖一、 計畫期程規劃

II. 方法與過程

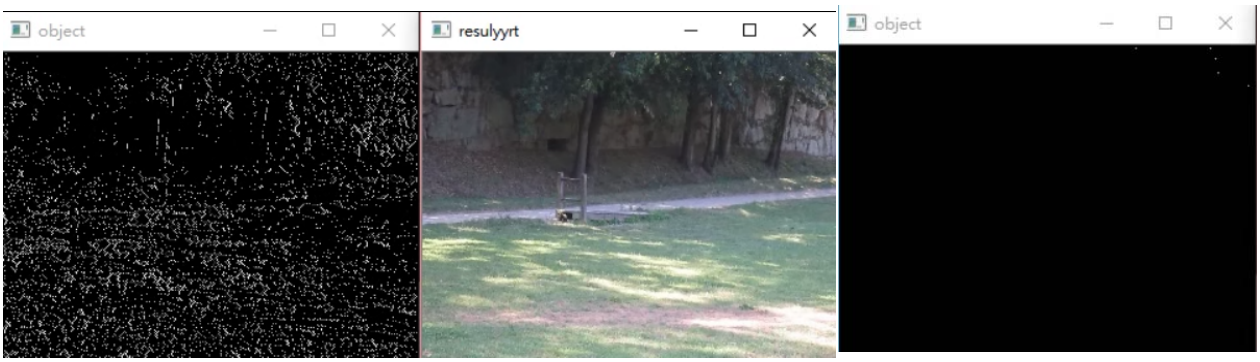
在此章節我們將描述智慧型影像監控系統的設計方法，其程式運作流程圖如圖二所示。

1. 首先我們將建立高斯混合模型分離前景背景，且不間斷地進行背景學習更新，如圖三，使程式更不容易被外界環境變化、光照突變、或是進入畫面便靜止的物體所影響，並且利用背景相減的方法找出在畫面邊緣出現之人物，如圖四。



圖二、 程式運作流程圖(HOTSPOT 與主程式為同心圓，表示平行處理)

高斯背景(學習前)　　影像　　高斯背景(學習後)

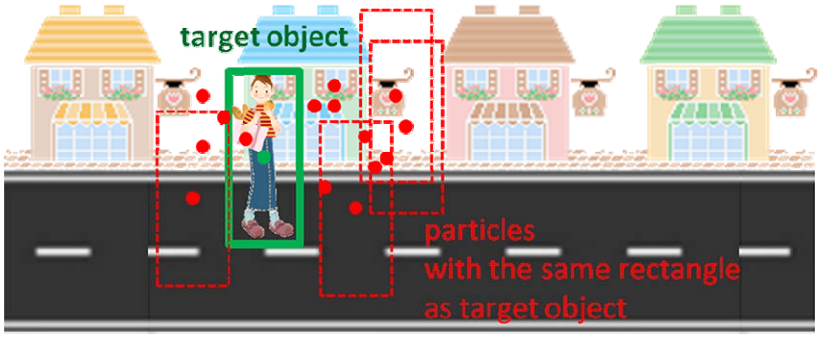


圖三、高斯背景更新機制

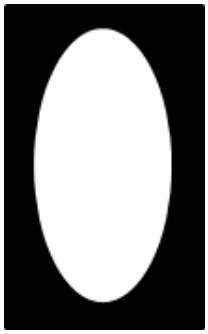
背景　　影像　　前景(二值化)

圖四、使用高斯混合模型分離前景、背景

2. 找到人物之後，針對每個人物，我們將建立一組粒子濾波器去追蹤此人物，粒子濾波器於目標物件附近，隨機灑出 N 個粒子，以每個粒子為中心取一個與目標相同大小的方框，如圖五，分別對兩方框取直方圖，採用 Bhattacharyya 比較方式，依結果給予不同的置信度，以每一粒子點置信度加權平均後找出最有可能之物件位置。
3. 框出目標人物時，主要所需資訊皆於方框中之橢圓部分，因此需要使用一個近似橢圓的遮罩，如圖六，將不必要的資訊去除後，使用粒子濾波器得到物件位置之測量值。



圖五、粒子濾波器執行過程



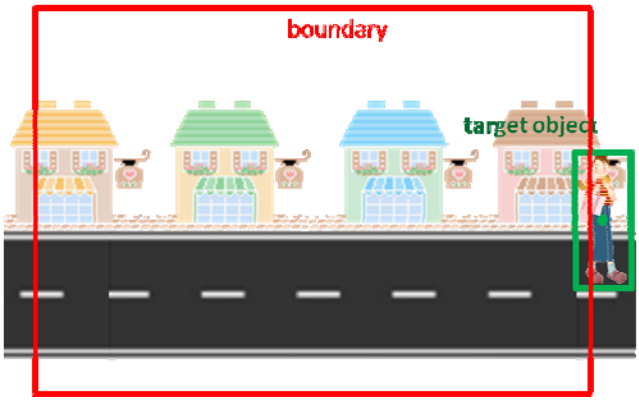
圖六、遮罩

4. 若追蹤目標中心超出設定的邊界條件，便將此物件於追蹤器中予以刪除。如圖七所示，物件中心已離開紅色邊界範圍，即進行刪除動作。

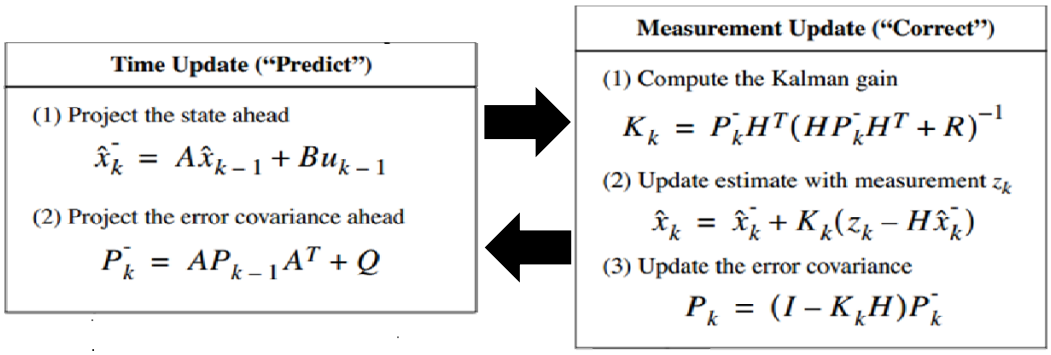
5. 接著我們還會利用卡曼濾波器，預測人物走向，解決物件重疊或被遮蔽之問題，以達成更良好之追蹤效果。卡曼濾波器分成預測和測量，預測來自對系統的建模推算得來的，另一部分使用外部的測量值(粒子濾波追蹤結果)，對模型進行修正，如圖八。

   卡曼濾波器在此計畫的運用層面如以下三點。

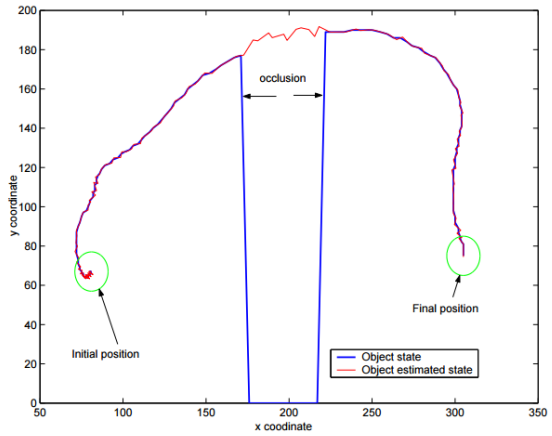
   (1) 藉由經驗模型以及測量值修正得到目標最佳估計位置，為物件最終追蹤位置。
   (2) 取得物體的移動速度，將其回傳給粒子濾波器，使粒子灑點範圍更有依據性。
   (3) 當物體被遮蔽、無法準確追蹤時，暫停測量修正部分，只運用預測部分估計其可能路徑，如圖九，當物體再度出現於視窗中時，方便持續追蹤物體不遺失資料。



