

國立交通大學

電信工程學系碩士班

碩士論文

雙攝影機追蹤系統

Dual Camera Tracking System

研究生：楊新華

Student : Sing-Wang Yeong

指導教授：張文鐘 博士

Advisor : Dr. Wen-Thong Chang

中華民國

96 年 8 月

雙攝影機追蹤系統
Dual Camera Tracking System

研究生：楊新華

Student : Sing-Wang Yeong

指導教授：張文鐘 博士

Advisor : Dr. Wen-Thong Chang

國立交通大學

電信工程學系碩士班



Submitted to Department of Communication Engineering

College of Electrical and Computer Engineering

National Chiao Tung University

in Partial Fulfillment of the Requirements

for the Degree of

Master of Science

in Communication Engineering

August 2007

Hsinchu, Taiwan, Republic of China

中華民國九十六年八月

雙攝影機追蹤系統


研究生：楊新華

指導教授：張文鐘

國立交通大學

電信工程學系碩士班

摘要

The logo of National Tsing Hua University is a circular emblem with a gear-like border. Inside the circle, there are the letters 'ES A' and the year '1896'.

爲了能追蹤移動物體並且紀錄特寫鏡頭做爲備查，我們實作一個雙攝影機追蹤系統。這個系統可以分成兩個階段。第一階段，目標追蹤子系統從視訊攝影機讀取視訊資料，然後擷取前景，偵測及追蹤物體。我們使用背景學習、變化偵測、變化分類和前景分割的方法來擷取前景。我們使用輪廓和物體追蹤技術來偵測及追蹤物體。第二階段，座標轉換子系統使用 *planar homography mapping* 的方法轉換物體在視訊攝影機的座標到 PTZ 的座標。爲了讓 PTZ 能追蹤移動物體並且給予特寫鏡頭，座標轉換子系統根據物體的座標及大小，自動地調整 PTZ 的參數。

Dual Camera Tracking System

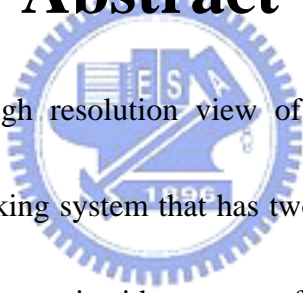
Student : Sing-Wang Yeong

Advisor : Wen-Thong Chang

Department of Communication Engineering

National Chiao Tung University

Abstract

The logo of National Chiao Tung University is a circular emblem with a gear-like border. Inside the circle, there is a stylized figure holding a torch, with the letters 'NCTU' and the year '1959' visible.

In order to capture a high resolution view of interested target region, we implement a dual camera tracking system that has two stages. In the first stage, the input to a target tracking subsystem is video streams from a single web camera. The subsystem analyzes the video content by extracting the foreground from the background, detecting and tracking the objects. The foreground is separated from the background by using background learning, change detection, change classification and foreground object segmentation methods. The objects are detected and tracked by using contouring and blobs tracking techniques. In the second stage, coordinate transformation subsystem transforms the coordinate of object on webcam coordinate system to PTZ coordinate system by using planar homography mapping method. In

order to capture a high resolution view of interested target region from PTZ, the coordinate transformation subsystem adjusts automatically the PTZ parameters based on the coordinate and size of the object on PTZ coordinate system.



I. Acknowledgement

I would like to express my gratitude to Professor Wen-Thong Chang for his supervision, encouragement, suggestions and trust throughout the development of this thesis. Without the support of my colleagues in Wireless Multimedia Communication Laboratory, this research would not be that much enjoyable for me.

Finally, my biggest gratitude is to my family for their endless love, support and trust in me. Without them I would never come up to this stage.



ii. Table of Contents

I.	Acknowledgement	iv
II.	Table of Contents	v
III.	List of Figures	vii
1.	Introduction.....	1
1.1.	Motivation.....	1
1.2.	System Overview	3
1.3.	Thesis Outline	4
2.	Theory of Dual Camera Tracking System	5
2.1.	Target Tracking System	5
2.1.1.	Block-based or Pixel-based Foreground/background Segmentation.....	7
2.1.2.	Simple Foreground/background Segmentation.....	21
2.1.3.	Blobs Tracking	23
2.2.	Coordinate Transformation System	27

2.2.1.	Coordinate System Changes and Rigid Transformations	27
2.2.2.	Intrinsic camera parameters	30
2.2.3.	Homography Derivation	34
2.2.4.	Homography Calculation	37
2.2.5.	Generation of Pan, Tilt and Zoom Tables	40
2.2.6.	Execution of Coordinate Transformation System.....	42
3.	Experimental Results	44
3.1.	Test application and system	44
3.2.	Result of Target Tracking System.....	46
3.3.	Module of Homography.....	49
3.4.	Result of Whole System.....	50
4.	Conclusion and Future Works.....	53
5.	References.....	54
6.	Appendix of Programming	56
6.1.	Simple Capturing Video from A Webcam.....	56

6.2.	Simple Capturing Video from A Webcam with OpenCV	65
6.3.	Simple Controlling PTZ Camera through RS232	70

III. List of Figures

Fig 1-1	System overview	3
Fig 2-1	Function block of Target Tracking System	6
Fig 2-2	Comparison of Foreground/background Segmentation methods.....	6
Fig 2-3	Function block of Block-based Foreground/background Segmentation.....	7
Fig 2-4	Characteristic of the moving background	9
Fig 2-5	Flow chart of Change Classification	17
Fig 2-6	Function block of Simple Foreground/background Segmentation.	21
Fig 2-7	Input, intermediate and output to the contour process.	23
Fig 2-8	Structure of blob.....	24
Fig 2-9	Relationship between status and frequency	25
Fig 2-10	Pure translation.....	28

Fig 2-11 Pure rotation	29
Fig 2-12 Rigid Transformation	29
Fig 2-13 Perspective projection model	30
Fig 2-14 Physical and normalized image coordinate systems.	31
Fig 2-15 The homography matrix, H induced by a plane Π	34
Fig 2-16 Plane Π with normal orthogonal vector $\vec{P_0N}$	35
Fig 2-17 Table of Pan/Tilt degree correspondent to coordinate of the frame.	40
Fig 2-18 Table of Zoom degree correspondent to pixels of the object.	41
Fig 2-19 Lookup Table of Zoom Ratio	42
Fig 2-20 Function block of Coordinate Transformation System	43
Fig 3-1 Camera Set-up	44
Fig 3-2 Software and hardware system architecture block diagram.....	45
Fig 3-3 Foreground and background segmentation (part I- moving object enters the scene).....	46

Fig 3-4 Foreground and background segmentation (part II- moving object stays temporary) in the scene).....	47
Fig 3-5 Foreground and background segmentation (part III- moving object leaves the scene)	48
Fig 3-6 Tracking the moving objects	49
Fig 3-7 Eight pairs of point selected manually	50
Fig 3-8 Homography projected results	50
Fig 3-9 Single object tracking with close-up	51
Fig 3-10 Multiple objects tracking with close-up	52
Fig 6-1 The block diagram of video capturing from webcam by using DirectShow	56
Fig 6-2 The block diagram of video capturing from webcam by using DirectShow and OpenCV	65

1.Introduction

1.1. Motivation

Traditional video surveillance system is labor intensive, low picture resolution and usually not effective. [Ref- 1] and [Ref- 2] provide good review of latest techniques of video surveillance system. We can classify those techniques roughly into:

Background Modeling. Running average is an easy background modeling method. Kalman-based background updating method [Ref- 1] is more advanced and provides better quality. Collins proposed a multilayered background model [

Ref- 3]. Stauffer and Grimson [Ref- 4] modeled the recent history of each pixel as a mixture of k Gaussian distributions.

Change Detection [Ref- 1]. The simple difference method computes for each time instant t the absolute difference between the pixel intensities of the input image and background image. The simple difference method is the simplest and fastest, but it is very sensitive to noise and illumination changes. In order to overcome the problem of noise sensitivity, the derivative model method considers $n \times n$ pixel regions in the two input images and computes a likelihood ratio L_{ij} by using the means and the variances of the two regions. The output binary image is obtained as

$$B(x, y) = \begin{cases} 0 & \text{if } L_{ij} < L_{Th} \\ 1 & \text{otherwise} \end{cases}$$

Blob Tracking. The blob tracking technique provides frame-by-frame tracking of the blob position and size. The connected-component tracker provides reliable and fast tracking results when there is no overlap of two blobs. A Kalman filter [Ref- 5] is used to predict the position of the blob in the next frame, thus implying that overlap will occur in the next frame. If overlap is to occur, the particle filter-based tracker [Ref- 6] is used.

We propose a system that captures a high resolution view of interested target region. The proposed system consists of target tracking system, followed by a coordinate transformation system. Target tracking system includes background learning, change detection, change classification, foreground object segmentation, contouring and blobs tracking techniques. Coordinate transformation system includes homography and automatic adjustment of camera parameters techniques. We propose three novel techniques which are foreground object detection, blob tracking and automatic adjustment of camera parameters. Other techniques are referred from books [Ref- 7, Ref- 8, Ref- 9] and papers.

1.2. System Overview

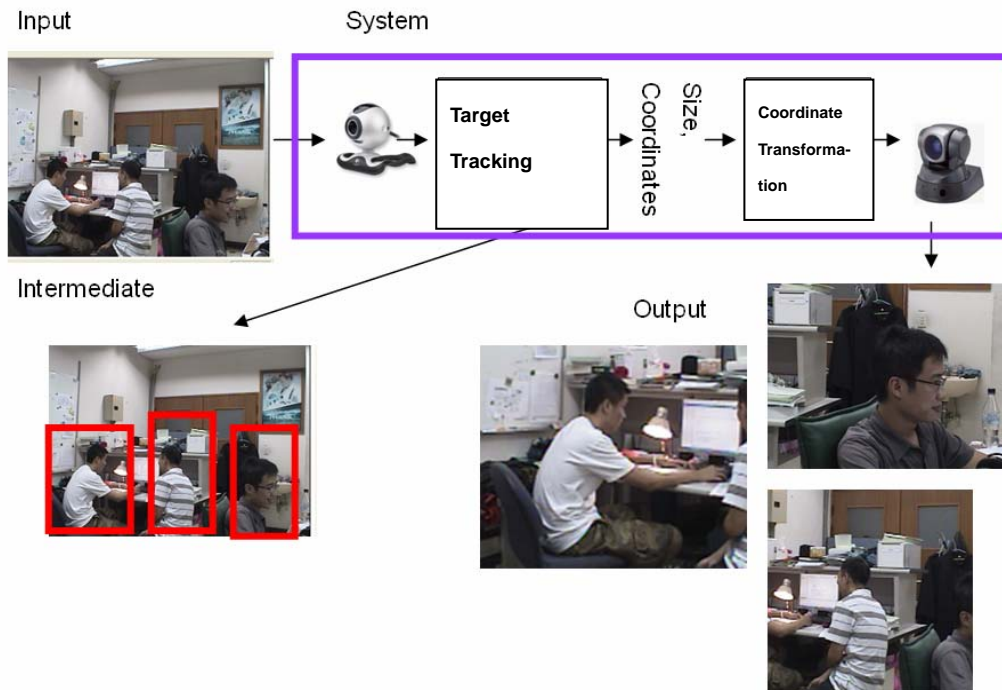


Fig 1-1 System overview



Fig 1-1 describes an overview of the proposed system that consists of (1) target tracking system and (2) coordinate transformation system. The input that will be processed by target tracking system to our main system is video streams from the webcam. The target tracking system analyzes the video content by separating the foreground from the background, detecting and tracking the objects, and sending the coordinate and size of each object to coordinate transformation system. The coordinate transformation system will give a close-up of each object based on size and coordinate of the rectangle, and PTZ lookup table.

1.3. Thesis Outline

This thesis is organized as follows. We explain the theory of dual camera tracking system in chapter 2. Results from our method are shown in chapter 3. We conclude with a summary of our findings and future research opportunities in chapter 4.



2.Theory of Dual Camera

Tracking System

In this chapter, we introduce the dual camera tracking system including target tracking system and coordinate transformation system. Section 2.1 describes the system that analyzes the video content by separating the foreground from the background, detecting and tracking the objects, and generating the coordinate of each object. The coordinate transformation system which will give a close-up of each blob based on size and coordinate of the rectangle, and PTZ lookup table is introduced in section 2.2. The techniques used in section 2.1 and 2.2 are Digital Image Processing and Computer Vision respectively.

2.1. Target Tracking System

The block diagram of the proposed algorithm for target tracking system is shown in Fig 2-1. Block-based or Pixel-based Foreground/background Segmentation and Simple Foreground/background Segmentation are described elaborately in section 2.1.1 and 2.1.2 respectively. Section 2.1.3 shows the blocks tracking method. Fig 2-2 shows the comparison of simple, block-based and pixel-based foreground/background

segmentation method for quality, speed and complexity.

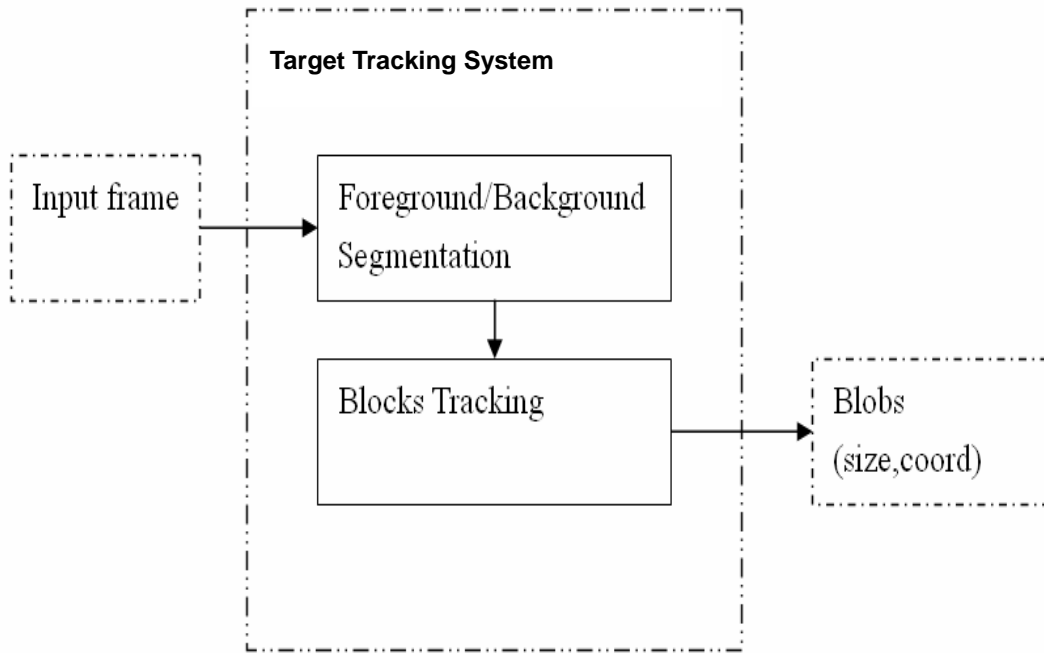


Fig 2-1 Function block of Target Tracking System

Method \	Quality	Complexity	Speed
Simple	Poor(1)	Low(3)	Fastest(3)
Block-based	Good(2)	High(1)	Fast(2)
Pixel-based	Best(3)	Medium(2)	Slow(1)

Fig 2-2 Comparison of Foreground/background Segmentation methods

2.1.1. Block-based or Pixel-based

Foreground/background Segmentation

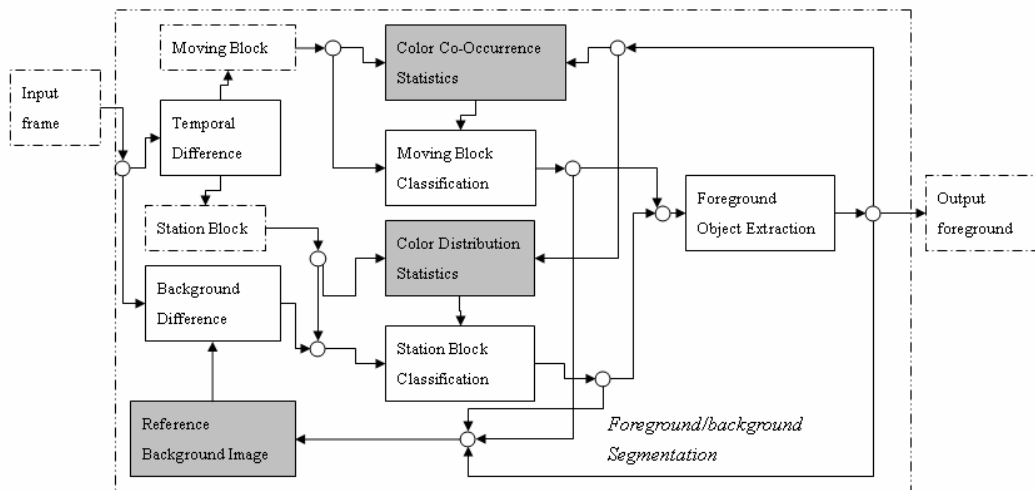


Fig 2-3 Function block of Block-based Foreground/background Segmentation

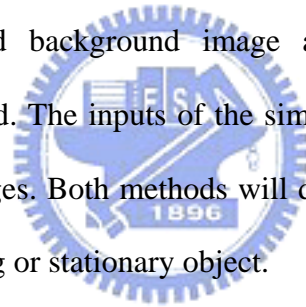
It consists of four parts: change detection, change classification, foreground object segmentation, and background learning and maintenance. The input and output to foreground/background segmentation are image and binary foreground image respectively. The block diagram of the proposed algorithm is shown in Fig 2-3. The light blocks from left to right correspond to the first three steps, and the gray blocks for the adaptive background learning. In the first step, the current image and background image are the inputs of the simple background difference method. This method will assign whether the block (pixel) of the current image is stationary object or moving object. The inputs to the simple temporal difference method are current and previous images. This method will assign whether the block (pixel) of the current image is moving object or stationary object. In the second step, the blocks (pixels)

associated with stationary or moving objects are classified as foreground or background based on the learned statistics by using the Bayes decision rule. Foreground objects are segmented by using morphological operation in the third step. In the fourth step, a reference background image is maintained to make the background difference accurate and adaptive to the changing background. Meanwhile, features statistics are updated in both gradual and “once-off” condition.