圖七、物件刪除機制

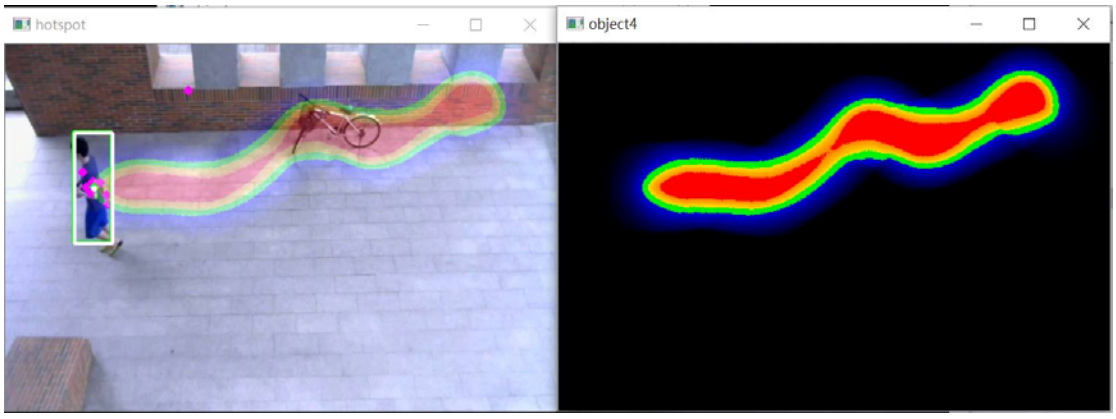


圖八、卡曼濾波器執行過程

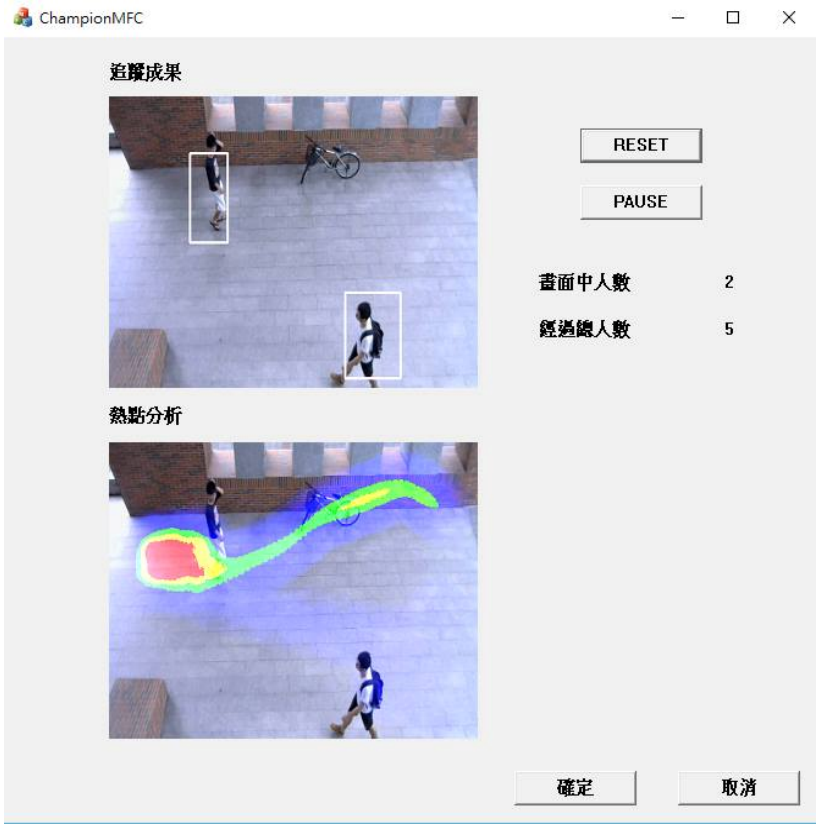


圖九、被遮蔽狀態下之可能路徑

6. 最後，藉由分析人物行經之軌跡，準確知道顧客的動線以及哪區商品較為熱門。應用卡曼濾波器修正後的物件位置為基準，根據所有顧客累積停留時間長短，運用三種顏色(R、G、B)調配出區域熱點分布圖，使其有漸層，讓使用者能夠一目瞭然。其中有顏色(非黑)的部分皆為有人走過的範圍，紅色為最熱門區域、綠色為中間值、藍色為最冷門區域，如圖十。
7. 將這些影像資訊藉由使用者介面提供給此智慧型影像監控系統的使用者，如圖十一。使用者介面會顯示追蹤結果畫面，以及區域熱點分析，因此將當前畫面中人數和總人數統計之變數回報在介面右方，讓使用者能夠更簡單方便的取得所需資訊。在此只顯示常用的兩幅影像，追蹤成果以及熱點分析，同時我們也有後台影像可供觀看。



圖十、熱點分析結果圖



圖十一、使用者介面

整個流程圖如圖二所示，接著我們將針對高斯混合模型、粒子濾波器及卡曼濾波器分別說明。

A. 高斯混合模型

高斯混合模型(Gaussian mixture model：GMM)是利用多個高斯分布的密度函數去趨近任何機率曲線形狀，在影像處理方面，常被用來建立背景影像模型。比起一般的背景相減(Background subtraction)，藉由後來得到的影像與原本取得的背景資訊相減的方式，來取得移動中的物體，高斯混合模型對於背景小幅度的變化，例如：飄動的樹葉、緩慢變化的光源、湖面水波等等，不會受到太多影響；背景相減在背景更新與初始化上也有著許多問題要解決，例如：原本在影像中被認定為背景的物體突然移動，就會造成原本物體的位置與移動後的位置，都被判定為移動物體。利用高斯混合模型，不僅不需要考慮背景更新所造成的錯誤判斷，對於不穩的背景也有較好的雜訊濾除。

高斯混合模型的主要概念為對整張影像的每個像素點做像素值(RGB、HSV或 Gray)的分析，建立 K 個高斯分布，依據每像素點所建立的混和高斯分布和馬式距離的平方做相似度分析，來判斷此點像素為陰影、背景或是前景，且更新背景資訊。

高斯混合模型有四個基本參數，w_i混合加權值(mixture weights)、μ_i平均值向量(mean vector)、Σ_i共變異矩陣(covariance matrix)以及 K 高斯分布的個數，如下所示：

$$\lambda = \{w_i, \mu_i, \Sigma_i\}, i = 1,2, \dots, K \tag{1}$$

對每一個像素而言，都可以用λ來表示像素模型。若資料$X_N = \{X_1, X_2, \dots, X_n\}$在 D 維空間中分布，第$i$個高斯分佈的機率密度函數如下：

$$g_i(x) = \frac{1}{((2\pi)^{-D/2}|\Sigma_i|)^{1/2}} \exp\left(-\frac{1}{2}\mathrm{D}\right) \tag{2}$$

其中 D 為馬式距離的平方，其定義如下：

$$\mathrm{D} = (X - \mu_i)^T \Sigma_i^{-1} (X - \mu_i)$$

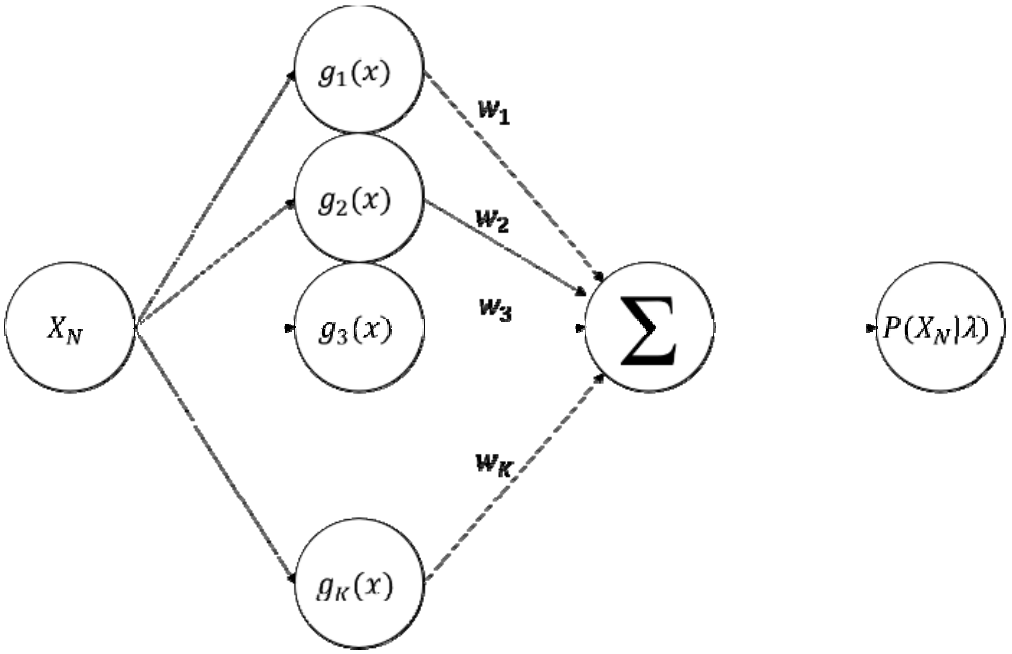
若將每個高斯分布的機率密度函數做權重分配即可得到完整的高斯混合模型，表示如下：

$$p(X_N|\lambda) = \sum_{i=1}^{K} w_i g_i(X_N) \quad (3)$$

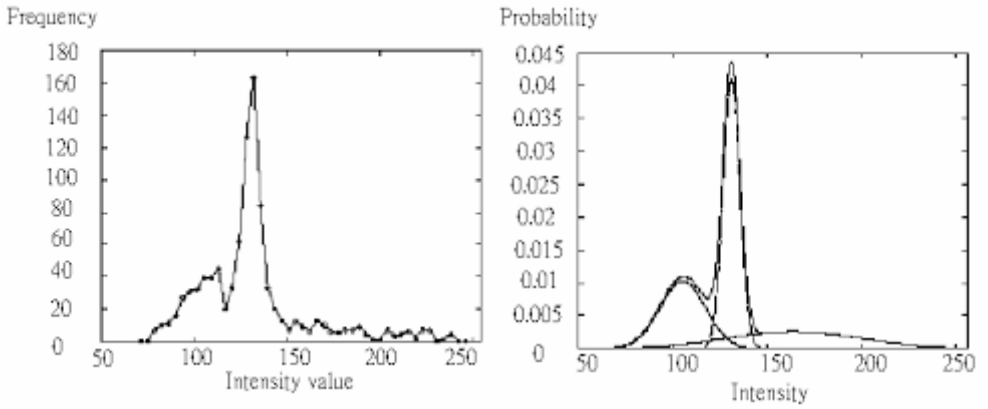
其中

$$\sum_{i=1}^{K} w_i = 1$$

上述之高斯混合模型之架構圖如圖十二所示。另外，圖十三為某強度次數統計與高斯混合模型機率函數的近似分佈圖，從圖十三左可看出，在某像素點的像素值出現的次數的曲線，可以用三個高斯分布密度函數近似為圖三右之高斯混合模型。



圖十二、 高斯混合模型架構圖



圖十三、 某強度次數統計與高斯混合模型機率函數的近似分佈圖

B. 粒子濾波器

粒子濾波器應用在物件追蹤上，主要是藉由視訊中目標物件的附近，隨機設置 N 個粒子，每一個粒子皆為一個樣本集合，以$S_t^{(n)}$表示，n 為第 n 個粒子，$n = 1, 2, \ldots, N$，t 為時刻，每一個粒子$S_t^{(n)}$都是物件的假設狀態(hypothetical)之集合，在此專題應用上，我們給予：

$$S_t^{(n)} = \left[x, y, H_x, H_y \right] \tag{4}$$

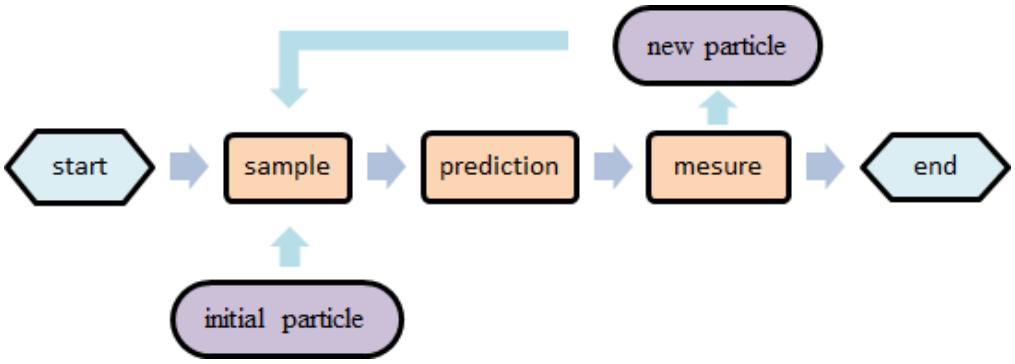
其中，(x, y)為粒子中心點位置，H_x與H_x分別代表方框的長和寬。且每一個粒子都有相對應的離散採樣機率$\pi_t^{(n)}$(亦可稱之為權重)，且

$$\sum_{n=1}^{N} \pi^{(n)} = 1 \tag{5}$$

因此，粒子與權重可以構成集合

$$\mathrm{S} = \left\{ (S_t^{(n)}, \pi_t^{(n)}) | n = 1, \ldots, N \right\} \tag{6}$$

整體系統流程分為三個主要步驟：取樣 (sample)、預測(prediction)與測量(measure)，如圖十四所示。



圖十四、粒子濾波器系統流程圖

接下來我們將針對取樣 (sample)、預測(prediction)與測量這三個主要步驟分別詳細說明。

1. 取樣

根據初始步驟，每一個粒子$S_{t-1}^{(n)}$會得到各自的權重$\pi_{t-1}^{(n)}$，藉由權重的大小來產生每一個粒子明顯不同的機率密度分佈。而取樣的主要目的是將機率密度較小的粒子去除，並將每一個粒子的機率密度值趨近一致。

2. 預測

預測的主要目的 是為了估測出每一個粒子移動後的位置，我們以簡單的動態模型表示：$s_t = As_{t-1} + w_{t-1}$，其中，A 為決定此模型的參數，w_{t-1}為系統的雜訊分布。

3. 測量

在此專題中，我們以色彩作為物件特徵，藉由計算 RGB 三維的直方圖，且將直方圖空間分成 16 塊(16x16x16 bins)以減少計算量，分析目標與觀測影像之間的相似度，我們採用巴氏(*Bhattacharyya*)相似係數來計算兩者之間的相似度，假設兩者的色彩分佈密度函數分別為

$$p = \{p^{(u)}\}_{u=1,\ldots,m}，q = \{q^{(u)}\}_{u=1,\ldots,m}$$

巴氏係數定義為

$$\rho[p,q] = \sum_{u=1}^{m} \sqrt{p^{(u)} q^{(u)}}，0 \le \rho \le 1 \quad (7)$$

而巴氏距離定義為

$$d = \sqrt{1 - \rho[p,q]} \quad (8)$$

若目標與觀測影像間的相似度愈高，ρ愈大，巴氏距離愈小。我們以高斯函數(Gaussian function)計算粒子權重：

$$\pi^{(n)} = \frac{1}{\sqrt{2\pi}\rho} e^{-\frac{(1-\rho[p_S(n),q])}{2\sigma^2}} \quad (9)$$