Bayes Classification of Background and Foreground

The background can consist of both stationary and moving objects. Meanwhile, the background might be undergoing two types of change over the time. There are gradual changes and sudden “once-off” changes.

The current image and background image are the inputs of the simple background difference method. The inputs of the simple temporal difference method are current and previous images. Both methods will decide whether the block (pixel) of the current image is moving or stationary object.



Let v_t be a discrete value feature vector extracted from an image sequence at the block (pixel) s and time instant t .

In the case of time difference, we consider the situation of the moving background object, if the color difference between a block (pixel) from current and previous image in the same place, $\|v_t - v_{t-1}\|$ is large. In the same time, the color difference between a block (pixel) from next and current image in the same place, $\|v_{t+1} - v_t\|$ is quite similar with $\|v_t - v_{t-1}\|$.

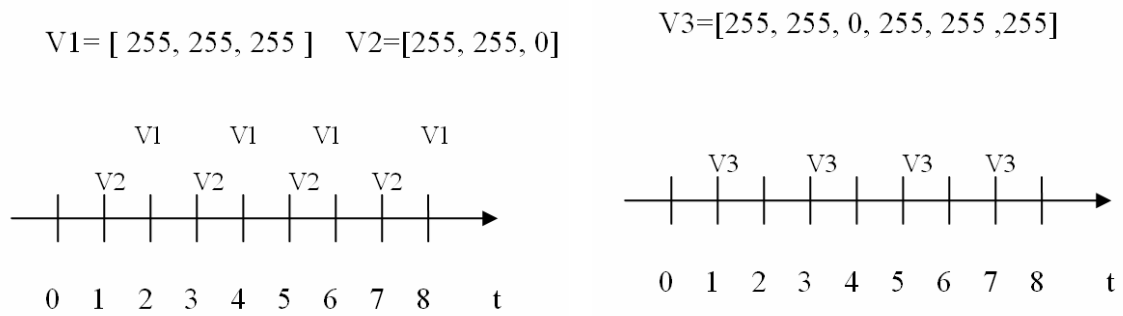


Fig 2-4 Characteristic of the moving background

Fig 2-4 shows the characteristic of the moving background. The feature vector $V1$ is not the same as $V2$. But $V1$ and $V2$ appear alternatively in the moving background. So, the joint simultaneous appearance of $V1$ and $V2$ can be used to identify whether there is a moving background or not. For this the co-occurrence feature vector $V3$ is formed by combining the adjacent-time feature vector $V2$ and $V1$. The co-occurrence feature vector indicates the information of two adjacent feature vectors. As shown in Fig 2-4, the co-occurrence feature vector $V3$ repeat at time 1, 3, 5 and 7.

Based on above observation, the feature vectors, i.e., v_t is replaced by $cc_t = [r_{t-1} \ g_{t-1} \ b_{t-1} \ r_t \ g_t \ b_t]^T$. The above feature vector is defined as color co-occurrences of the inter-frame feature vector.

To classify a block (pixel) as background or foreground, we design three feature distributions as below:

- (1) The frequency of block (pixel) that is labeled as background or foreground. This probability can be seen as the a priori probabilities of a block (pixel) s to belong to background $P(b|s)$ and foreground $P(f|s)$ respectively.

(2) The frequency of the feature vector of a block (pixel). This probability $P(v_t | s)$ indicate the possibility that a particular feature appear.

(3) When the block (pixel) is background or foreground, the frequency of each feature vector of the block (pixel) can be modeled as the conditional probability $P(v_t | b, s)$ and $P(v_t | f, s)$ respectively.

By observing features vectors of the block (pixel), if the probability of background is larger than the probability of foreground then the block (pixel) is background. In other words, the posterior probability $P(b | v_t, s)$ is larger than the posterior probability $P(f | v_t, s)$ then the block (pixel) is background. The posterior probability can be calculated through above three probabilities. Now, we show the relationship between above three probabilities and posterior probability.

Using Bayes rule, $P(C | v_t, s)$ is a probability of event C, given the occurrence of matched features vectors. It follows that the posterior probability of v_t from the background b or foreground f is

$$P(C | v_t, s) = \frac{P(v_t | C, s)P(C | s)}{P(v_t | s)} \quad \text{Eq. 2.1.1-1}$$

where $C = b$ or f .

Using the Bayes decision rule, the block (pixel) is classified as background if the feature vector satisfies

$$P(b | v_t, s) > P(f | v_t, s) \quad \text{Eq. 2.1.1-2}$$

Substituting Eq. 2.1.1-1 into Eq. 2.1.1-2

$$P(v_t | b, s)P(b | s) > P(v_t | f, s)P(f | s) \quad \text{Eq. 2.1.1-2a}$$

The feature vectors associated the block (pixel) s are either from background or from foreground objects, it follows

$$P(v_t | s) = P(v_t | b, s).P(b | s) + P(v_t | f, s).P(f | s) \quad \text{Eq. 2.1.1-3}$$

Rearrange Eq. 2.1.1-3

$$P(v_t | f, s).P(f | s) = P(v_t | s) - P(v_t | b, s).P(b | s) \quad \text{Eq. 2.1.1-3a}$$

Substituting Eq. 2.1.1-3a into Eq. 2.1.1-2a

$$P(v_t | b, s).P(b | s) > P(v_t | s) - P(v_t | b, s).P(b | s)$$

$$2P(v_t | b, s).P(b | s) > P(v_t | s) \quad \text{Eq. 2.1.1-4}$$

This shows that by learning the conditional probability $P(v_t | b, s)$, a prior probability $P(b | s)$ and the probability $P(v_t | s)$ in advance, we may classify a feature v_t as either associated with foreground or with background.

The above case discusses the situation when time difference is large. When the time difference is small, there are two possible situations. If the color difference between a block (pixel) from current and background image in the same place is large, while the color difference between a block (pixel) from current and previous image in the same place is small, this block is needed to be inspected. If both the time difference and the background difference are small the block is surely classified as background.

If the color difference between a block (pixel) from current and background image in the same place is large, while the color difference between a block (pixel) from current and previous image in the same place is small, each feature vector is the same. No adjacent time feature vector is needed. Based on above observation, the

stationary feature vectors, i.e., v_t is replaced by $c_t = [r_t \ g_t \ b_t]^T$. The stationary feature vector is defined as color feature vector. The classification method for stationary object is similar as classification for moving object.

The $P(v_t | s)$ and $P(v_t | b, s)$ could be represented by the histograms of features vectors over the entire feature space because they are unknown in a general cases. For n dimensional feature vector with L quantization levels, the histogram for $P(v_t | s)$ and $P(v_t | b, s)$ contains L^n bins. If L or n is large, a good approximation is desirable to make the computation and storage efficiency.

If the selected features are effective to represent background, at a block (pixel) s, the feature vectors from foreground object would distribute widely in the feature space, while the features vectors from the background would concentrate in a very small subspace of the feature histogram.

Let $P(v_t^i | b, s)$, $i = 1, \dots, N$ be the first N bins from the histogram sorted according to the descendent order of $P(v_t | b, s)$, i.e., $P(v_t^i | b, s) \geq P(v_t^{i+1} | b, s)$. For giving percentage values M_1 and M_2 , i.e., $M_1 > M_2$.

$$\sum_{i=1}^{N_1} P(v_t^i | b, s) > M_1 \quad \text{and} \quad \sum_{i=1}^{N_1} P(v_t^i | f, s) < M_2 \quad \text{Eq. 2.1.1-5}$$

There exists a small integer N_1 such that the above conditions are satisfied if selected features are effective to represent background, at a block (pixel) s.

For each type of feature vectors, a table of feature statistics, denoted as $S_{v_t}^{s,t,i}$, $i = 1, \dots, N_2$ ($N_2 > N_1$), is maintained at block (pixel) s and time t to record the statistics for N_2 most significant values. Each element in the table consists of three components,

i.e.,

$$\begin{aligned}
 p_v^{t,i} &= P(v_t^i | s) \\
 S_{v_t}^{s,t,i} &= \{p_{vb}^{t,i} = P(v_t^i | b, s)\} \\
 v_t^i &= [a_1^i, \dots, a_n^i]^T
 \end{aligned}
 \tag{Eq. 2.1.1-6}$$

The elements in the list are sorted according to the descendent order of $p_v^{t,i}$. The probabilities in Eq. 2.1.1-6 are initially set to zero. The vectors in Eq. 2.1.1-6 are also set as zero vectors initially.

To make the computation and storage efficiency, $L=64$ quantization levels in each color component are used for color vector. Meanwhile, $N_1=30$ and $N_2=50$ are selected. Obviously, 256^3 bins $\gg 64^3$ bins $\gg N_2$ bins $> N_1$ bins. For the feature vectors of color co-occurrences, $L=32$ with $N_1=50$ and $N_2=80$ are chosen. Similarly, 256^6 bins $\gg 32^6$ bins $\gg N_2$ bins $> N_1$ bins.

Algorithm Description

In the first step, **change detection**, simple background and temporal differencing filter out blocks (pixels) of insignificant changes. Let $I(pixel, t) = \{I_c(pixel, t)\}$ be the input color image and $B(pixel, t) = \{B_c(pixel, t)\}$ be the reference background image maintained by the system at time t , and $c \in \{r, g, b\}$ represents a color component. A simple picture differencing is done for each color component with adaptive thresholding, using the method described in [Ref- 10]. Let the absolute picture difference of each color component at time t as $D_c(pixel, t)$. The histogram of the absolute picture difference of each color component with gray levels in the range $[0, 255]$ is a discrete function $His_c(r_k, t) = n_{k,c,t}$, where r_k is the k th gray level and $n_{k,c,t}$ is the number of pixels in the image difference of each color component having

gray level r_k at time t . The normalized histogram of each color component is given by $P_c(r_k, t) = n_{k,c,t} / n$, where n is the total number pixels in the difference image. The relative variance of each color component $V_{r,c}$ is the ratio of the sample variance to the sample mean of the normalized histogram of each color component. The k th gray level of relative variance of each color component $V_{r,c}(r_k)$ is calculated by using normalized histogram of each color component in the range $[r_k, 255]$. The threshold of each color component is obtain as

$$Thres_{c,t} = \left\{ r_k \mid \max_{0 \leq r_k \leq 254} V_{r,c}(r_k) \right\}$$

For pixel level,

the background difference of each color component is obtain as

$$F_{bd,c}(pixel, t) = \begin{cases} 1, & |B_c(pixel, t) - I_c(pixel, t)| > Thres_{bd,c,t} \\ 0, & otherwise \end{cases}$$

Background difference $F_{bd}(pixel, t)$ is generated by combining the three components as below

$$F_{bd}(pixel, t) = F_{bd,r}(pixel, t) \text{ OR } F_{bd,g}(pixel, t) \text{ OR } F_{bd,b}(pixel, t)$$

If $F_{bd}(pixel, t) = 0$ is detected, the pixel is classified as a station pixel.

The temporal difference of each color component is obtain as

$$F_{td,c}(pixel, t) = \begin{cases} 1, & |I_c(pixel, t-1) - I_c(pixel, t)| > Thres_{td,c,t} \\ 0, & otherwise \end{cases}$$

Temporal difference $F_{td}(pixel, t)$ is generated by combining the three components as below

$$F_{td}(pixel,t) = F_{td,r}(pixel,t) \text{ OR } F_{td,g}(pixel,t) \text{ OR } F_{td,b}(pixel,t)$$

If $F_{td}(pixel,t) = 1$ is detected, the pixel is classified as a motion pixel. Otherwise, it is a station pixel.

For block level,

the block-based temporal difference is obtain as

$$F_{td}(block,t) = \begin{cases} 1, & Np > BS/2 \\ 0, & otherwise \end{cases}$$

where Np is the number of motion pixel of temporal difference for each block and BS is block size.

If $F_{td}(block,t) = 1$ is detected, the block is classified as a motion block. Otherwise, it is a station block.

The block-based background difference is obtain as

$$F_{bd}(block,t) = \begin{cases} 1, & Np > BS/2 \\ 0, & otherwise \end{cases}$$

where Np is the number of motion pixel of background difference for each block and BS is block size.

If $F_{bd}(block,t) = 0$ is detected, the block is classified as a station block.

In the second step, **change classification**, the motion and station block (pixel) are further classified as foreground or background separately. Fig 2-5 shows the flow chart of the change classification.

If $F_{bd}(s,t) = 0$ and $F_{td}(s,t) = 0$, it assigns s as background. If $F_{td}(s,t) = 1$ and

moving tables (color co-occurrence statistics) have been trained, it uses Eq. 2.1.1-7 and Eq. 2.1.1-4 to determine whether s is foreground or background. If $F_{id}(s,t) = 1$ and moving tables (color co-occurrence statistics) have not been trained, it assigns s as foreground. If $F_{bd}(s,t) = 1$, $F_{id}(s,t) = 0$ and stationary tables (color statistics) have been trained, it uses Eq. 2.1.1-7 and Eq. 2.1.1-4 to determine whether s is foreground or background. If $F_{bd}(s,t) = 1$, $F_{id}(s,t) = 0$ and stationary tables (color statistics) have not been trained, it assigns s as background.