經由測量模型後，每一個粒子具有各自的權重，而粒子之權重經過正規化後，依期望值(expect)估測物件位置：

$$E[S] = \sum_{n=1}^{N} \pi^{(n)} s^{(n)} \quad (10)$$

C. 卡曼濾波器

卡曼濾波器是預測和測量的結合。預測來自經驗模型，是人對系統的建模推算得來的，另一部分是測量修正，來自外部的測量，是對模型的修正。而預測雜訊和測量雜訊分別是對模型和測量不確定性的定量描述。卡曼濾波器又稱「最佳線性濾波器」。顧名思義，想要運用卡曼濾波器消除信號中的雜訊，被量測的系統程序必須能用線性系統描述。

通常卡曼濾波器可以由系統預測模型及測量模型表示。卡曼濾波器之預測模型如下所示：

$$x_t = F x_{t-1} + w_t \quad (11)$$

其中 t 為時間變數，F 為狀態轉移矩陣，表示如何從 t-1 的狀態來推測 t 的預測系統狀態x_t，w_t為預測雜訊。另外，卡曼濾波器之測量模型如下所示：

$$y_t = Hx_t + v_t \tag{12}$$

y_t為時間 t 時的測量量，H 為測量矩陣，v_t為測量雜訊。在此w_t和v_t之間是不相關的。在此我們定義$\hat{x}_t^-$為時刻 t 之先驗估測狀態，$\hat{x}_t$為時刻 t 之後驗估測狀態，$e_t^- \equiv x_t - \hat{x}_t^-$為時刻 t 之先驗估測誤差，$e_t \equiv x_t - \hat{x}_t$為時刻 t 之後驗估測誤差。因此，我們可以得到先驗估測誤差之共變異矩陣

$$P_t^- = E[e_t^- e_t^{-T}] \tag{13}$$

以及後驗估測誤差之共變異矩陣

$$P_t = E[e_t e_t^T] \tag{14}$$

卡曼濾波器的操作包括兩個階段：「預測」與「更新」。在預測階段，濾波器使用上一狀態的估計，做出對當前狀態的估計：

$$\hat{x}_t^- = F\hat{x}_{t-1} \tag{15}$$

$$P_t^- = FP_{t-1}F^T + Q \tag{16}$$

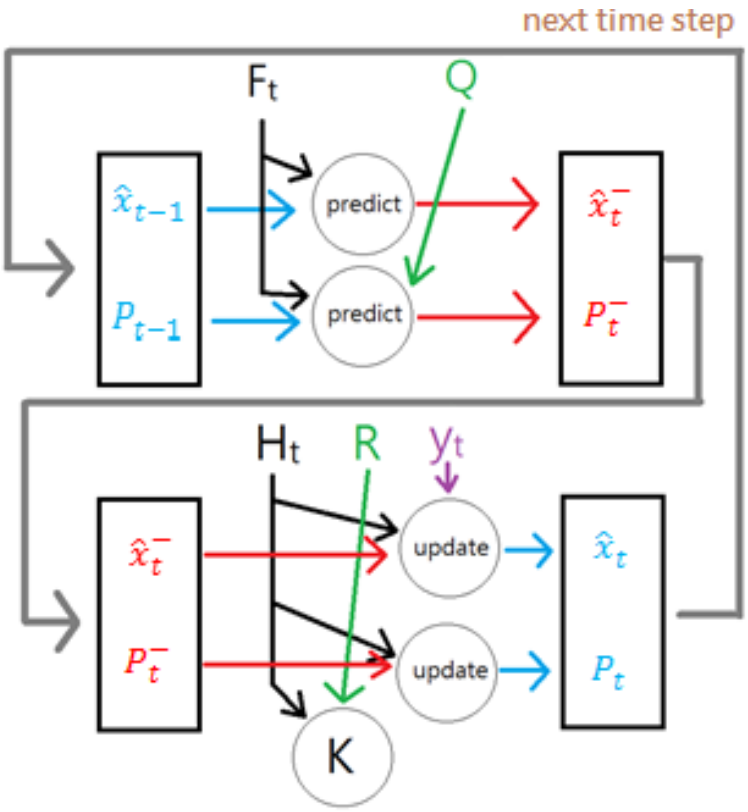
其中 Q 為預測模型雜訊w_t之共變異矩陣。在更新階段，濾波器利用對當前狀態的觀測值優化在預測階段獲得的預測值，以獲得一個更精確的新估計值：

$$K_t = P_t^- H^T (HP_t^- H^T + R)^{-1} \tag{17}$$

$$\hat{x}_t = \hat{x}_t^- + K_t(y_t - H\hat{x}_t^-) \tag{18}$$

$$P_t = (I - K_t H)P_t^- \tag{19}$$

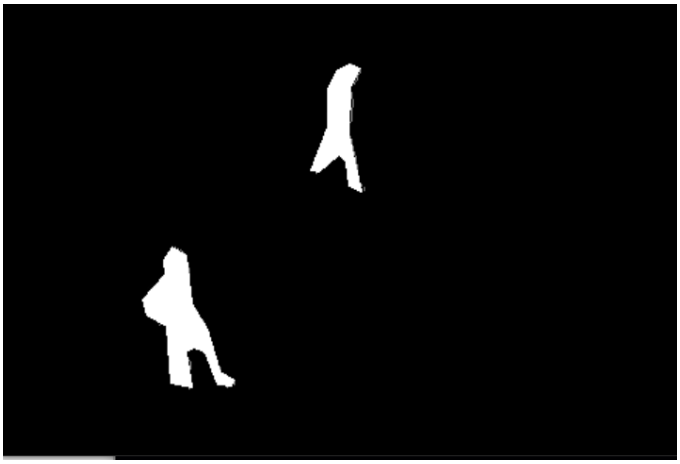
其中 R 為測量模型雜訊v_t之共變異矩陣。卡曼濾波器之運作架構如圖十五所示。卡曼濾波器在此專題研究中最主要的功能是當物體重疊交會、被遮擋、無法準確追蹤時，能夠運用卡曼濾波器預測其可能路徑、估算其可能位置，當物體再度出現於視窗中時，方便持續追蹤物體不遺失資料。



圖十五、 卡曼濾波器運作架構示意圖

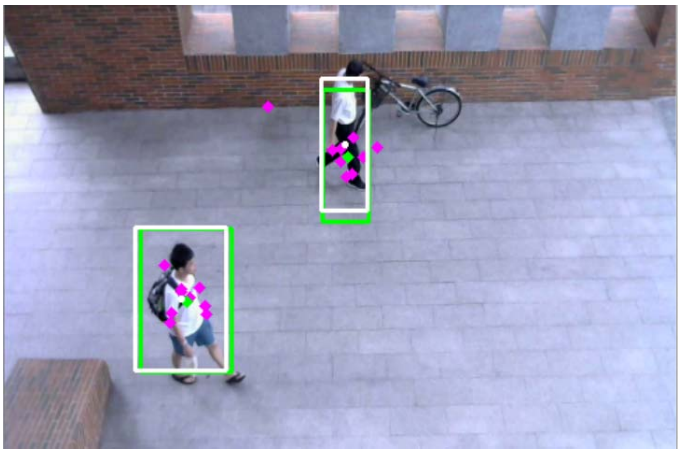
III. 成果展示

1. 建立高斯混合模型分離前景背景



圖十六、前景、背景

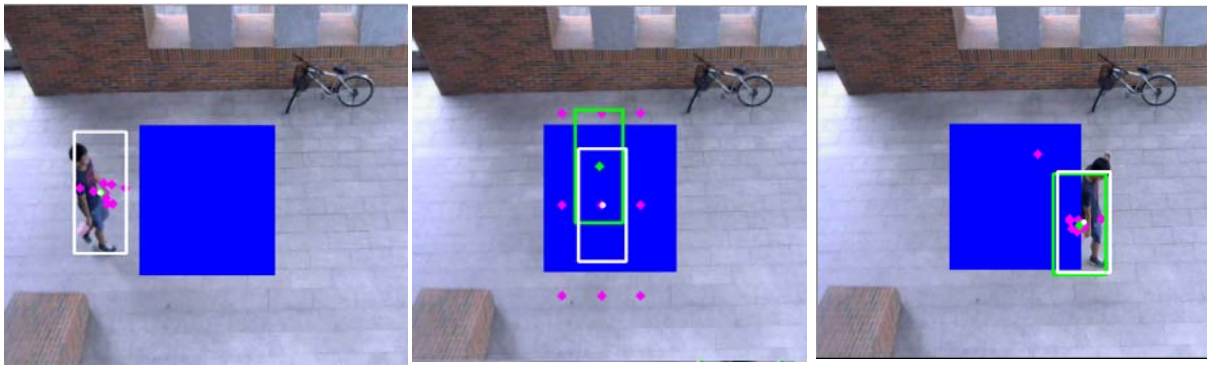
2. 建立粒子濾波器予以追蹤，並依據人物移動的速度定義撒點範圍



圖十七、人物追蹤(紫點為粒子點、綠框為粒子濾波器追蹤成果、白框為卡曼修正結果)

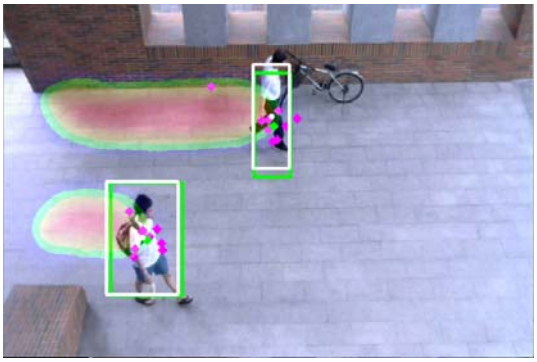
3. 當人物被遮蔽時，利用卡曼濾波器作預測，並將粒子點撒點擴大以期人物重新出現在畫面中能夠被精準追蹤

遮蔽前　　遮蔽狀態　　遮蔽後



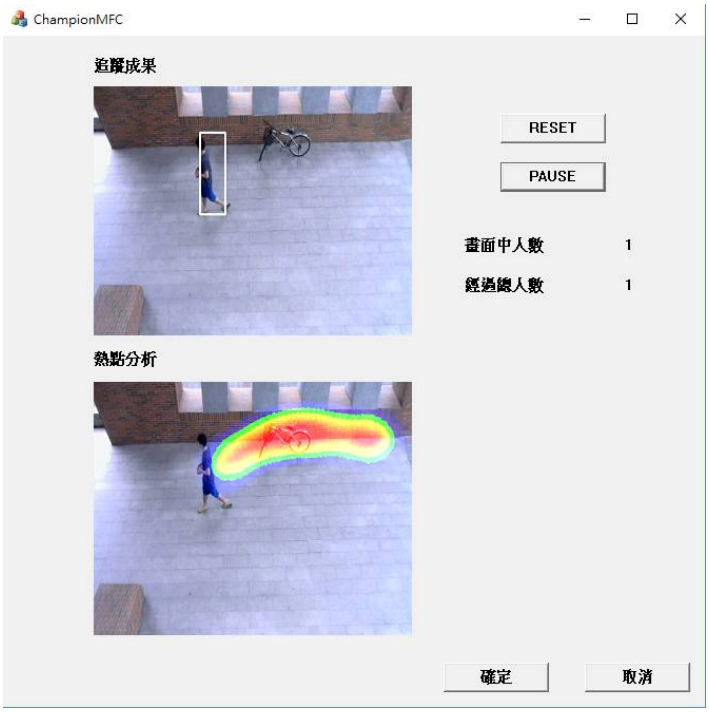
圖十八、遮蔽預測(藍色方框為假設障礙物)

4. 依據人物移動的軌跡作熱點分析，運用三種顏色(R、G、B)調配出區域熱點分布圖，使其有漸層。



圖十九、熱點分析

5. 將影像資訊整理至 MFC 使用者介面



圖二十、使用者介面

6. 為了增進運算速度及效能，利用 OPENMP 以及 THREAD 將程式以多執行緒處理，可以發現 CPU 使用率很高。

名稱	89% CPU	52% 記憶體	3% 磁碟	0% 網路
threadtest.exe (32 位元) (6)	84.2%	38.8 MB	0 MB/秒	0 Mbps

圖二十一、cpu 使用效能

IV. 結語

在本計畫中，我們設計了一個基於粒子濾波器之智慧影像監控系統。我們利用高斯混合模型建立背景，並利用背景相減之技術，找出畫面中之人物。當人物出現在影像監控的範圍中，即建立一粒子濾波器追蹤此新人物，若是被追蹤之人物離開了影像範圍，便終止及刪除其對應之粒子濾波器，以減少電腦運算量。此外，當遇到人物重疊或是被遮蔽之特殊情況，我們將使用卡曼濾波器預測人物走向，使追蹤更加精確。最後分析人物行經之軌跡，畫出熱點分析圖，並藉由使用者介面，將偵測出之重要資訊，例如：區域人流計數、賣場熱點分析等等，提供給此智慧型影像監控系統之使用者。

V. 參考資料

[1] Collins, R., A. Lipton, and T. Kanade, “A System for Video Surveillance and Monitoring,” *In Proc. American Nuclear Society (ANS) Eighth International Topical Meeting on Robotic and Remote Systems*, April. 1999.

[2] Y. Nam, S. Rho, and J. H. Park, “Intelligent video surveillance system: 3-tier context-aware surveillance system with metadata,” *Multimedia Tools and Applications*, vol. 57, no. 2, pp. 315-334, Mar. 2012.

[3] C. Stauffer and W.E.L Grimson, “Adaptive background mixture models for real-time tracking,” *Computer Vision and Pattern Recognition*, vol. 2, pp. 252-258, Jun.1999.

[4] P. KaewTraKulPong, R. Bowden, “An improved adaptive background mixture model for real-time tracking with shadow detection,” *Proceedings of the 2nd European Workshop on Advanced Video Based Surveillance Systems (AVBS '01)*, pp. 1-5, Sep. 2001.

[5] K. Nummiaro, E. Koller-Meier, and L. Van Gool, “Object Tracking with an Adaptive Color-based Particle Filter,” Pattern Recognition, Springer-Verlag, Berlin Heidelberg, pp.353-360, 2002.

[6] S. K. Weng, C. M. Kuo, and S.K. Tu, “Video object tracking using adaptive Kalman filter,” *Journal of Visual Communication and Image Representation*, vol.17, no. 6, pp. 1190-1208, Dec. 2006.