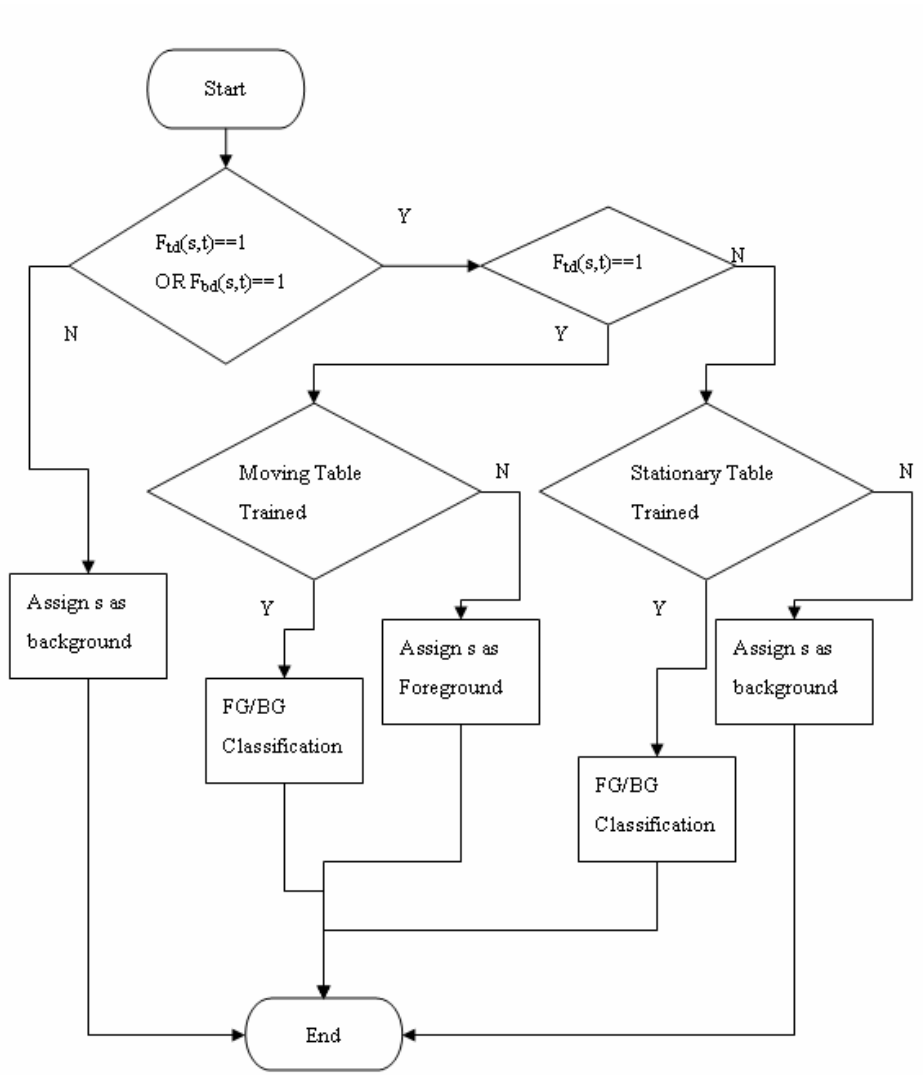


Fig 2-5 Flow chart of Change Classification

The probabilities from Eq. 2.1.1-4 are obtained as

$$\begin{cases} P(b | s) = p_b^{s,t} \\ P(v_t | b, s) = \sum_{j \in M(v_t)} P_{vb}^{s,t,j} \\ P(v_t | s) = \sum_{j \in M(v_t)} P_v^{s,t,j} \end{cases} \quad \text{Eq.2.1.1- 7}$$

where the matched feature set in $S_{v_t}^{s,t,i}$ is defined as

$$M(v_t) = \{k : \forall m \in \{1, \dots, n\}, |a_m^k - a_m^k| \leq \delta\} \quad \text{Eq. 2.1.1-8}$$

where $\delta = \frac{256 * dis}{L}$, L is quantization level and dis is maximum distance between each component of two matched feature vectors.

If no element in the table $S_{v_t}^{s,t,i}$ matches v_t , both $P(v_t | s)$ and $P(v_t | b, s)$ are set 0.

For block-based method, the feature vector, v_t of the block is calculated as mean of pixels' feature vector in the block for each component. Substituting the Eq. 2.1.1-7 into Eq. 2.1.1-4. If the inequality of Eq. 2.1.1.4 is true, then the block (pixel) is classified as foreground, otherwise as background.

In the stage of **foreground object segmentation**, a morphological operation (a pair of dilation and erosion) is applied to remove the scattered error points and connect the foreground points.

Let $Foreground(pixel, t) = Foreground(x, y, t)$. Use of erosion is for eliminating outer object noise from a binary image. Outer object noise free binary image is obtained as

$$Er(x, y, t) = \min_{((x', y') \text{ in element})} Foreground(x + x', y + y', t)$$

where element can be rectangular, cross and ellipse shape structuring element, and $Er(x, y, t)$ is outer object noise free binary image.

The inner object noise can be cancelled by using dilation process. Noise free binary image is computed as

$$Di(x, y, t) = \max_{((x', y') \text{ in element})} Er(x + x', y + y', t)$$

where $Di(x, y, t)$ is the segmented foreground objects and $O(pixel, t) = Di(x, y, t)$.

Finally, the **background maintenance** includes two parts, updating the tables of feature statistics and a reference background image. Two tables of color and color co-occurrence statistics are maintained at each block (pixel). If $F_{td}(s, t) = 1$, then it updates color co-occurrence statistics (motion tables). Otherwise, it updates color statistics (stationary tables). Two different updating strategies are proposed to adapt them to both gradual and “once-off” background changes.

For updating table to **gradual background changes**, the statistics table is gradually updated by

$$\begin{cases} p_v^{s,t+1,i} = (1 - \alpha_2) p_v^{s,t,i} + \alpha_2 M_v^{s,t,i} \\ p_b^{s,t+1,i} = (1 - \alpha_2) p_b^{s,t,i} + \alpha_2 M_b^{s,t,i} \\ p_{vb}^{s,t+1,i} = (1 - \alpha_2) p_{vb}^{s,t,i} + \alpha_2 (M_v^{s,t,i} \wedge M_b^{s,t,i}) \end{cases} \quad \text{Eq. 2.1.1-9}$$

where $i = 1, \dots, N_2$ and α_2 is the learning rate which controls the speed of feature learning. The Boolean values for matching labels are generated as follows. $M_v^{s,t,i} = 1$ when v_i^i of $S_v^{s,t,i}$ in Eq. 2.1.1-6 matches v_i best and $M_v^{s,t,i} = 0$ for the others. The probability for the matched feature $p_v^{s,t+1,i}$ is increased due to best matching features vectors ($M_v^{s,t,i} = 1$). The probability for the un-matched features $p_v^{s,t+1,i}$ is slightly

decreased due to not best matching features vectors ($M_v^{s,t,i} = 0$). $M_b^{s,t} = 1$ when s is labeled as the background at time t, otherwise, $M_b^{s,t} = 0$. The probability $p_b^{s,t+1}$ is increased due to s is a background ($M_b^{s,t} = 1$) and decreased due to s is a foreground ($M_b^{s,t} = 0$). Similarly, $p_{vb}^{s,t+1,i}$ is increased due to s is a background ($M_b^{s,t} = 1$) and best matching features vectors ($M_v^{s,t,i} = 1$). $p_{vb}^{s,t+1,i}$ is decreased due to other conditions. If the s is labeled as a foreground at time t, $p_b^{s,t+1}$ and $p_{vb}^{s,t+1,i}$ are slightly decreased with $M_b^{s,t} = 0$. However, for the matched element in $S_{v_t}^{s,t+1,i}$, $p_v^{s,t+1,i}$ is increased.

If there is no match between v_t and the elements in the table $S_{v_t}^{s,t,i}$, the N_2 th element in the table is substituted by a new feature vector

$$p_v^{s,t+1,N_2} = \alpha_2, \quad p_{vb}^{s,t+1,N_2} = \alpha_2, \quad v_t^{N_2} = v_t \quad \text{Eq. 2.1.1-10}$$

The updated elements in the table $S_{v_t}^{s,t+1,i}$ are re-sorted on a descendent order for $p_v^{s,t+1,i}$.

From Eq. 2.1.1-5 and Eq. 2.1.1-3, “**once-off background changes**” at block (pixel) s is detected if

$$P(f | s) \sum_{i=1}^{N_1} P(v_t^i | f, s) > T \quad \text{Eq. 2.1.1-11}$$

where T is a percentage value which determines when the new features can be recognized as new background appearance.

From Eq. 2.1.1-3, Eq. 2.1.1-11 becomes

$$\sum_{i=1}^{N_1} P(f|s)P(v_i^i|f,s) > T$$

$$\sum_{i=1}^{N_1} [P(v_i^i|s) - P(b|s)P(v_i^i|b,s)] > T$$

$$\sum_{i=1}^{N_1} P(v_i^i|s) - P(b|s) \sum_{i=1}^{N_1} P(v_i^i|b,s) > T \quad \text{Eq. 2.1.1-11a}$$

From Eq. 2.1.1-6, Eq. 2.1.1-11a becomes

$$\sum_{i=1}^{N_1} p_v^{s,t,i} - p_b^{s,t} \sum_{i=1}^{N_1} p_{vb}^{s,t,i} > T \quad \text{Eq. 2.1.1-12}$$

If Eq.2.1.1-12 is true, then the statistics table is adjusted as follows

$$\begin{cases} p_b^{s,t+1} = 1 - p_b^{s,t} \\ p_{vb}^{s,t+1,i} = (p_v^{s,t,i} - p_b^{s,t} \cdot p_{vb}^{s,t,i}) / p_b^{s,t+1} \end{cases} \quad \text{Eq. 2.1.1-13}$$

In order make the background difference accurate, a **reference background** image is also maintained at each time step.

If $F_{td}(s,t) = 1$ or $F_{bd}(s,t) = 1$ and $O(pixel,t) = 0$, it represents a background change is detected. The reference background image is updated as

$$B_c(pixel,t+1) = I_c(pixel,t), \text{ for } c = r, g, b \quad \text{Eq. 2.1.1-14}$$

If $F_{td}(s,t) = 0$ and $F_{bd}(s,t) = 0$ and $O(pixel,t) = 0$, it represents an insignificant change. The reference background image is updated as

$$B_c(pixel,t+1) = \alpha_1 I_c(pixel,t) + (1 - \alpha_1) B_c(pixel,t) \quad \text{Eq. 2.1.1-15}$$

where α_1 is update rate which controls the speed of background updating.

2.1.2. Simple Foreground/background

Segmentation

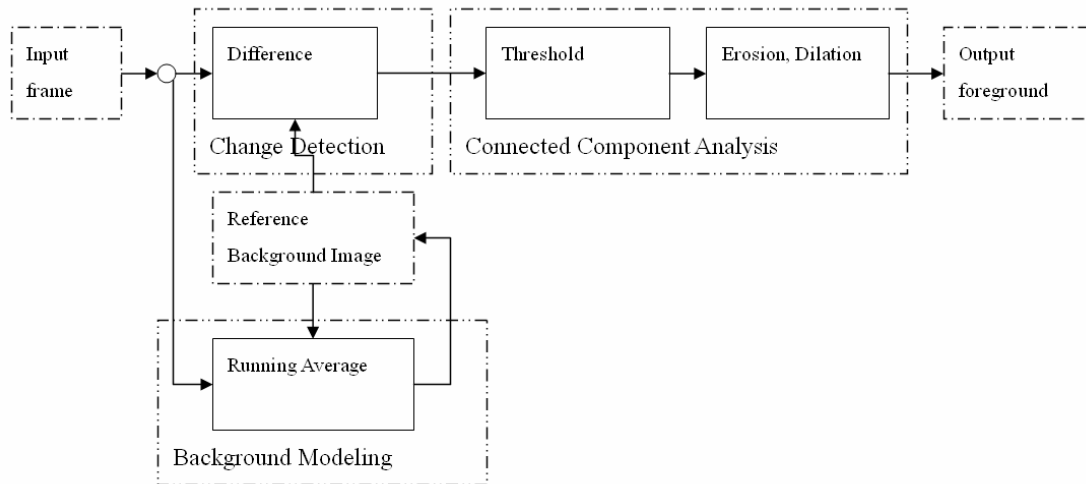


Fig 2-6 Function block of Simple Foreground/background Segmentation.

It consists of three parts: background modeling, change detection and connected component analysis. The input and output to foreground/background segmentation are image and binary foreground image respectively. The block diagram of the proposed algorithm is shown in Fig 2-6.

The background modeling method we apply in our system is **running average** method. The output of running average is obtained as

$$Bg(x, y, t) = (1 - \alpha) \times Bg(x, y, t - 1) + \alpha \times I(x, y, t) \quad \text{Eq. 2.1.2-1}$$

where $I(x, y, t)$ is current input image, $Bg(x, y, t)$ and $Bg(x, y, t - 1)$ are current and previous background image respectively, and α regulates update speed (how fast accumulator forgets about previous frames).

The change detection of foreground/background segmentation is **simple difference** method which is the simplest and fastest, but it is very sensitive to noise and illumination changes. The difference is computed as

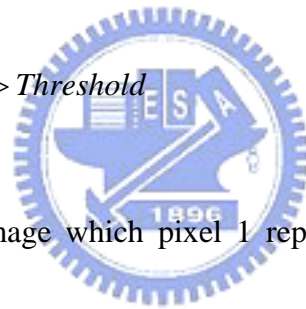
$$D(x, y, t) = |I(x, y, t) - Bg(x, y, t)| \quad \text{Eq. 2.1.2-2}$$

where $D(x, y, t)$ is absolute difference between pixel intensities of the input and background images.

Thresholding is used to segment an image. It sets all pixels whose absolute difference intensity values between background and input image are above a threshold to a foreground value (1) and all the remaining pixels to a background value (0). The binary image is obtained as

$$B(x, y, t) = \begin{cases} 1 & \text{if } D(x, y, t) > \text{Threshold} \\ 0 & \text{otherwise} \end{cases} \quad \text{Eq. 2.1.2-3}$$

where $B(x, y, t)$ is binary image which pixel 1 represents foreground and pixel 0 represents background.



The foreground we get from the thresholding process is noisy foreground. Noise can be classified as inner object noise and outer object noise. Inner object noise causes that an object is broken into multiple fractional object, in the meantime, outer object noise causes that a non-object is recognized as object by the system.

In order to reduce that noise, **morphological operations** (dilation and erosion) are used. Use of erosion is for eliminating outer object noise from a binary image. Outer object noise free binary image is obtained as

$$Er(x, y, t) = \min_{((x', y') \text{ in element})} B(x + x', y + y', t) \quad \text{Eq. 2.1.2-4}$$

where element can be rectangular, cross and ellipse shape structuring element, and $Er(x, y, t)$ is outer object noise free binary image.

The inner object noise can be cancelled by using dilation process. Noise free binary image is computed as

$$Di(x, y, t) = \max_{((x', y') \text{ in element})} Er(x + x', y + y', t) \quad \text{Eq. 2.1.2-5}$$

where $Di(x, y, t)$ is noise free binary image.

In order to get noiseless objects from binary image, the considerations of size and shape of structuring element, and iteration of erosion and dilation are important.

2.1.3. Blobs Tracking

The blobs tracking system consists of contouring, blob generation and blob update processes. This system processes binary foreground image, produces blobs and keeps update the latest blobs information.

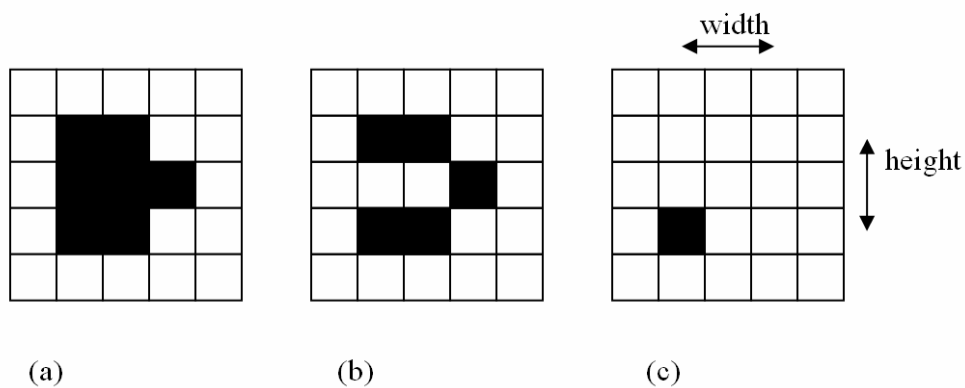


Fig 2-7 Input, intermediate and output to the contour process.

Contouring process consists of boundary representation and bounding rectangle method. The chain codes are used to represent a boundary of the object by vertices of

polygon. This representation is based on 8- connectivity of the segments. Black pixels represent foreground and white pixels represent background as shown in Fig 2-7(a). Fig 2-7 (b) shows the result of the chain codes process where black pixels represent vertices coordinate. Bounding rectangle procedure returns height and width, and bottom-left coordinate (black pixel) of object as shown in Fig 2-7 (c).

Name	Function
Id	Represent the blob
Freq	Change the blob status depend on Freq
Rect	Represent the bottom-left coordinate, width and height of the blob
Status	Represent active or candidate blob

Fig 2-8 Structure of blob

Blob generation process is used to assign each object as a blob. The structure of blob is shown in Fig 2-8. Each new blob which is assigned as candidate blob has unique ID. Rect variable represents the offset and size of a blob rectangle. Freq variable is set by a predetermined value (CANDIDATE_FREQ). CANDIDATE_FREQ is larger than zero and smaller than ACTIVE_FREQ. The purpose of the freq variable is changing the blob status. Freq variable will be changed in the Blob update process. When the freq variable is larger than ACTIVE_FREQ, the blob is assigned as an active blob. The blob will be deleted when the freq variable is smaller than zero. The blob is assigned as candidate blob in other situations.