VI. 附錄

1-1、 完整 OpenCV 程式碼(.cpp)：

```
#include <math.h>
#include <iostream>
#include <ctype.h>
#include <opencv2/core/core.hpp>
#include <opencv2/opencv.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <opencv2/legacy/legacy.hpp>
#include <opencv2/imgproc/imgproc.hpp>
#include <opencv2/video/background_segm.hpp>
#include <windows.h>
#include <ctime>
#include <iomanip>
#include <vector>
#include <thread>
#include "tracker.h"
#include "function.h"
using namespace std;
using namespace cv;
using namespace Object_Tracking;

const float *ranges1[] = { hrange, srange };
//-------------------------------------------------------------------------
void hotspotfunc();

int objectnums = 0, Total = 0, t = 0;
int lastcenterx = 0;
int lastcentery = 0;
int size = 2;
int RER = 40;
float maxx = 0.0;
Mat hotpoint = Mat::zeros(H, W, CV_8UC3);
Mat hotpoint2(H, W, CV_32FC1, Scalar(0));
Mat frame, frame1, frame2, object, ftrack, hotspot;
vector<vector<Point> > contours;
```

```
vector<Vec4i> hierarchy;
vector<PARTICLEFILTER> objects;
vector<KalManTT> KFS;

Mat obstacle(160, 160, CV_8UC3, Scalar(255));

int main(int argc, char** argv)
{
	VideoCapture cap;

	if (!cap.open("Video16.mp4"))
	{
		cout << "byeybe" << endl;
		return -1;
	}
	const float p = 0.07;
	cout << (W * (1 - 2 * p)) << endl;
	cout << (H * (1 - 2 * p)) << endl;

	BackgroundSubtractorMOG2 background(16, 25, true);
	//background.nShadowDetection = 0;
	//background.fTau = 0.5;
	//int area = 0;
	//-------------------------------------------------------------------
	Mat element = getStructuringElement(MORPH_ELLIPSE, Size(2 * size + 1, 2 *
size + 1));

	for (;;){

		cap >> frame2;

		Mat imgROI = frame2(Rect(160, 160, 160,160 ));
		addWeighted(imgROI, 0, obstacle, 1,0, imgROI);

		if (frame2.empty())
		{
			cout << "byeybe" << endl;
			cvDestroyWindow("object4");
```

```
                cvDestroyWindow("object");
                cvDestroyWindow("resulyyrt");
                cvDestroyWindow("object3");
                break;
            }
            resize(frame2, frame1, Size( W, H), INTER_LINEAR);
            LARGE_INTEGER startTime, endTime, fre;
            double times;
            QueryPerformanceFrequency(&fre); //取得CPU頻率
            QueryPerformanceCounter(&startTime); //取得開機到現在經過幾個CPU
Cycle
//-------------------------------------------------------------------------------------------
            background(frame1, object, 0.005);

            if (t >= 40){
            thread hotspotThread(hotspotfunc);
            threshold(object, object, 170,255,0);

            morphologyEx(object, object, MORPH_CLOSE, element);
            morphologyEx(object, object, MORPH_OPEN, element);

            findContours(object, contours, hierarchy, CV_RETR_TREE,
CV_CHAIN_APPROX_SIMPLE);
            cvtColor(frame1, frame, CV_BGR2HSV);
            vector<Rect> boundRect(contours.size());
            vector<vector<Point> > contours_poly(contours.size());
            int newobject = 0;
            for (signed int i = 0; i < contours.size() ; i++)
            {
                approxPolyDP(Mat(contours[i]), contours_poly[i], 3, true);
                boundRect.at(i) = boundingRect(Mat(contours_poly[i]));
                //cout << boundRect[i].area() << endl;
                Point boundcenter(boundRect.at(i).x + (boundRect.at(i).width / 2),
boundRect.at(i).y + (boundRect.at(i).height / 2));
                drawContours(object, contours_poly, i, Scalar(255, 255, 255),
CV_FILLED);
                if (boundRect[i].area() > 3200 &&
                    ((boundcenter.x >(W * (1 - 2 * p)) && boundcenter.x <(W * (1 -
```

```
p)))
                    || (boundcenter.x <(W * 2 * p) && boundcenter.x >(W * p))
                    || (boundcenter.y >(H * (1 - 2 * p)) && boundcenter.y < (H * (1 -
p)))
                    || (boundcenter.y < (H * 2 * p) && boundcenter.y >(H * p))))
                {
                    drawContours(object, contours_poly, i, Scalar(255, 255, 255),
CV_FILLED);
                    lastcenterx = boundRect.at(i).x + (boundRect.at(i).width / 2);
                    lastcentery = boundRect.at(i).y + (boundRect.at(i).height / 2);
//-------------------------------------------------------------------------------------------
                        int check=0;
                        for (int j = 0; j < objectnums; j++)
                        {
                            float dist_x = sqrt(pow(lastcenterx -
(objects.at(j).result.x + objects.at(j).result.width/2), 2));
                            float dist_y = sqrt(pow(lastcentery -
(objects.at(j).result.y + objects.at(j).result.height/2), 2));
                            if (dist_x    >    objects.at(j).result.width/2 || dist_y >
objects.at(j).result.height/2)
                            {
                                check++;
                            }
                            else
                            {
                                Mat boundRect_hist;
                                Mat boundRect_img = Mat(frame,
boundRect.at(i));
                                calcHist(&boundRect_img, 1, channels,
Mask(boundRect.at(i)), boundRect_hist, 2, hist_size, ranges1);
                                normalize(boundRect_hist, boundRect_hist);
                                float similarity =
1-pow(compareHist(objects.at(j).targethist, boundRect_hist,
CV_COMP_BHATTACHARYYA),4);
                                //cout << "similarity = " << similarity << endl;
                                if (similarity > 0.6)
                                    break;
                                else
```

```
                                        check++;
                                }
                        }
                                if (check == objectnums)
                                {
                                        PARTICLEFILTER Object;
                                        newobject++;
                                        Object.targetRect = boundRect.at(i);
                                        Object.initial(Object.targetRect);
                                        Object.target(frame, Object.targetRect);
                                        objects.push_back(Object);

        //-------------------------------------------------------------
                                        KalManTT kfs;
                                        kfs.initial(lastcenterx, lastcentery);
                                        KFS.push_back(kfs);
                                        //cout << "newobject1!" << endl;
                                        //cout << Object.trackvalid << endl;
                                }
                        }
                }
                if (newobject > 0){
                        objectnums = objectnums + newobject;
                        Total = objectnums + newobject;
                }
#pragma omp parallel for
                for (int j = 0; j < objectnums; j++)
                {
                        objects.at(j).track(objects.at(j).targethist, frame, frame1,
objects.at(j).targetRect, KFS.at(j).dx, KFS.at(j).dy, KFS.at(j).statePt.x,
KFS.at(j).statePt.y);
                        KFS.at(j).prediction(objects.at(j).trackvalid, objects.at(j).center.x,
objects.at(j).center.y);
                        rectangle(frame1, objects.at(j).result, Scalar(0, 255, 0), 2, 8, 0);
                        //cout << KFS.at(j).statePt << "    objectn: " << objectnums << endl;
                        //system("pause");
                }
```

```
            hotspotThread.join();
            for (int j = 0; j < objectnums; j++)
            {
                KFS.at(j).update();
                rectangle(frame1, Rect(KFS.at(j).statePt.x - objects.at(j).HXY.x / 2,
KFS.at(j).statePt.y - objects.at(j).HXY.y / 2, objects.at(j).HXY.x,
objects.at(j).HXY.y), Scalar(255, 255, 255), 2, 8, 0);
                circle(frame1, KFS.at(j).statePt, 1, CV_RGB(255, 255, 255), 2);
                if (KFS.at(j).statePt.x > (W * (1 - 1 * p)) || KFS.at(j).statePt.x < (W * 1
* p)
                    || KFS.at(j).statePt.y >(H * (1 - 1 * p)) || KFS.at(j).statePt.y < (H *
1 * p))
                {
                    //------------------------------------------------
                    vector<PARTICLEFILTER>::iterator itortrack;
                    vector<KalManTT>::iterator itorTT;
                    //------------------------------------------------
                    itortrack = objects.begin() + j;
                    itorTT = KFS.begin() + j;
                    //------------------------------------------------
                    objects.erase(itortrack);
                    KFS.erase(itorTT);
                    //------------------------------------------------
                    objectnums--;
                }
            }
            }
            addWeighted(frame1, 1.0, hotpoint, 0.3, 0, hotspot);
            imshow ("object", object);
            imshow("object3", hotpoint2);
            imshow("hotspot", hotspot);
            //imshow("object4", hotpoint);
            imshow ("resulyyrt", frame1);

            QueryPerformanceCounter(&endTime); //取得開機到程式執行完成經過
幾個CPU Cycle
            times = ((double)endTime.QuadPart - (double)startTime.QuadPart) /
fre.QuadPart;
```

```
            cout << fixed << setprecision(20) << times << 's' << endl;
            t++;
            char c = cvWaitKey(20);
            if (c == 27){
                  break;
            }
      }
      cout << Total << endl;
      system("pause");
      cout << endl;
      return 0;
}


void hotspotfunc(){

      for (int j = 0; j < objectnums; j++)
      {
            for (int a = KFS.at(j).statePt.x - RER; a < KFS.at(j).statePt.x + RER; a++)
            {
                  for (int b = KFS.at(j).statePt.y - RER; b < KFS.at(j).statePt.y + RER;
b++)
                  {
                        if (KFS.at(j).statePt.y - RER >= 0 && KFS.at(j).statePt.y + RER
< H && KFS.at(j).statePt.x - RER >= 0 && KFS.at(j).statePt.x + RER < W)
                        {
                              if (sqrt(pow((a - KFS.at(j).statePt.x), 2) + pow((b -
KFS.at(j).statePt.y), 2)) <= RER)
                              {
                                    hotpoint2.at<float>(b, a) += exp(-sqrt(pow((a -
KFS.at(j).statePt.x), 2) + pow((b - KFS.at(j).statePt.y), 2)) / 20);
                                    if (hotpoint2.at<float>(b, a) >= maxx)
                                    {
                                          maxx = hotpoint2.at<float>(b, a);
                                    }
                              }
                              else
                                    hotpoint2.at<float>(b, a) += 0;
```

```
                    }
                }
            }

        }
        if (objectnums >= 1)
        {
#pragma omp parallel for
            for (int ii = 0; ii < W; ii++)
            {
                for (int jj = 0; jj < H; jj++)
                {
                    if (hotpoint2.at<float>(jj, ii) > 6 * maxx / 16)
                    {
                        hotpoint.at<Vec3b>(jj, ii)[2] =
(80/maxx)*hotpoint2.at<float>(jj, ii)+175;
                        hotpoint.at<Vec3b>(jj, ii)[0] = 0;
                        hotpoint.at<Vec3b>(jj, ii)[1] = (-510 /
maxx)*hotpoint2.at<float>(jj, ii) + 408;
                        if ((-510 / maxx)*hotpoint2.at<float>(jj, ii) + 408 < 0)
                            hotpoint.at<Vec3b>(jj, ii)[1] = 0;
                    }
                    else if (hotpoint2.at<float>(jj, ii) < 4 * maxx / 16)
                    {
                        hotpoint.at<Vec3b>(jj, ii)[2] = 0;
                        hotpoint.at<Vec3b>(jj, ii)[0] = (510 /
maxx)*hotpoint2.at<float>(jj, ii)-510;
                        hotpoint.at<Vec3b>(jj, ii)[1] = 0;
                    }
                    else
                    {
                        hotpoint.at<Vec3b>(jj, ii)[2] = 0;
                        hotpoint.at<Vec3b>(jj, ii)[0] = 0;
                        hotpoint.at<Vec3b>(jj, ii)[1] = 255 - 60 *
(hotpoint2.at<float>(jj, ii) / maxx);
                    }
                }
            }
```

```
    }
        }
```

1-2、 完整 OpenCV 程式碼(.cpp)

```
#include <math.h>
#include <iostream>
//#include <opencv2/core/core.hpp>
#include <opencv2/opencv.hpp>
//#include <vector>
//#include <opencv2/imgproc/imgproc.hpp>
using namespace std;
using namespace cv;

Mat Mask(Rect msk){
    Mat ellipsemask;
    ellipsemask = Mat::zeros(msk.height, msk.width, CV_8UC1);
    float a = 0, b = 0, c = 0;
    Point F1, F2, Center;
    if (msk.height > msk.width)
    {
        a = msk.height / 2;
        b = msk.width / 2;
        c = sqrt(a*a - b*b);
        F1 = Point(b, a + c);
        F2 = Point(b, a - c);
        Center = Point(b, a);
    }
    else
    {
        a = msk.width / 2;
        b = msk.height / 2;
        c = sqrt(a*a - b*b);
        F1 = Point(a + c, b);
        F2 = Point(a - c, b);
        Center = Point(a, b);
    }
```

```
    for (float i = 0; i < msk.width; i++)
    {
        for (float j = 0; j < msk.height; j++)
        {
            float distFF = sqrt((i - F1.x)*(i - F1.x) + (j - F1.y)*(j - F1.y)) + sqrt((i - F2.x)*(i - F2.x) + (j - F2.y)*(j - F2.y));
                if (distFF > 2 * a)
                {
                    ellipsemask.at<uchar>(j, i) = 0;
                }
                else
                {
                    //ellipsemask.at<uchar>(j, i) = 255 * (1 - 0.9*(((float)j - Center.y)*((float)j - Center.y) / Center.y*Center.y + ((float)i - Center.x)*((float)i - Center.x) / Center.x*Center.x));
                    ellipsemask.at<uchar>(j, i) = 255 * (1 - 0.9*(((j - Center.y)*(j - Center.y)) / (Center.y*Center.y) + ((i - Center.x)*(i - Center.x)) / (Center.x*Center.x)));
                }
        }
    }
    return ellipsemask;
}
```