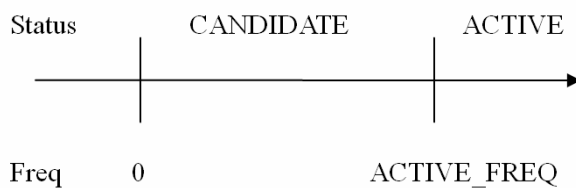


Fig 2-9 Relationship between status and frequency

Blob update process is used to assign each object whether as a new blob or merged blob. The object will be merged with memory blob in two situations as follows:

#the object most near the memory blob

#the object overlaps with the memory blob

The object which does not satisfy above situations will be assigned as a new blob. The frequency of merged memory blob will be increased. The frequency of unmerged memory blob will be decreased. The detailed blob tracking algorithm is described as follows:

Blob Tracking Algorithm

Input: Binary Foreground image

Output: list of blob structure

Procedure

Contouring Process

1. for each object, represent boundary by chain codes
2. calculate height, width and bottom-left coordinate from each object boundary
3. If first-time blob generation, go to step 4. Otherwise go to step 5.

Blob Generation and Update

4. Blob Generation
 - a. For each object, create new blob
 - i. Set unique Id.
 - ii. Set height, width and bottom-left coordinate in Rect variable.
 - iii. Set CANDIDATE_FREQ in Freq variable.



- iv. Set CANDIDATE in Status variable.
- v. If last object, return output. Otherwise go to step 4-a

5. Blob Update

- a. Decrease the Freq variable by 1 for each blob.
- b. For each current frame object,
 - i. If nearest distance of the memory blob $<$ MAXDISTANCE, go to step 5-b-iii. Otherwise go to step 5-b-ii
 - ii. If overlap with the blob, go to step 5-b-iii. Otherwise go to step 5-b-iv
 - iii. Match the memory blob
 - I. Increase the Freq variable by 2
 - II. Modify the Rect variable depend on height, width and bottom-left coordinate of the object.
 - III. If last object, go to step 5-c. Otherwise go to step 5-b.
 - iv. Create new memory blob
 - I. Set unique Id.
 - II. Set height, width and bottom-left coordinate in Rect variable.
 - III. Set CANDIDATE_FREQ in Freq variable.
 - IV. Set CANDIDATE in Status variable.
 - V. If last object, go to step 5-c. Otherwise go to step 5-b.
- c. For each memory blob,
 - i. If $\text{Freq} < 0$, delete blob
 - ii. If $\text{Freq} > \text{ACTIVE_FREQ}$, set ACTIVE in Status variable.
 - iii. If $0 < \text{Freq} < \text{ACTIVE_FREQ}$, set CANDIDATE in Status variable.
- d. Return output.

2.2. Coordinate Transformation System

We introduce knowledge about the Computer Vision as below

- ✓ Coordinate System Changes and Rigid Transformations described in section 2.2.1 tell about elementary notions of analytical Euclidean geometry.
- ✓ Intrinsic camera parameters described in section 2.2.2 that relate the camera coordinate system to the idealized coordinate system.
- ✓ Homography matrix describe in section 2.2.3 and 2.2.4 that defines a planar mapping between two overlapping camera views.

Then, we build the pan, tilt and zoom tables in section 2.2.5. The execution of coordinate transformation system which will give a close-up of each blob based on size and coordinate of the rectangle, and PTZ and homography lookup table is introduced in section 2.2.6.

2.2.1. Coordinate System Changes and Rigid Transformations

When several different coordinate systems are considered at the same time, it is convenient to denote

$${}^wP = \overrightarrow{O_wP} = x_i \mathbf{i}_w + y_j \mathbf{j}_w + z_k \mathbf{k}_w \quad \text{Eq. 2.2.1-1}$$

where ${}^W P$ is the coordinate vector of point P in the frame W .

Let us consider two coordinate systems: $(W) = (O_W, i_W, j_W, k_W)$

and $(C1) = (O_{C1}, i_{C1}, j_{C1}, k_{C1})$ are world and camera 1 coordinate systems respectively.

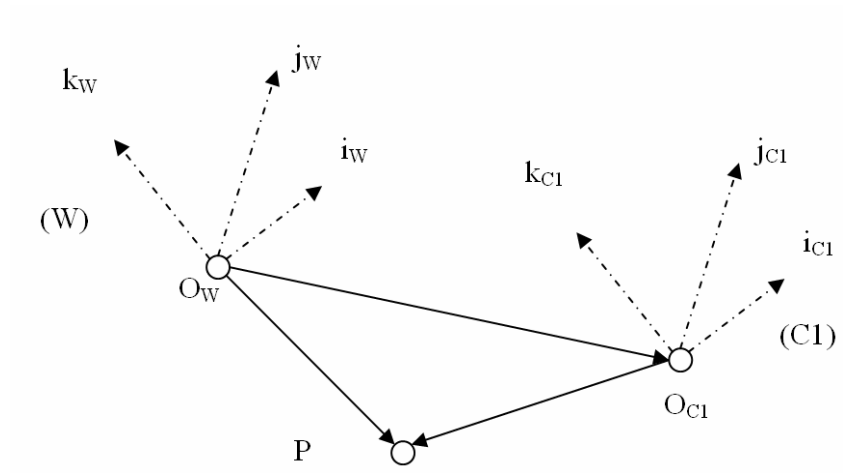


Fig 2-10 Pure translation

Fig 2-10 shows **pure translation** that basis vectors of both coordinate systems are parallel to each other, but the origin O_W and O_{C1} are different. We

have $\overrightarrow{O_{C1}P} = \overrightarrow{O_{C1}O_W} + \overrightarrow{O_WP}$, thus ${}^{C1}P = {}^W P + {}^{C1}O_W$. Eq. 2.2.1-2

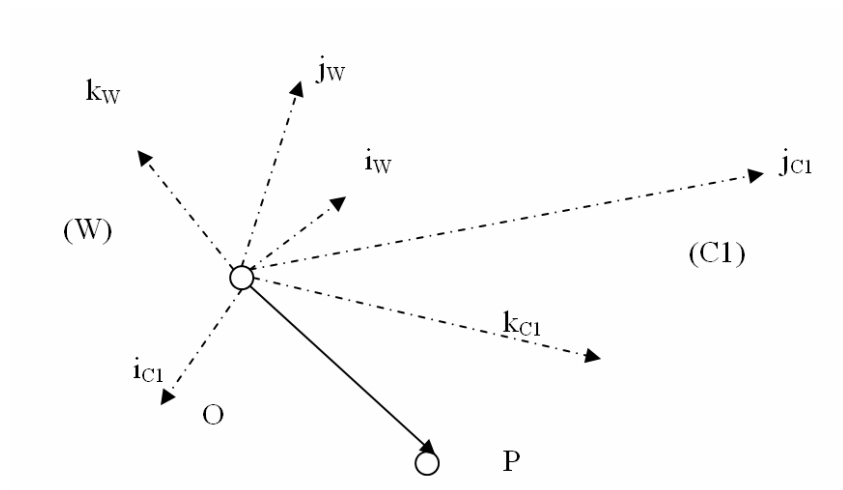


Fig 2-11 Pure rotation

When the origins of the two coordinate systems coincide, then the frames are separated by a **pure rotation** shown in Fig 2-11. Let us define the rotation matrix as

$${}^{C1}_W R = \begin{pmatrix} i_W \cdot i_{C1} & j_W \cdot i_{C1} & k_W \cdot i_{C1} \\ i_W \cdot j_{C1} & j_W \cdot j_{C1} & k_W \cdot j_{C1} \\ i_W \cdot k_{C1} & j_W \cdot k_{C1} & k_W \cdot k_{C1} \end{pmatrix}, \text{ then we have}$$

$${}^{C1}P = {}^{C1}_W R {}^W P \tag{Eq. 2.2.1-3}$$

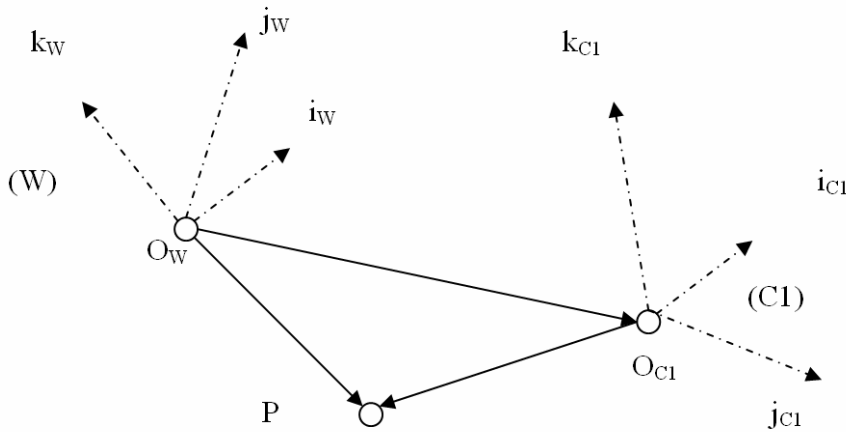


Fig 2-12 Rigid Transformation

The frames are separated by a **rigid transformation** (translation + rotation) when the origins and basis vectors of the two coordinate systems are different. We have

$${}^{C1}P = {}^{C1}_W R {}^W P + {}^{C1}O_W \tag{Eq. 2.2.1-4}$$

2.2.2. Intrinsic camera parameters

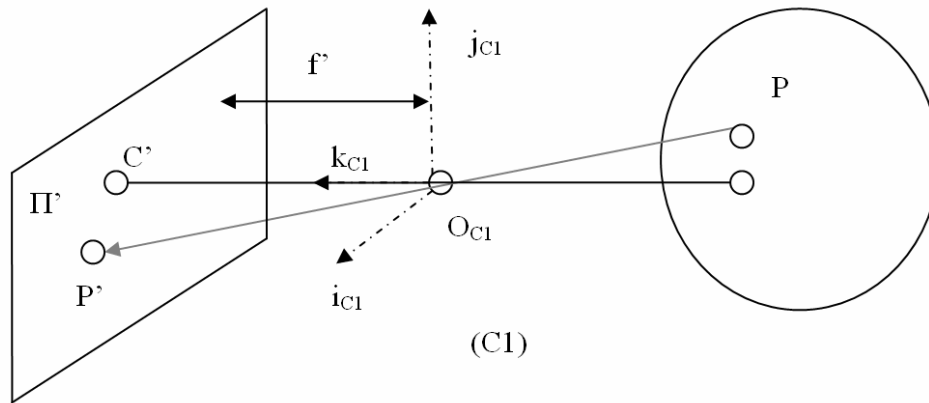


Fig 2-13 Perspective projection model

Fig 2-13 shows that a camera 1 coordinate system $(C1) = (O_{C1}, i_{C1}, j_{C1}, k_{C1})$ attached to a pinhole camera, whose origin O_{C1} coincides with the pinhole, and vector i_{C1} and j_{C1} form a basis for a vector plane parallel to the image plane Π' , which is located at a positive distance f' from the pinhole along the vector k_{C1} . The line perpendicular to Π' and passing through the pinhole is called the optical axis, and the point C' where it pierces Π' is called the image center.

Let ${}^{C1}P$ denote a scene point with the coordinates (x, y, z) and ${}^{C1}P'$ denote its images with coordinates (x', y', z') . We have $z'=f'$ because ${}^{C1}P'$ lies in the image plane. We have $\overrightarrow{O_{C1}P'} = \lambda \overrightarrow{O_{C1}P}$ for some number λ , because the three points ${}^{C1}P'$, O_{C1} and ${}^{C1}P$ are collinear. So,

$$\begin{cases} x' = \lambda x \\ y' = \lambda y \Leftrightarrow \lambda = \frac{x'}{x} = \frac{y'}{y} = \frac{f'}{z} \\ f' = \lambda z \end{cases} \quad \text{Eq. 2.2.2-1}$$

and therefore

$$\begin{cases} x' = f' \frac{x}{z} \\ y' = f' \frac{y}{z} \end{cases} \quad \text{Eq. 2.2.2-2}$$

Eq. 2.2.2-2 is valid when all distances are measured in the camera's reference frame, and image coordinates have their origin at the principal point where the axis of symmetry of the camera pierces its retina.

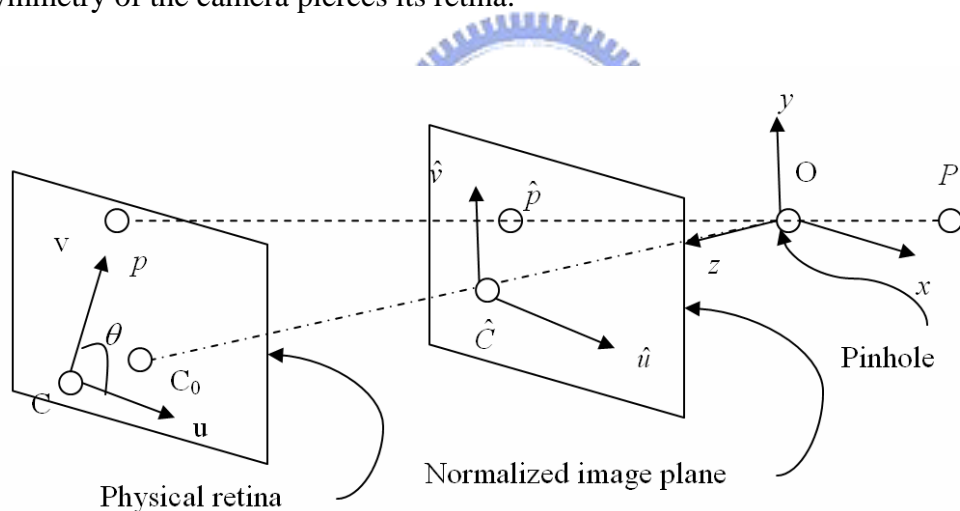


Fig 2-14 Physical and normalized image coordinate systems.

Now, we make an assumption that camera 1 coordinate system $(C1) = (O_{C1}, i_{C1}, j_{C1}, k_{C1})$ is same as world coordinate system $(W) = (O_w, i_w, j_w, k_w)$ as shown in Fig 2-14. Normalized image plane is located at a unit distance from the pinhole and parallel to its physical retina. Eq. 2.2.2-2 can be rewritten in this normalized coordinate system as

$$\begin{cases} \hat{u} = \frac{x}{z} \\ \hat{v} = \frac{y}{z} \end{cases} \quad \text{Eq.2.2.2-3}$$

Fig 2-14 shows that physical retina of the camera is located at a distance $f \neq 1$ from the pinhole. The image coordinates (u, v) of the images point p are expressed in pixel units where pixels are rectangular, so the camera has two additional scale parameters k and l . Eq.2.2.2-2 can be rewritten as

$$\begin{cases} u = kf \frac{x}{z} \\ v = lf \frac{y}{z} \end{cases} \quad \text{Eq.2.2.2-4}$$

where k and l are expressed in pixel/meter.

Fig 2-14 also shows that the center of CCD matrix does not coincide with the principal point C_0 , so Eq.2.2.2-4 can be rewritten as

$$\begin{cases} u = kf \frac{x}{z} + u_0 \\ v = lf \frac{y}{z} + v_0 \end{cases} \quad \text{Eq.2.2.2-5}$$

where u_0 and v_0 that define the position of C_0 in the retinal coordinate system are expressed in pixel units.

The angle θ between the two image axes is not exactly equal to 90 degree due to some manufacturing error, so Eq.2.2.2-5 transforms into

$$\begin{cases} u = kf \frac{x}{z} - kf \cot \theta \frac{y}{z} + u_0 \\ v = \frac{lf}{\sin \theta} \frac{y}{z} + v_0 \end{cases} \quad \text{Eq.2.2.2-6}$$

Eq.2.2.2-6 can be rewritten into compact form as

$${}^{C1}p = \frac{1}{z} M_1 {}^{C1}P \quad \text{Eq.2.2.2-7a}$$

where ${}^{C1}p = (u, v, 1)^T$, ${}^{C1}P = (x, y, z, 1)^T$ and

$$M_1 = \begin{pmatrix} kf & -kf \cot \theta & u_0 & 0 \\ 0 & \frac{lf}{\sin \theta} & v_0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} = (K_1 \ 0) \quad \text{where } 0 = (0, 0, 0)^T$$

Matrix K_1 is an intrinsic parameter related matrix of camera 1 where k and l are related to size of pixels, u_0 and v_0 are related to position of the principal point, f is related to focal length of the lens, and θ is related to manufacturing skew error.