1-3、完整 OpenCV 程式碼(.cpp)

```
#include <iostream>
#include <opencv2/opencv.hpp>
#include <opencv2/legacy/legacy.hpp>
#include <opencv2/imgproc/imgproc.hpp>
#include <opencv2/video/tracking.hpp>
#include <vector>
#include <iomanip>
#include "tracker.h"
#include "function.h"
using namespace Object_Tracking;
using namespace std;
```

```
using namespace cv;

const float *ranges[] = { hrange, srange };

int WR = W* 0.05, HR = H *0.05;

void PARTICLEFILTER::initial(Rect bound){
	CvMat lowerBound;
	CvMat upperBound;

	float maxxRange = bound.x + .5*bound.width + WR, maxyRange = bound.y
+ .5*bound.height + HR, maxHxRange = 1.1*bound.width, maxHyRange =
1.1*bound.height;
	float minxRange = bound.x + .5*bound.width - WR , minyRange = bound.y
+ .5*bound.height -HR , minHxRange = 0.9*bound.width, minHyRange =
0.9*bound.height;
	float minRange[] = { minxRange, minyRange, minHxRange, minHyRange };
	float maxRange[] = { maxxRange, maxyRange, maxHxRange, maxHyRange };
	cvInitMatHeader(&lowerBound, 4, 1, CV_32FC1, minRange);
	cvInitMatHeader(&upperBound, 4, 1, CV_32FC1, maxRange);
	cvConDensInitSampleSet(ConDens, &lowerBound, &upperBound);

	// 4*4 dynamMat including Hx & Hy
	ConDens->DynamMatr[0] = 1.0; ConDens->DynamMatr[1] = 0.0;
ConDens->DynamMatr[2] = 0.0; ConDens->DynamMatr[3] = 0.0;
	ConDens->DynamMatr[4] = 0.0; ConDens->DynamMatr[5] = 1.0;
ConDens->DynamMatr[6] = 0.0; ConDens->DynamMatr[7] = 0.0;
	ConDens->DynamMatr[8] = 0.0; ConDens->DynamMatr[9] = 0.0;
ConDens->DynamMatr[10] = 1.0; ConDens->DynamMatr[11] = 0.0;
	ConDens->DynamMatr[12] = 0.0; ConDens->DynamMatr[13] = 0.0;
ConDens->DynamMatr[14] = 0.0; ConDens->DynamMatr[15] = 1.0;

	trackvalid = true;
	cover = true;
}


void PARTICLEFILTER::target(Mat image,Rect select){
```

```
        target_img = image(select);
        imshow("mask", Mask(select));
        calcHist(&target_img, 1, channels, Mask(select), targethist, 2, hist_size, ranges);
        normalize(targethist, targethist);
}

void PARTICLEFILTER::track(Mat target_hist, Mat image, Mat image1, Rect select,
int dx, int dy,int KFSX,int KFSY){
        Rect rect;
        int X, Y, Hx, Hy;
        int n = 0;
        if (trackvalid)
        {
                for (int i = 0; i < SamplesNum; i++){

                        ConDens->flSamples[i][0] = ConDens->flSamples[i][0] + dx;
                        ConDens->flSamples[i][1] = ConDens->flSamples[i][1] + dy;

                        X = ConDens->flSamples[i][0];
                        Y = ConDens->flSamples[i][1];

                        if (ConDens->flSamples[i][2] > 1.1*select.width)
ConDens->flSamples[i][2] = 1.1*select.width;
                        else if (ConDens->flSamples[i][2]< 0.9*select.width)
ConDens->flSamples[i][2] = 0.9*select.width;
                        if (ConDens->flSamples[i][3] > 1.1*select.height)
ConDens->flSamples[i][3] = 1.1*select.height;
                        else if (ConDens->flSamples[i][3] < 0.9*select.height)
ConDens->flSamples[i][3] = 0.9*select.height;

                        Hx = ConDens->flSamples[i][2];
                        Hy = ConDens->flSamples[i][3];



                        rect = Rect(X - Hx / 2, Y - Hy / 2, Hx, Hy);
                        rect &= Rect(0, 0, image.cols, image.rows);
                        rect_img = Mat(image, rect);
```

```
                calcHist(&rect_img, 1, channels, Mask(rect), recthist, 2, hist_size,
ranges);
                normalize(recthist, recthist);
                ConDens->flConfidence[i] = exp(-Beta * compareHist(target_hist,
recthist, CV_COMP_BHATTACHARYYA));

                //cout << (i+1) << "," << ConDens->flConfidence[i] << endl;
                circle(image1, Point(X, Y), 2, Scalar(255, 0, 255), 4);
                if (ConDens->flConfidence[i] < exp(-Beta * 0.53))
                {
                    n++;
                }
            }
        }
        else
        {
            for (int i = -1; i < 2; i++){
                for (int j = -1; j < 2; j++)
                {
                    int I;
                    if (i == -1 && j == -1)
                        I = 0;
                    else if (i == -1 && j == 0)
                        I = 1;
                    else if (i == -1 && j == 1)
                        I = 2;
                    else if (i == 0 && j == 1)
                        I = 3;
                    else if (i == 1 && j == 1)
                        I = 4;
                    else if (i == 1 && j == 0)
                        I = 5;
                    else if (i == 1 && j == -1)
                        I = 6;
                    else if (i == 0 && j == -1)
                        I = 7;
                    else
                        I = 8;
```

```
                    ConDens->flSamples[I][0] = KFSX + i*0.8*HXY.x+dx;
                    ConDens->flSamples[I][1] = KFSY + j*0.8*HXY.y+dy;

                    X = ConDens->flSamples[I][0];
                    Y = ConDens->flSamples[I][1];

                    if (ConDens->flSamples[I][2] > 1.1*select.width)
ConDens->flSamples[I][2] = 1.1*select.width;
                    else if (ConDens->flSamples[i][2]< 0.9*select.width)
ConDens->flSamples[I][2] = 0.9*select.width;
                    if (ConDens->flSamples[I][3] > 1.1*select.height)
ConDens->flSamples[I][3] = 1.1*select.height;
                    else if (ConDens->flSamples[I][3] < 0.9*select.height)
ConDens->flSamples[I][3] = 0.9*select.height;

                    Hx = ConDens->flSamples[I][2];
                    Hy = ConDens->flSamples[I][3];



                    rect = Rect(X - Hx / 2, Y - Hy / 2, Hx, Hy);
                    rect &= Rect(0, 0, image.cols, image.rows);
                    rect_img = Mat(image, rect);
                    calcHist(&rect_img, 1, channels, Mask(rect), recthist, 2, hist_size,
ranges);
                    normalize(recthist, recthist);
                    ConDens->flConfidence[I] = exp(-Beta *
compareHist(target_hist, recthist, CV_COMP_BHATTACHARYYA));

                    //cout << (i+1) << "," << ConDens->flConfidence[i] << endl;
                    circle(image1, Point(X, Y), 2, Scalar(255, 0, 255), 4);
                    if (ConDens->flConfidence[I] < exp(-Beta * 0.53))
                    {
                        n++;
                    }
                }
            }
        }
```

```
        if (n == SamplesNum)
            trackvalid = false;
        else
            trackvalid = true;

        cvConDensUpdateByTime(ConDens);
        if (ConDens->State[2] > 1.1*select.width) ConDens->State[2] =
1.1*select.width;
        else if (ConDens->State[2]< 0.9*select.width)    ConDens->State[2] =
0.9*select.width;
        if (ConDens->State[3] > 1.1*select.height) ConDens->State[3] =
1.1*select.height;
        else if (ConDens->State[3]< 0.9*select.height)    ConDens->State[3] =
0.9*select.height;

        center.x = ConDens->State[0] ;
        center.y = ConDens->State[1] ;
        HXY.x = ConDens->State[2];
        HXY.y = ConDens->State[3];

        result = Rect(ConDens->State[0] - ConDens->State[2] / 2, ConDens->State[1] -
ConDens->State[3] / 2, ConDens->State[2], ConDens->State[3]);

        result &= Rect(0, 0, image.cols, image.rows);
        circle(image1, Point(ConDens->State[0], ConDens->State[1]), 2, Scalar(0, 255,
0), 3);
}

void KalManTT::initial(int xx, int yy){

        dx = 0;
        dy = 0;

        KF.init(KFDP, KFMP, KFCP);

        Mat processNoise(KFDP, 1, CV_32F);
        measurement = Mat_<float>(KFMP, 1);
```

```
	KF.statePost.at<float>(0) = xx;
	KF.statePost.at<float>(1) = yy;
	KF.statePost.at<float>(2) = 0;
	KF.statePost.at<float>(3) = 0;

	KF.transitionMatrix = *(Mat_<float>(4, 4) << 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0,
0, 0, 1);
	setIdentity(KF.measurementMatrix, Scalar::all(1));
	setIdentity(KF.processNoiseCov, Scalar::all(1e-5));
	setIdentity(KF.measurementNoiseCov, Scalar::all(1));
	setIdentity(KF.errorCovPost, Scalar::all(1));
}

void KalManTT::prediction(bool a,int X, int Y){

	Mat prediction = KF.predict();

	measurement.setTo(Scalar(0));

	Point predictPt(prediction.at<float>(0), prediction.at<float>(1));

	if (a)
	{
		measurement(0) = X;
		measurement(1) = Y;
	}
	else
	{
		measurement(0) = KF.statePre.at<float>(0) +KF.statePre.at<float>(2);
		measurement(1) = KF.statePre.at<float>(1) +KF.statePre.at<float>(3);
	}
}

void KalManTT::update(){
	Mat estimated = KF.correct(measurement);
	statePt.x = estimated.at<float>(0);
	statePt.y = estimated.at<float>(1);
```

```
        dx = KF.statePost.at<float>(2);
        dy = KF.statePost.at<float>(3);
}
```

2-1、完整 OpenCV 程式碼(.h)：

```
const int PFDP = 4;
const int PFMP = 4;
const int SamplesNum = 9;

using namespace std;
using namespace cv;

const int W = 480;
const int H = 320;

const int KFDP = 4;
const int KFMP = 2;
const int KFCP = 0;

const int Beta = 20;

static int hist_size[] = {24 , 24 };
static int channels[] = { 0, 1 };
static float hrange[] = { 0, 180 };
static float srange[] = { 80, 255 };
namespace Object_Tracking {

        class PARTICLEFILTER{
        public:
                CvConDensation* ConDens = cvCreateConDensation(PFDP, PFMP,
SamplesNum);
                Mat target_img, rect_img;
                Mat targethist, recthist;
                Point center;
```

```
            //vector<Point> Hcenter;
            void target(Mat image, Rect select);
            Rect targetRect, result;
            void track(Mat target_hist, Mat image, Mat image1, Rect select, int dx, int
dy , int KFSX ,int KFSY);
            void initial(Rect bound);
            Point HXY;

            bool trackvalid;
            bool cover;
      };

      class KalManTT{
      public:
            KalmanFilter KF;
            Mat processNoise;
            Mat_<float> measurement;
            Point statePt;
            //vector<Point> HstatePt;
            void initial(int xx, int yy);
            void prediction(bool a, int X, int Y);
            void update();
            float dx, dy ;
      };
}
```

2-2、完整 OpenCV 程式碼(.h)

```
using namespace std;
using namespace cv;

Mat Mask(Rect msk);
```