The depth z in Eq. 16a is not independent of M_1 and ${}^{C1}P$, then we rewrite Eq.2.2.2-7a as

$$\begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = \frac{1}{z} M_1 {}^{C1}P$$

$$\begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = \frac{1}{z} \begin{pmatrix} m_1^T \\ m_2^T \\ m_3^T \end{pmatrix} {}^{C1}P \Leftrightarrow \begin{cases} u = \frac{m_1^T \bullet {}^{C1}P}{z} \\ v = \frac{m_2^T \bullet {}^{C1}P}{z} \\ 1 = \frac{m_3^T \bullet {}^{C1}P}{z} \end{cases} \quad \text{Eq.2.2.2-7b}$$

where m_1^T, m_2^T, m_3^T denote the three rows of M_1 . We rewrite again Eq.2.2.2-7a as

$${}^{c1}p = \frac{1}{\lambda_1} M_1 {}^{c1}P \quad \text{Eq.2.2.2-7c}$$

where $\lambda_1 = m_3^T \bullet {}^{c1}P$

2.2.3. Homography Derivation

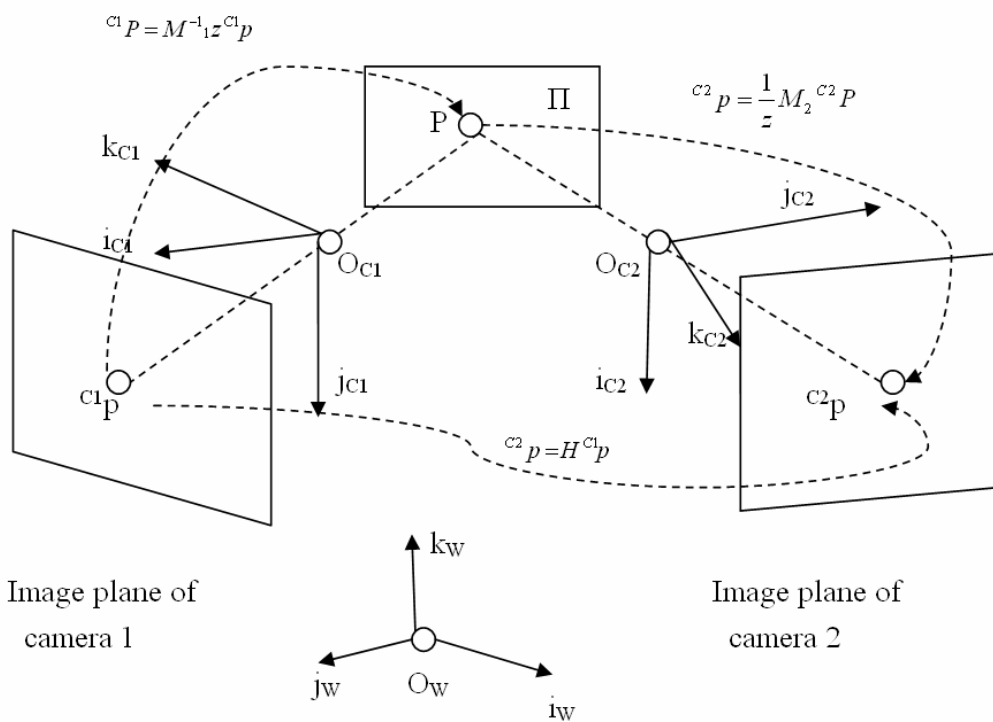


Fig 2-15 The homography matrix, H induced by a plane Π

The homography matrix being derived here is a matrix which maps pixels in camera 1 to pixels in camera 2. We can use rigid body motion to map points from the world coordinate system to camera 1 coordinate system like Eq. 2.2.1-4, ${}^{c1}O_w$ (translation

vector) is in the same reference coordinate system as ${}^{C1}P$. We modify Eq. 2.2.1-4 to be

$${}^{C1}P = {}^{C1}_W R ({}^W P - {}^W O_{C1}) \quad \text{Eq. 2.2.3-1}$$

where ${}^W O_{C1}$ (translation vector) is in the same reference coordinate system as ${}^W P$.

Similarly for camera 2:

$${}^{C2}P = {}^{C2}_W R ({}^W P - {}^W O_{C2}) \quad \text{Eq. 2.2.3-2}$$

We rearrange Eq. 2.2.3-1 as

$${}^W P = {}^{C1}_W R^T {}^{C1}P + {}^W O_{C1} \quad \text{Eq. 2.2.3-3}$$

Inserting Eq. 2.2.3-3 into Eq. 2.2.3-2, we obtain

$${}^{C2}P = {}^{C2}_W R ({}^{C1}_W R^T {}^{C1}P + {}^W O_{C1} - {}^W O_{C2}) \quad \text{Eq. 2.2.3-4}$$

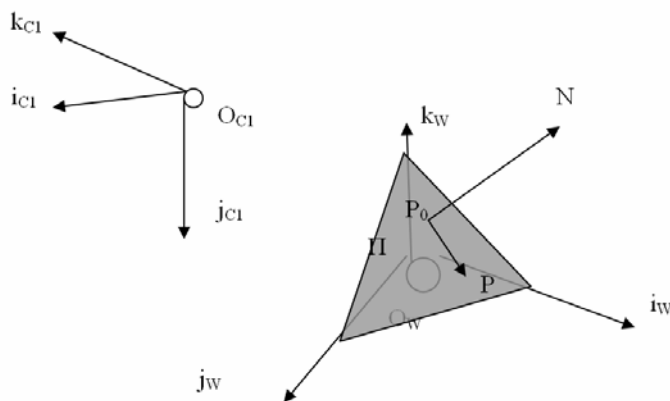


Fig 2-16 Plane II with normal orthogonal vector $\overrightarrow{P_0 N}$

$$\overrightarrow{P_0 N} = \overrightarrow{O_w N} - \overrightarrow{O_w P_0} = {}^W N - {}^W P_0$$

Let ${}^{C1}R({}^W N - {}^W P_0)$ be unit normal orthogonal vector of the plane Π with respect to the camera 1 coordinate system, the inner product of any point ${}^{C1}P$ that lie on plane Π with ${}^{C1}R({}^W N - {}^W P_0)$ equals to zero. We can write as

$$({}^{C1}R({}^W N - {}^W P_0)) \bullet ({}^{C1}P - {}^{C1}P_0) = 0 \quad \text{Eq. 2.2.3-5}$$

Since $({}^{C1}P - {}^{C1}P_0) = (x - x_0, y - y_0, z - z_0)$ and ${}^{C1}R({}^W N - {}^W P_0) = (a, b, c)$,

Eq. 2.2.3-5 can be written as

$$\begin{aligned} a(x - x_0) + b(y - y_0) + c(z - z_0) &= 0 \\ ax + by + cz &= ax_0 + by_0 + cz_0 \end{aligned}$$

$$ax + by + cz = d$$

Eq. 2.2.3-6

where $d = ax_0 + by_0 + cz_0$



We write Eq. 2.2.3-5 as

$$({}^{C1}R({}^W N - {}^W P_0)) \bullet {}^{C1}P = d$$

$$\frac{1}{d} ({}^{C1}R({}^W N - {}^W P_0))^T {}^{C1}P = 1$$

Eq. 2.2.3-7

Inserting Eq. 2.2.3-7 into Eq. 2.2.3-4 we obtain

$${}^{C2}P = {}^{C2}R({}^{C1}R^T {}^{C1}P + ({}^W O_{C1} - {}^W O_{C2}) \frac{1}{d} ({}^{C1}R({}^W N - {}^W P_0))^T {}^{C1}P) \quad \text{Eq. 2.2.3-8}$$

Factoring ${}^{C1}P$ out, we have

$${}^{C2}P = {}^{C2}R({}^{C1}R^T + ({}^W O_{C1} - {}^W O_{C2}) \frac{1}{d} ({}^{C1}R({}^W N - {}^W P_0))^T) {}^{C1}P \quad \text{Eq. 2.2.3-9}$$

Refer Eq.2.2.2-7c , we can write

$${}^{C2}p = \frac{1}{\lambda_2} M_2 {}^{C2}P \quad \text{Eq.2.2.3-10}$$

Rearrange Eq.2.2.2-7c and Eq.2.2.3-10, and substituting into Eq.2.2.3-9, we obtain

$$\lambda_2 M_2^{-1} {}^{C2}p = {}^{C2}R({}^{C1}R^T + ({}^W O_{C1} - {}^W O_{C2}) \frac{1}{d} ({}^{C1}R({}^W N - {}^W P_0))^T) \lambda_1 M_1^{-1} {}^{C1}p$$

Eq.2.2.3-11

Rearrange Eq.2.2.3-11 to

$$\lambda {}^{C2}p = H {}^{C1}p \quad \text{Eq.2.2.3-12}$$

where $H = M_2 {}^{C2}R({}^{C1}R^T + ({}^W O_{C1} - {}^W O_{C2}) \frac{1}{d} ({}^{C1}R({}^W N - {}^W P_0))^T) M_1^{-1}$ and

$$\lambda = \frac{\lambda_2}{\lambda_1}$$

The homography matrix maps pixels on web camera plane (C1) to PTZ camera plane (C2). Target tracking system sends the size and coordinate of each object on web camera to coordinate transformation system. Coordinate transformation system maps the coordinate on web camera to PTZ camera by using homography matrix.

2.2.4. Homography Calculation

The Eq.2.2.3-12 may be expressed in terms of the vector cross product as

$$0 = {}^{c2}p \times (H {}^{c1}p)$$

$$\begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} {}^{c2}x \\ {}^{c2}y \\ {}^{c2}z \end{pmatrix} \times \begin{pmatrix} h_1^T {}^{c1}p \\ h_2^T {}^{c1}p \\ h_3^T {}^{c1}p \end{pmatrix}$$

$$\begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} {}^{c2}y h_3^T {}^{c1}p - {}^{c2}z h_2^T {}^{c1}p \\ {}^{c2}z h_1^T {}^{c1}p - {}^{c2}x h_3^T {}^{c1}p \\ {}^{c2}x h_2^T {}^{c1}p - {}^{c2}y h_1^T {}^{c1}p \end{pmatrix} \quad \text{Eq.2.2.4-1}$$

where h_1^T, h_2^T, h_3^T denote the three rows of H and ${}^{c2}p = ({}^{c2}x \quad {}^{c2}y \quad {}^{c2}z)^T$.

The Eq.2.2.4-1 may be written in the form

$$\begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} 0^T & -{}^{c2}z {}^{c1}p^T & {}^{c2}y {}^{c1}p^T \\ {}^{c2}z {}^{c1}p^T & 0^T & -{}^{c2}x {}^{c1}p^T \\ -{}^{c2}y {}^{c1}p^T & {}^{c2}x {}^{c1}p^T & 0^T \end{pmatrix} \begin{pmatrix} h_1 \\ h_2 \\ h_3 \end{pmatrix} \quad \text{Eq.2.2.4-2}$$

Although Eq.2.2.4-2 contains three equations, only two of them are linearly independent. Thus the set of equations can be written as

$$\begin{pmatrix} 0 \\ 0 \end{pmatrix} = \begin{pmatrix} 0^T & -{}^{c2}z {}^{c1}p^T & {}^{c2}y {}^{c1}p^T \\ {}^{c2}z {}^{c1}p^T & 0^T & -{}^{c2}x {}^{c1}p^T \end{pmatrix} \begin{pmatrix} h_1 \\ h_2 \\ h_3 \end{pmatrix} \quad \text{Eq.2.2.4-3}$$

Given n points, Eq.2.2.4-3 can be written in matrix equation as

$$Lh = 0$$

where $h = [h_1^T \quad h_2^T \quad h_3^T]^T$ and L is a $2n \times 9$ matrix. As h is defined up to a scale factor, the solution is unit singular vector of L associated with the smallest singular value (or equivalently, the eigenvector of $L^T L$ associated with the smallest eigenvalue).

Proof of $Lh=0$

Find the h that minimizes $\|Lh\|$ subject to $\|h\|=1$. The singular value decomposition is a factorization of L as $L=UDV^T$, where U and V are orthogonal matrices, and D is a diagonal matrix with non-negative entries in descending order. The problem requires us to minimize $\|UDV^T h\|$.

However,

$$\begin{aligned}\|UDV^T h\| &= \left((UDV^T h) \cdot (UDV^T h) \right)^{\frac{1}{2}} = \left((UDV^T h)^T (UDV^T h) \right)^{\frac{1}{2}} \\ &= \left((DV^T h)^T U^T (UDV^T h) \right)^{\frac{1}{2}} = \left((DV^T h)^T U^T U (DV^T h) \right)^{\frac{1}{2}} \\ &= \left((DV^T h)^T U^{-1} U (DV^T h) \right)^{\frac{1}{2}} = \left((DV^T h)^T (DV^T h) \right)^{\frac{1}{2}} \\ &= \left((DV^T h) \cdot (DV^T h) \right)^{\frac{1}{2}} = \|DV^T h\|\end{aligned}$$

and similarly, $\|h\| = \|V^T h\|$.

Thus, we need to minimize $\|DV^T h\|$ subject to the condition $\|V^T h\|=1$. We write $y = V^T h$, and the problem is: minimize $\|Dy\|$ subject to $\|y\|=1$. It follows that the solution to this problem is $y = (0,0,\dots,0,1)^T$ having one non-zero entry. Finally, $h = Vy$ is simply the last column of V .

Algorithm

Input: $n \geq 4$ 2D to 2D point correspondences $p_i \leftrightarrow^{C1} p_i^{C2}$

Output: 2D homography matrix H

Procedure

1. For each correspondence ${}^{C2}p_i \leftrightarrow {}^{C1}p_i$ compute L_i .
2. Assemble n 2×9 matrices L_i into a single $2n \times 9$ matrix L .
3. Obtain SVD of L as UDV^T with D diagonal with positive diagonal entries, arranged in descending order down the diagonal, then h is last column of V .
4. Determine H from h .

2.2.5. Generation of Pan, Tilt and Zoom Tables

Coordinate transformation system adjusts the panning and tilting parameters based on coordinate information in order to frame the object. After movement of PTZ, the new coordinate of object is at center of the PTZ frame. For instance, if the object is at the bottom of the PTZ frame, we control the tilt parameter of the camera and move the frame upwards. Similarly, if the object is at the left of the PTZ frame, we pan the frame leftward.

	TL		T		TR
	L		C		R
	BL		B		BR

Fig 2-17 Table of Pan/Tilt degree correspondent to coordinate of the frame.

Now, we build the pan and tilt tables based on some control

points(TL,T,TR,L,C,R,BL,B,BR). We assume TL pixel coordinate as a center of the object, then we manually adjust PTZ until the new TL pixel coordinate is at the center frame. We record the pan and tilt degree in the pan and tilt tables respectively. We find the others control points using similar method.

After getting pan and tilt degree at each control points, we generate dense pan and tilt tables for all pixels by using interpolation and extrapolation method. The pan and tilt degree at gray pixels in Fig 2-17 are generated by using interpolation method. Extrapolation method is used to generate pan and tilt degree at other pixels in Fig 2-17.

After we centrally frame the target object by doing panning and tilting operations, we give a close-up of target object with appropriate zoom ratio. Appropriate zoom ratio means that PTZ give a close-up of whole target object with large zoom ratio.

We build the zoom table for each zoom level. We assume that (a) the shape of the object is square and (b) the object is at the center of the frame. We set the zoom parameter as nine, then, we find maxima length of square (20) that PTZ can give a close-up of whole target object. Similarly, we set the zoom parameter as eight, then, we find maxima length of square (26). If the length of square is 27, we can't give a close-up of whole object with zoom ratio eight.

Pixels	240	152	89	69	55	36	30	26	20
Zoom	1	2	3	4	5	6	7	8	9

Fig 2-18 Table of Zoom degree correspondent to pixels of the object.

If the shape of the object is rectangle, we find the maxima value of height and width. For example, if the value is greater than 152, then set the zoom ratio as one.

Fig 2-19 shows a lookup table of zoom ratio.

(Input) Pixels	(Output) Zoom Ratio	(Input) Pixels	(Output) Zoom Ratio
<20	9	$55 \leq \text{pixels} < 69$	4
$20 \leq \text{pixels} < 26$	8	$69 \leq \text{pixels} < 89$	3
$26 \leq \text{pixels} < 30$	7	$89 \leq \text{pixels} < 152$	2
$30 \leq \text{pixels} < 36$	6	152>	1
$36 \leq \text{pixels} < 55$	5		

Fig 2-19 Lookup Table of Zoom Ratio

2.2.6. Execution of Coordinate Transformation System



The Coordinate Transformation System Stage gives a close-up of each blob based on size and coordinate of the blob, and PTZ and homography lookup table as shown in Fig 2-20.

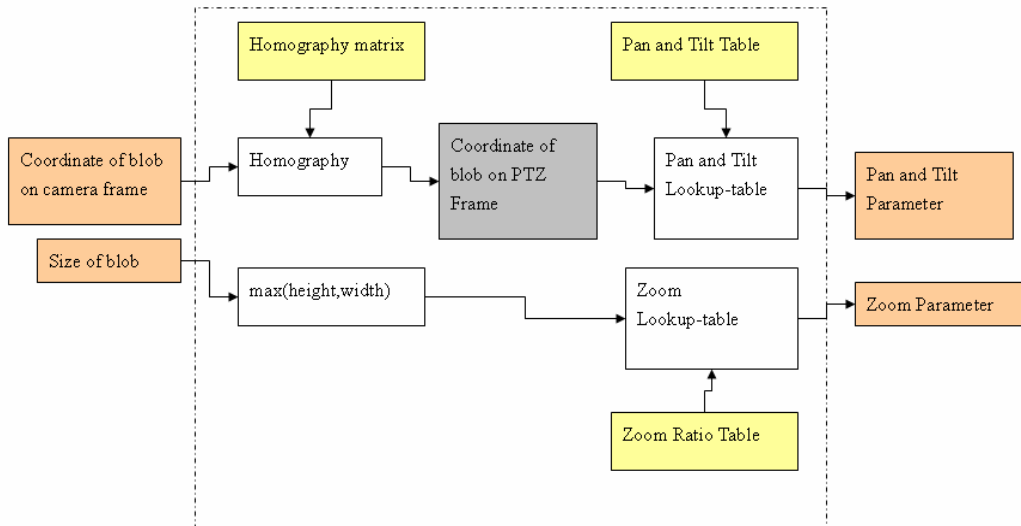


Fig 2-20 Function block of Coordinate Transformation System

The homography function maps coordinate of blob on web camera to PTZ camera by using homography matrix. The pan and tilt lookup-table function takes coordinate of blob on PTZ camera to generate pan and tilt parameters by using pan and tilt tables. We find the maxima value of height and width of the blob, then, the zoom lookup-table function takes maxima value to generate zoom parameter by using zoom table.

3. Experimental Results

In this chapter we present the test environment and the experimental results of our algorithms.

3.1. Test application and system

The test application is implemented using Microsoft Visual C++ with DirectShow and OpenCv library. All of the tests in next sections are performed by using test application on Microsoft Windows XP operating system on a computer with an Intel Pentium® 4 CPU 2.40GHz and 760MB RAM. Upmost MTV video capture card send the video signal from PTZ camera (Sony EVI-D70) to our computer. Web camera (Logitech QuickCam IM) send the video signal to computer through USB port. We can control the PTZ camera through RS 232. Fig 3-1 shows the two camera set-up.



Fig 3-1 Camera Set-up

Fig 3-2 shows the software and hardware system architecture block diagram. The

webcam and PTZ camera appear as two different video input source filter in DirectShow. The DirectShow passes the video sources to the OpenCV for image processing. Serial communication API controls the PTZ through RS232.

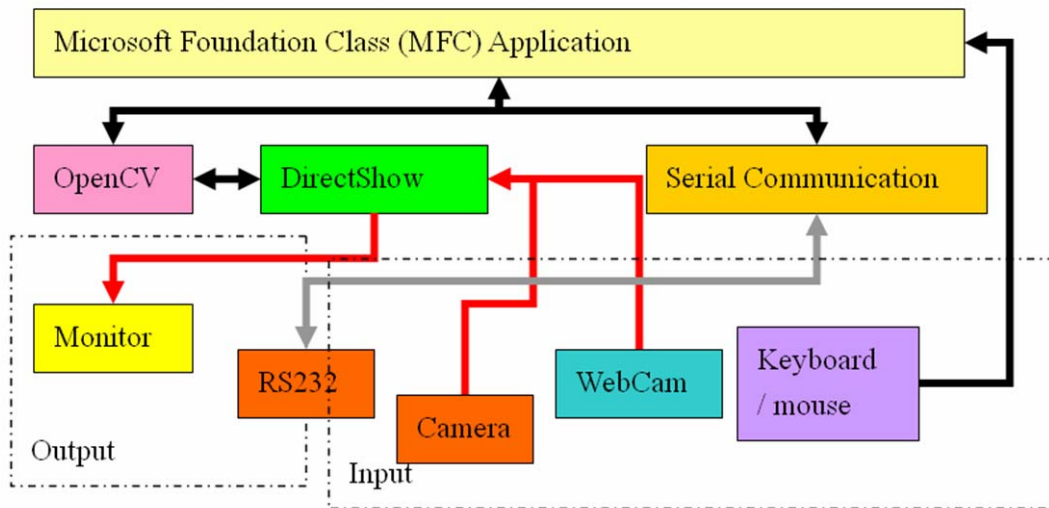


Fig 3-2 Software and hardware system architecture block diagram



3.2. Result of Target Tracking System

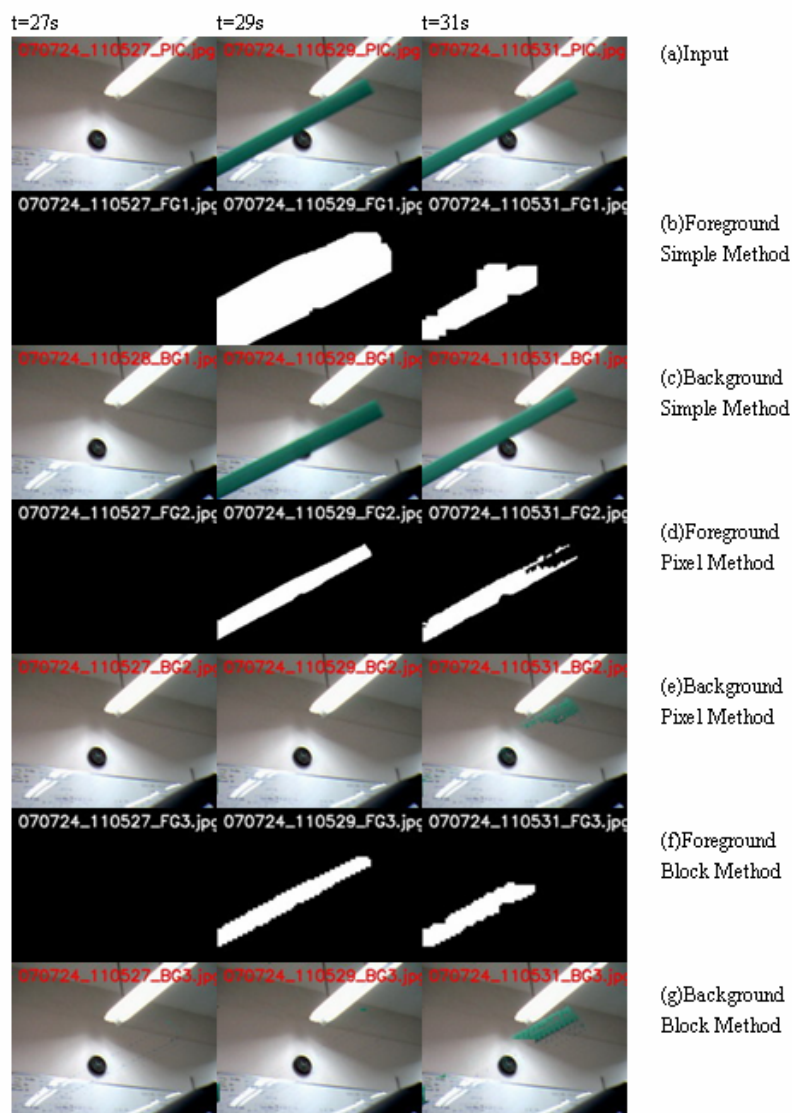


Fig 3-3 Foreground and background segmentation (part I- moving object enters the scene)

Fig 3-3, Fig 3-4 and Fig 3-5 show the results of foreground and background segmentation. Row (a) shows the input of the video. Foreground and background

images of the simple foreground/background segmentation method are shown in row (b) and row (c) respectively. Row (d) and row (e) show the results of foreground and background images of pixel based foreground/background segmentation method. The images shown in row (f) and row (g) are foreground and background images of the block-based foreground/background segmentation method respectively.

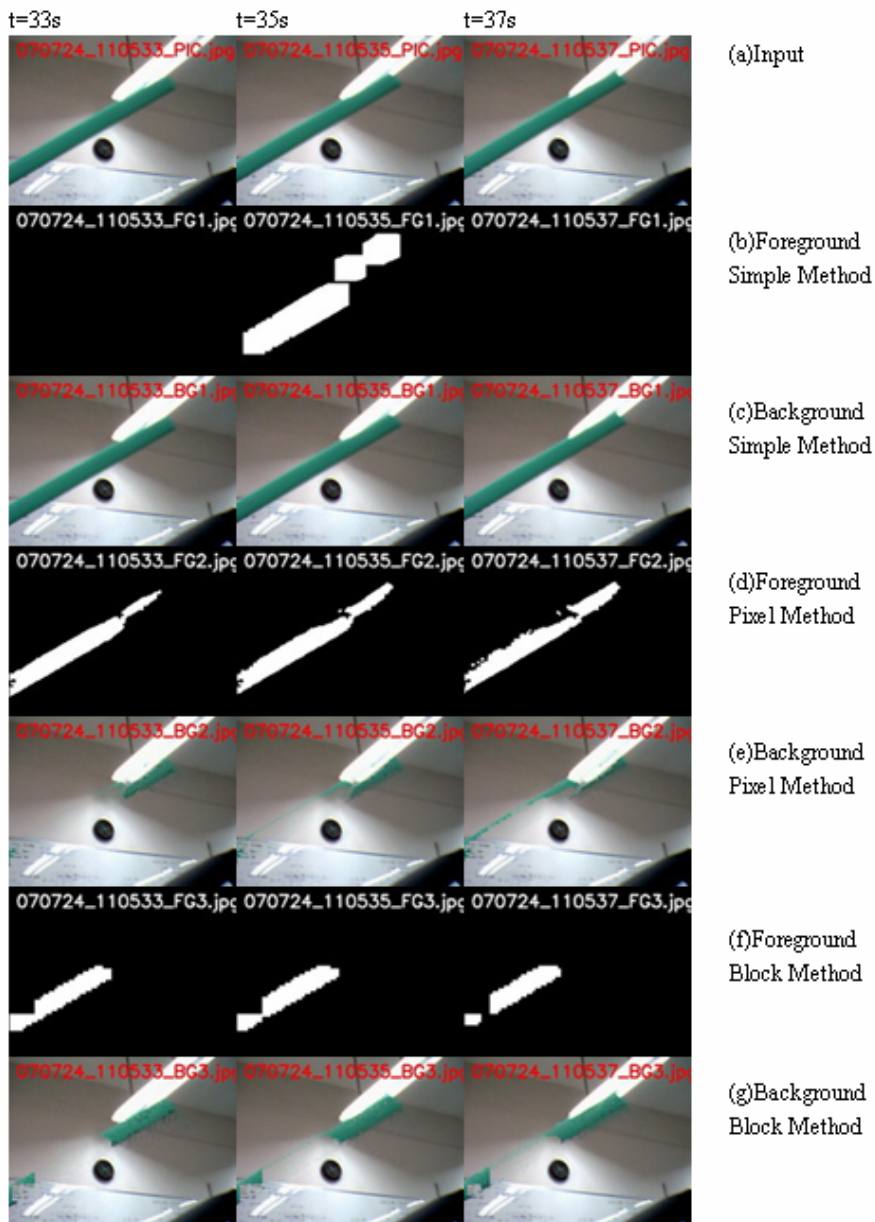


Fig 3-4 Foreground and background segmentation (part II- moving object stays temporary) in the scene)

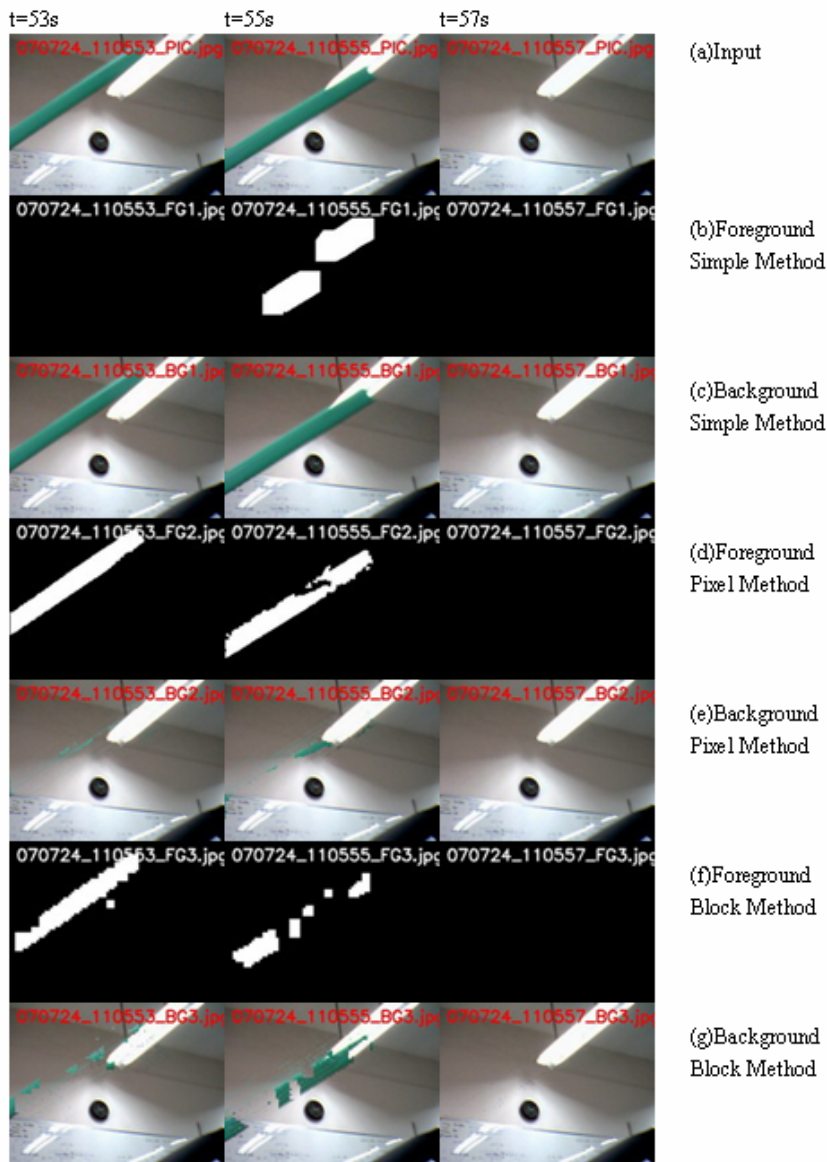


Fig 3-5 Foreground and background segmentation (part III- moving object leaves the scene)

The scenario which moving object enters the scene is shown in Fig 3-3. The result of pixel-based method was more accurate than the two methods. Simple method shows the poor quality of the detection.

Fig 3-4 shows that moving object moves slowly (or stays temporary) in the scene. The simple method shows unstable result that missed foreground object temporary. On the other hand, pixel-based and block-based methods show promising results.

Pixel-based method can more accurately detect the foreground pixels than the block-based method.

The scenario which moving object leaves the scene is shown in Fig 3-5. All method wouldn't misclassify the background points as the foreground when there are stationary foregrounds leave the scene.

As shown in Fig 3-6, the target tracking system automatically tracks two moving objects. Each object has unique ID.

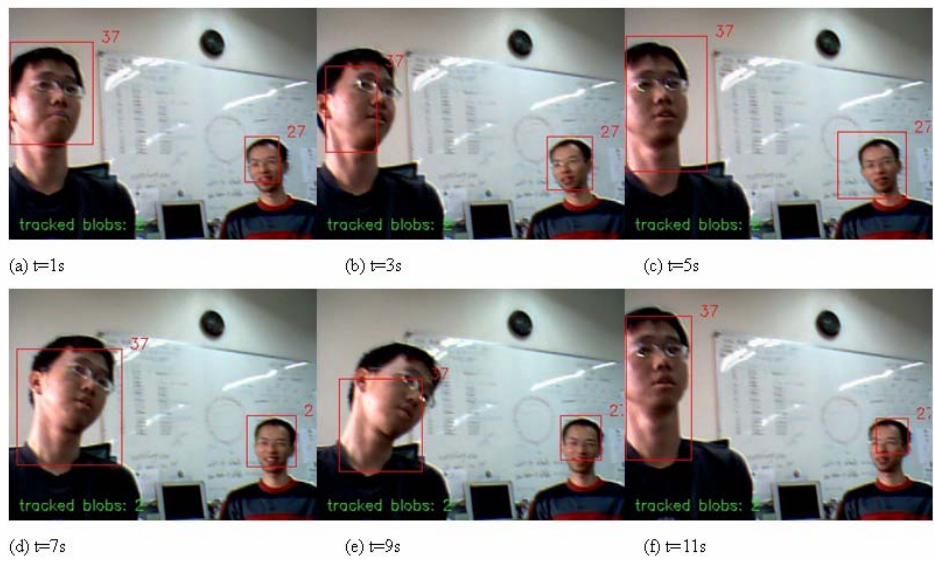


Fig 3-6 Tracking the moving objects

3.3. Module of Homography

Fig 3-7 shows that eight pairs of point are selected manually. These points are used as input of homography calculation algorithm described in section 2.2.4. The output of this algorithm is a homography matrix which maps coordinate of rectangle on source frame to destination frame as shown in Fig 3-8.



Fig 3-7 Eight pairs of point selected manually

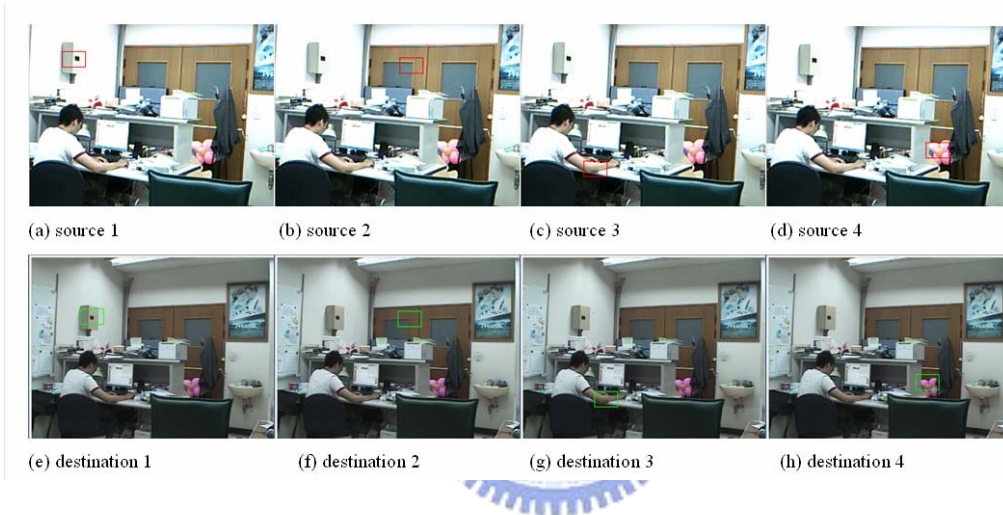


Fig 3-8 Homography projected results

3.4. Result of Whole System

The whole system tracks the single moving object and gives a close-up as shown in Fig 3-9. The result shows that this system can smoothly track the moving object and give a close-up of whole moving object.



(a) t=5s



(b) t=7s



(c) t=9s



(d) t=12s



(e) t=15s

Fig 3-9 Single object tracking with close-up

The whole system can also track the multiple moving objects and gives each close-up

as shown in Fig 3-10. The result shows that this system assigns time slices to each moving object in equal portions for tracking and giving a close-up of whole moving object.

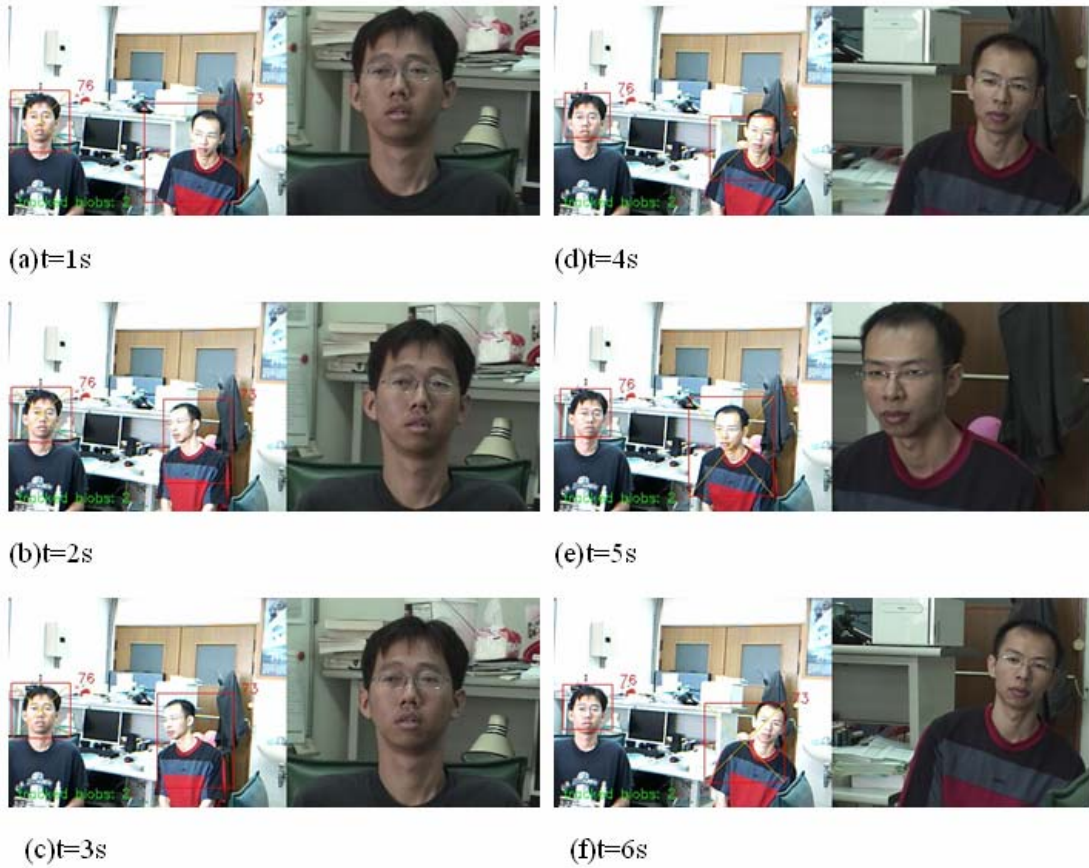


Fig 3-10 Multiple objects tracking with close-up

4. Conclusion and Future Works

In this thesis we presented a set of methods for moving object tracking system based on dual cameras (webcam and PTZ camera).

The block based foreground segmentation method gives the promising results in terms of detection quality and computational complexity to be used in a real-time surveillance system.

The proposed blob tracking algorithm successfully tracks objects in consecutive frames. Nearest neighbor matching scheme used in our application gives promising results. It is not complicated methods for whole-body tracking of objects. Improvement in handling object occlusions must be done in the future because our system fails in distinguishing occluding objects.

The homography matrix which maps pixels in webcam to pixels in PTZ camera gives good correspondent relationship between two cameras. The homography matrix is suitable for planar scenes. The homography matrix gives an acceptable correspondent relationship result for the scenes with depth variation in our experiment environment.

Finally, we propose a novel technique which can adjust automatically camera parameters based on size and coordinate of the object.

5. References

- Ref- 1 Foresti, G.L.; Micheloni, C.; Snidaro, L.; Remagnino, P.; Ellis, T.; “Active video-based surveillance system: the low-level image and video processing techniques needed for implementation”, Signal Processing Magazine, IEEE, Volume 22, Issue 2, Mar 2005 Page(s):25 – 37
- Ref- 2 Valera, M.; Velastin, S.A.; “Intelligent distributed surveillance systems: a review”, Vision, Image and Signal Processing, IEE Proceedings-, Volume 152, Issue 2, 8 April 2005 Page(s):192 – 204
- Ref- 3 R.T. Collins, A.J. Lipton, H. Fujiyoshi, and T. Kanade, “A system for video surveillance and monitoring”, Proc. IEEE, vol. 89, no. 10, pp. 1456–1477, Oct. 2001.
- Ref- 4 C. Stauffer and E. Grimson, “Learning patterns of activity using real-time tracking,” IEEE Trans. Pattern Anal. Machine Intell., vol. 22, no. 8, pp. 747–757, 2000.
- Ref- 5 D. Comaniciu, V. Ramesh, P. Meer, “Real-time tracking of non-rigid objects using mean shift,” IEEE International Conference on Pattern Recognition, vol. 2, pp. 142-149, 13-15 June, 2000.
- Ref- 6 K. Nummiaro, E. Koller-Meier, and L. Van Gool, “A color based particle filter,” in First International Workshop on Generative-Model-Based Vision, A.E.C. Pece, Ed., 2002.

Ref- 7 R. Hartley and A. Zisserman, “Multiple View Geometry in computer vision”,2nd edition, Cambridge University Press, 2004.

Ref- 8 O. Faugeras and Q. Luong, “The Geometry of Multiple Images”,1st edition, The MIT Press, 2001.

Ref- 9 R. C. Gonzalez and R. E. Woods, “Digital Image Processing”,2nd edition, Pearson Education International, 2002.

Ref- 10 P. Rosin. Thresholding for change detection. In Proceedings of IEEE Int’l Conf. on Computer Vision, pages 274–279, 1998.



6. Appendix of Programming

In this chapter, we show two different applications that capture video inputs from a Logitech webcam, process the image sequence and render it. The first application process the image sequence by using ISampleGrabber filter created by Microsoft. Meanwhile, the second application process the image sequence by using proxy transform filter created by OpenCv. So, the second application can process the image sequence by using OpenCV library. Finally, we show a simple PTZ application that controls PTZ camera through RS232 serial port.

6.1. Simple Capturing Video from A Webcam

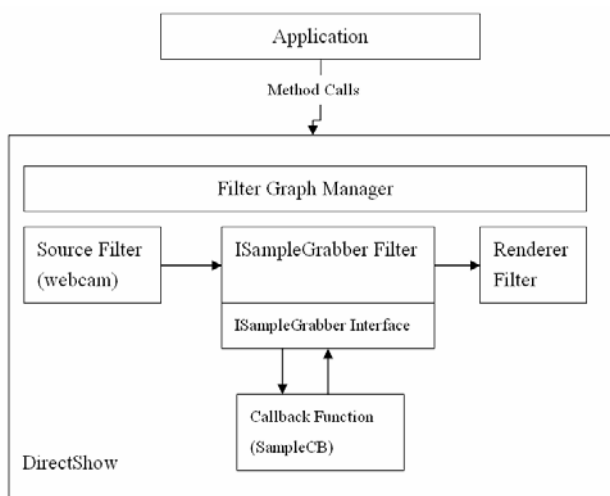


Fig 6-1 The block diagram of video capturing from webcam by using DirectShow

This application captures video inputs from a Logitech webcam, processes the image sequence, and renders it.

1. Create a dialog-based application

This application can easily be created using the MFC application wizard. The name of the application is DX_CV2. VC++ should create a simple OK/Cancel Dialog for you. The class with a name ending by Dlg will contain the member functions that control the widget of the dialog.

2. Include header files

Add the following header file to the DX_CV2Dlg.cpp file

```
////////////////////////////////////  
#include <DShow.h>           // include DirectShow interface  
#include <Qedit.h>          // include ISampleGrabber interface
```

3. Declare global variables and class

Declare the following global variables and class in the DX_CV2Dlg.cpp file

```
////////////////////////////////////  
#include <Qedit.h>          // include ISampleGrabber interface  
  
AM_MEDIA_TYPE mt;  
  
class SampleGrabberCallback :public ISampleGrabberCB  
{  
public: // The interface needed by ISampleGrabber Filter  
  
    // Fake out any COM ref counting  
    STDMETHODIMP_(ULONG) AddRef() { return 2; }  
    STDMETHODIMP_(ULONG) Release() { return 1; }  
    // Fake out any COM QI'ing  
    STDMETHODIMP QueryInterface(REFIID riid, void **ppv) {  
        if( riid == IID_ISampleGrabberCB || riid == IID_IUnknown ) {  
            *ppv = (void *) static_cast<ISampleGrabberCB*>( this );  
            return NOERROR;  
        }  
        return E_NOINTERFACE;  
    }  
}  
  
// The frame callback function called on each frame  
// We can add the processing code in this function  
STDMETHODIMP SampleCB(double SampleTime, IMediaSample *pSample){  
    // get current media type  
    VIDEOINFOHEADER *pvi = (VIDEOINFOHEADER *) mt.pbFormat;  
    BYTE *pData;           // Pointer to the actual image buffer  
    long lDataLen;        // Holds length of any given sample  
    int iPixel;           // Used to loop through the image pixels  
    pSample->GetPointer(&pData);  
    lDataLen = pSample->GetSize();  
    //////////////////////////////////////
```

```

// Get the image properties from the BITMAPINFOHEADER
int iPixelSize = pvi->bmiHeader.biBitCount / 8;
int cxImage    = pvi->bmiHeader.biWidth;
int cyImage    = pvi->bmiHeader.biHeight;
int cbImage    = cyImage * cxImage * iPixelSize;
int numPixels  = cxImage * cyImage;
BYTE *prgb = (BYTE*) pData;
// convert to grayscale image
for (iPixel=0; iPixel < numPixels; iPixel++, prgb+=iPixelSize) {
    *(prgb + 1) = *(prgb); // G channel
    *(prgb + 2) = *(prgb); // R channel
}
return 0;
}
// Another callback function. I am not using it now.
STDMETHODIMP BufferCB(double SampleTime, BYTE *pBuffer, long BufferLen) {
    return E_NOTIMPL;
}
};

IGraphBuilder *pGraph; // Graph builder object
IMediaControl *pControl = NULL; // Media control object
IBaseFilter *pVideoInputFilter = NULL; // Video Capture filter
IBaseFilter *pGrabFilter; // Sample Grabber filter
ISampleGrabber *pSampleGrabber ; // Sample Grabber interface
SampleGrabberCallback g_StillCapCB; // Sample Grabber Callback class
// Function Prototype

```

4. Declare Function Prototype

Declare the following function prototypes in the DX_CV2Dlg.cpp file

```

// Function Prototype
int webcamCap(HWND hwnd);
int releaseWebcamCap();
HRESULT GetVideoInputFilter(IBaseFilter** gottaFilter, wchar_t* matchName);
HRESULT GetUnconnectedPin(IBaseFilter *pFilter, PIN_DIRECTION PinDir, IPin **ppPin);
HRESULT ConnectFilters(IGraphBuilder *pGraph, IBaseFilter *pSrc, IBaseFilter *pDest);
void MyFreeMediaType(AM_MEDIA_TYPE& mt);

```

5. Examining webcamCap function

```

////////////////////////////////////
int webcamCap(HWND hwnd)
{
    HRESULT hr;

    // Initialize the COM library
    hr = CoInitialize(NULL);

    // Create the Filter Graph Manager.
    hr = CoCreateInstance(CLSID_FilterGraph, 0, CLSCTX_INPROC_SERVER,
        IID_IGraphBuilder, (void**)&pGraph);
    if (FAILED(hr)){
        AfxMessageBox("ERROR- Could not initialize capture graph builder");
        CoUninitialize();
        return hr;
    }

    // Using QueryInterface on the graph builder,
    // get the Media Control object.
    hr = pGraph->QueryInterface(IID_IMediaControl, (void **)&pControl);
    if (FAILED(hr))
    {
        AfxMessageBox("ERROR - Could not create the Media Control object.");
        pGraph->Release();
        CoUninitialize();
        return hr;
    }
}

```

```

// Now create the video input filter from the webcam
// and add it to the Filter Graph Manager
hr = GetVideoInputFilter(&pVideoInputFilter, L"Logitech");
if (SUCCEEDED(hr)) {
    pGraph->AddFilter(pVideoInputFilter, L"Webcam Video Capture");
} else {
    AfxMessageBox("ERROR - Could not create the video input filter.");
    pControl->Release ();
    pGraph->Release();
    CoUninitialize();
    return hr;
}

// Now create the ISampleGrabber Filter to grab the video frames
hr = CoCreateInstance(CLSID_SampleGrabber, NULL, CLSCTX_INPROC_SERVER,
    IID_IBaseFilter, (void*)&pGrabFilter);
if (FAILED(hr))
{
    AfxMessageBox("ERROR - Could not create the ISampleGrabber filter.");
    pControl->Release ();
    pVideoInputFilter->Release ();
    pGraph->Release();
    CoUninitialize();
    return hr;
}

// Now create the ISampleGrabber interface
hr = pGrabFilter->QueryInterface(IID_ISampleGrabber, (void*)&pSampleGrabber);
if (FAILED(hr))
{
    AfxMessageBox("ERROR - Could not create the SampleGrabber interface.");
    pControl->Release ();
    pGrabFilter->Release ();
    pVideoInputFilter->Release ();
    pGraph->Release();
    CoUninitialize();
    return hr;
}

// Now Set the working mode of this ISampleGrabber
// Specify what media type to process, to make sure it is connected correctly
ZeroMemory(&mt, sizeof(AM_MEDIA_TYPE));
mt.majorType = MEDIATYPE_Video;
mt.subtype = MEDIASUBTYPE_RGB24;
mt.FormatType = FORMAT_VideoInfo;
hr = pSampleGrabber->SetMediaType(&mt);

// Set working mode as continuous with no buffer
pSampleGrabber->SetOneShot(FALSE);
pSampleGrabber->SetBufferSamples(FALSE);

// Add the filter to the graph and connect it
pGraph->AddFilter(pGrabFilter, L"Grabber");
hr = ConnectFilters(pGraph, pVideoInputFilter, pGrabFilter); //
if (FAILED(hr))
{
    AfxMessageBox("ERROR - Could not connect two filters ");
    pControl->Release ();
    pSampleGrabber->Release();
    pGrabFilter->Release ();
    pVideoInputFilter->Release();
    pGraph->Release();
    CoUninitialize();
    return hr;
}

// Set up the callback interface of this grabber
pSampleGrabber->SetCallback(&g_StillCapCB, 0);
// Set the media type (needed to get the bmp header info)
pSampleGrabber->GetConnectedMediaType(&mt);

```

```

// Find an output pin on the pGrabFilter.
// and render it
IPin *pOut = 0;
GetUnconnectedPin(pGrabFilter, PINDIR_OUTPUT, &pOut);
hr=pGraph->Render (pOut);
pOut->Release();
if (FAILED(hr)) {
    AfxMessageBox("ERROR - Could not render the sample grabber filter ");
    pControl->Release ();
    pSampleGrabber->Release();
    pGrabFilter->Release ();
    pVideoInputFilter->Release();
    pGraph->Release();
    CoUninitialize();
    return hr;
}

// Send the video to main window
IVideoWindow *pVidWin = NULL;
pGraph->QueryInterface(IID_IVideoWindow, (void **)&pVidWin);
pVidWin->put_Owner((OAHWND)hwnd);
pVidWin->put_WindowStyle(WS_CHILD | WS_CLIPSIBLINGS);
RECT rc;
GetClientRect(hwnd,&rc);
pVidWin->SetWindowPosition (0,0,rc.right-100,rc.bottom );

// Run the graph
hr = pControl->Run();
return hr;
}

```

Before COM can be used within a DirectShow application, the COM facilities must be initialized.



Now, we create the filter graph manager. The filter graph manager implements an interface that enables an application to build a filter graph.

Get the media control object by using QueryInterface on the graph builder. This media control interface enables the graph to be run, paused and stopped.

Create the video input filter from the webcam by using *GetVideoInputFilter* function and add it to the filter graph.

```

////////////////////////////////////
// Enumerate all of the video input devices
// Return the filter with a matching friendly name
HRESULT GetVideoInputFilter(IBaseFilter** gottaFilter, wchar_t* matchName)
{
    BOOL done = false;

    // Create the System Device Enumerator.
    ICreateDevEnum *pSysDevEnum = NULL;
    HRESULT hr = CoCreateInstance(CLSID_SystemDeviceEnum, NULL, CLSCTX_INPROC_SERVER,
        IID_ICreateDevEnum, (void **)&pSysDevEnum);
    if (FAILED(hr))
    {
        return hr;
    }

    // Obtain a class enumerator for the video input category.
    IEnumMoniker *pEnumCat = NULL;
    hr = pSysDevEnum->CreateClassEnumerator(CLSID_VideoInputDeviceCategory, &pEnumCat, 0);

    if (hr == S_OK)
    {
        // Enumerate the monikers.
        IMoniker *pMoniker = NULL;
        ULONG cFetched;
        while ((pEnumCat->Next(1, &pMoniker, &cFetched) == S_OK) && (!done))
        {
            // Bind the first moniker to an object
            IPropertyBag *pPropBag;
            hr = pMoniker->BindToStorage(0, 0, IID_IPropertyBag,
                (void **)&pPropBag);
            if (SUCCEEDED(hr))
            {
                // To retrieve the filter's friendly name, do the following:
                VARIANT varName;
                VariantInit(&varName);
                hr = pPropBag->Read(L"_FRIENDLY_NAME", &varName, 0);
                if (SUCCEEDED(hr))
                {
                    wprintf(L"Testing Video Input Device: %s\n", varName.bstrVal);

                    // Do a comparison, find out if it's the right one
                    if (wcsncmp(varName.bstrVal, matchName,
                        wcslen(matchName)) == 0) {

                        // We found it, so send it back to the caller
                        hr = pMoniker->BindToObject(NULL, NULL, IID_IBaseFilter, (void**) gottaFilter);
                        done = true;
                    }
                }
                VariantClear(&varName);
                pPropBag->Release();
            }
            pMoniker->Release();
        }
        pEnumCat->Release();
    }
    pSysDevEnum->Release();
    if (done) {
        return hr; // found it, return native error
    } else {
        return UFW_E_NOT_FOUND; // didn't find it error
    }
}

```

This function creates the system device enumerator and obtains a class enumerator for the video input category. It enumerates all the monikers and binds each moniker to each object. It retrieves the filter's friendly name from the each object and does a comparison with friendly filter's name assigned by user. If found it, then send it back to the caller. Otherwise, it returns error message.

After adding video input filter to the filter graph, we add ISampleGrabber filter to the filter graph. ISampleGrabber filter would call a function as a callback on every frame. The ISampleGrabber interface is queried from ISampleGrabber filter. We

specify video RGB24 media type and set working mode as continuous with no buffer through ISampleGrabber interface. The *SampleCB* callback function is set through ISampleGrabber interface.

```

/
// The frame callback function called on each frame
// We can add the processing code in this function
STDMETHODIMP SampleCB(double SampleTime, IMediaSample *pSample){
    // get current media type
    VIDEOINFOHEADER *pvi = (VIDEOINFOHEADER *) mt.pbFormat;
    BYTE *pData;           // Pointer to the actual image buffer
    long lDataLen;         // Holds length of any given sample
    int iPixel;            // Used to loop through the image pixels
    pSample->GetPointer(&pData);
    lDataLen = pSample->GetSize();
    // Get the image properties from the BITMAPINFOHEADER
    int iPixelSize = pvi->bmiHeader.biBitCount / 8;
    int cxImage = pvi->bmiHeader.biWidth;
    int cyImage = pvi->bmiHeader.biHeight;
    int cbImage = cyImage * cxImage * iPixelSize;
    int numPixels = cxImage * cyImage;
    BYTE *prgb = (BYTE*) pData;
    // convert to grayscale image
    for (iPixel=0; iPixel < numPixels; iPixel++, prgb+=iPixelSize) {
        *(prgb + 1) = *(prgb); // G channel
        *(prgb + 2) = *(prgb); // R channel
    }
    return 0;
}

```

This function converts color image to grayscale image and sends it back to caller.

After adding video input filter and ISampleGrabber filter, we connect both filters by using *ConnectFilters* function.

```

////////////////////////////////////
HRESULT ConnectFilters(
    IGraphBuilder *pGraph,
    IBaseFilter *pSrc,
    IBaseFilter *pDest)
{
    if ((pGraph == NULL) || (pSrc == NULL) || (pDest == NULL))
    {
        return E_POINTER;
    }

    // Find an output pin on the first filter.
    IPin *pOut = 0;
    HRESULT hr = GetUnconnectedPin(pSrc, PINDIR_OUTPUT, &pOut);
    if (FAILED(hr))
    {
        AfxMessageBox("Fail get unconnected pin ");
        return hr;
    }
}

```

```

// Find an input pin on the downstream filter.
IPin *pIn = 0;
hr = GetUnconnectedPin(pDest, PINDIR_INPUT, &pIn);
if (FAILED(hr))
{
    AfxMessageBox("Fail get unconnected pin ");
    return hr;
}
// Try to connect them.
hr = pGraph->Connect(pOut, pIn);

pIn->Release();
pOut->Release();
return hr;
}

```

This function finds a first output pin on the source filter and a first input pin on the downstream filter by using *GetUnconnectedPin* function. Finally, we connect two filters by connect method of filter graph manager.

```

|
|////////////////////////////////////|
HRESULT GetUnconnectedPin(
    IBaseFilter *pFilter, // Pointer to the filter.
    PIN_DIRECTION PinDir, // Direction of the pin to find.
    IPin **ppPin) // Receives a pointer to the pin.
{
    *ppPin = 0;
    IEnumPins *pEnum = 0;
    IPin *pPin = 0;
    HRESULT hr = pFilter->EnumPins(&pEnum);
    if (FAILED(hr))
    {
        return hr;
    }
    while (pEnum->Next(1, &pPin, NULL) == S_OK)
    {
        PIN_DIRECTION ThisPinDir;
        pPin->QueryDirection(&ThisPinDir);
        if (ThisPinDir == PinDir)
        {
            IPin *pTmp = 0;
            hr = pPin->ConnectedTo(&pTmp);
            if (SUCCEEDED(hr)) // Already connected, not the pin we want.
            {
                pTmp->Release();
            }
            else // Unconnected, this is the pin we want.
            {
                pEnum->Release();
                *ppPin = pPin;
                return S_OK;
            }
        }
        pPin->Release();
    }
    pEnum->Release();
    // Did not find a matching pin.
    return E_FAIL;
}

```

This function finds a first unconnected input/output pin on the filter depended on PinDir's parameter.

After connecting video input filter and ISampleGrabber filter, we find an output pin on the ISampleGrabber filter and render it.

Finally, we send the video to the main window and run the graph.

6. Examining releaseWebcamCap function

```
////////////////////////////////////  
int releaseWebcamCap(){  
    HRESULT hr;  
    hr = pControl->Stop();  
    pProxyFilter->Release();  
    pVideoInputFilter->Release();  
    pControl->Release ();  
    pGraph->Release ();  
    CoUninitialize();  
    return hr;  
}
```

This function releases all the things and cleans up.

7. Examining OnOK and OnCancel functions

```
void CDX_CU2D1g::OnOK()  
{  
    HRESULT hr;  
    // Start the webcam  
    hr=webcamCap(this->m_hWnd);  
    if (FAILED(hr)) CDialoq::OnCancel();  
}  
  
void CDX_CU2D1g::OnCancel()  
{  
    // Release the webcam  
    releaseWebcamCap();  
    CDialoq::OnCancel();  
}
```



The OnOK function calls the webcamCap function to implement the webcam capture operation. The OnCancel function calls the releaseWebcamCap function to release all global variables and exit the main program.

6.2. Simple Capturing Video from A Webcam with OpenCV

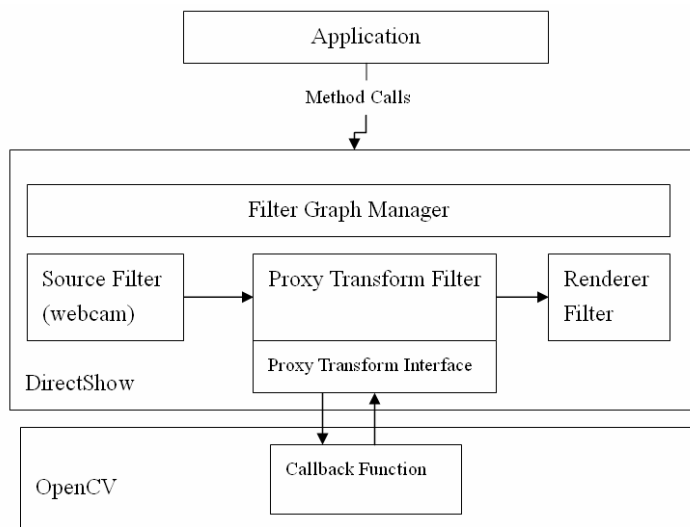


Fig 6-2 The block diagram of video capturing from webcam by using DirectShow and OpenCV

This application captures video inputs from a Logitech webcam, processes the image sequence by using OpenCV, and renders it.

1. Create a dialog-based application

This application can easily be created using the MFC application wizard. The name of the application is DX_CV2. VC++ should create a simple OK/Cancel Dialog for you. The class with a name ending by Dlg will contain the member functions that control the widget of the dialog.

2. Include header files

Add the following header file to the DX_CV2Dlg.cpp file

```

////////////////////////////////////
#include <DShow.h>           // include DirectShow interface
#include "cv.h"             // include core library interface
#include <initguid.h>       // include it for using ProxyTransform filter
#include "iProxyTrans.h"    // include ProxyTransform filter interface
#include "ProxyTransuids.h" // include ProxyTransform filter GUID

```

3. Declare global variables

Declare the following global variables in the DX_CV2Dlg.cpp file

```

IGraphBuilder *pGraph;           // Graph builder object
IMediaControl *pControl = NULL; // Media control object
IBaseFilter *pVideoInputFilter = NULL; // Video Capture filter
IBaseFilter* pProxyFilter = NULL; // Proxy Transform filter

```

4. Declare Function Prototype

Declare the following function prototypes in the DX_CV2Dlg.cpp file

```

// Function Prototype
int webcamCap(HWND hwnd);
int releaseWebcamCap();
HRESULT GetVideoInputFilter(IBaseFilter** gottaFilter, wchar_t* matchName);
HRESULT GetUnconnectedPin(IBaseFilter *pFilter, PIN_DIRECTION PinDir, IPin **ppPin);
HRESULT ConnectFilters(IGraphBuilder *pGraph, IBaseFilter *pSrc, IBaseFilter *pDest);
void process(void* img);

```

5. Examining webcamCap function

```

////////////////////////////////////
int webcamCap(HWND hwnd)
{
    HRESULT hr;

    // Initialize the COM library
    hr = CoInitialize(NULL);

    // Create the Filter Graph Manager.
    hr = CoCreateInstance(CLSID_FilterGraph, 0, CLSCTX_INPROC_SERVER,
        IID_IGraphBuilder, (void**)&pGraph);
    if (FAILED(hr)){
        AfxMessageBox("ERROR- Could not initialize capture graph builder");
        return hr;
    }

    // Using QueryInterface on the graph builder,
    // get the Media Control object.
    hr = pGraph->QueryInterface(IID_IMediaControl, (void **)&pControl);
    if (FAILED(hr))
    {
        AfxMessageBox("ERROR - Could not create the Media Control object.");
        pGraph->Release();
        CoUninitialize();
        return hr;
    }
}

```

```

// Now create the video input filter from the webcam
// and add it to the Filter Graph Manager
hr = GetVideoInputFilter(&pVideoInputFilter, L"Logitech");
if (SUCCEEDED(hr)) {
    pGraph->AddFilter(pVideoInputFilter, L"Webcam Video Capture");
}
else{
    AfxMessageBox("ERROR - Could not create the video input filter.");
    pControl->Release ();
    pGraph->Release();
    CoUninitialize();
    return hr;
}

// Create a proxy transform filter
// and add it to the Filter Graph Manager
IProxyTransform* pProxyTrans = NULL;
if(FAILED(CoCreateInstance(CLSID_ProxyTransform, NULL,CLSCTX_INPROC_SERVER,
    IID_IProxyTransform, (void**)&pProxyTrans))
|| !pProxyTrans)
{
    AfxMessageBox("ERROR - Could not create the proxy transform filter.");
    pVideoInputFilter->Release();
    pControl->Release ();
    pGraph->Release();
    CoUninitialize();
    return E_NOINTERFACE;
}
pProxyTrans->QueryInterface(IID_IBaseFilter, (void**)&pProxyFilter);
pGraph->AddFilter(pProxyFilter,L"Proxy Filter");

// Set the proxy callback
pProxyTrans->set_transform(process, 0);
pProxyTrans->Release();

// Connect video input filter and proxy transform filter
hr= ConnectFilters(pGraph, pVideoInputFilter, pProxyFilter);//
if (FAILED(hr)) {
    AfxMessageBox("ERROR - Could not connect two filters ");
    pProxyFilter->Release ();
    pVideoInputFilter->Release();
    pControl->Release ();
    pGraph->Release();
    CoUninitialize();
    return hr;
}

// Find an output pin on the proxy filter.
// and render it
IPin *pOut = 0;
GetUnconnectedPin(pProxyFilter, PINDIR_OUTPUT, &pOut);
hr=pGraph->Render (pOut);
pOut->Release();
if (FAILED(hr)) {
    AfxMessageBox("ERROR - Could not render the proxy filter ");
    pProxyFilter->Release ();
    pVideoInputFilter->Release();
    pControl->Release ();
    pGraph->Release();
    CoUninitialize();
    return hr;
}

// Send the video to main window
IVideoWindow *pVidWin = NULL;
pGraph->QueryInterface(IID_IVideoWindow, (void **)&pVidWin);
pVidWin->put_Owner((OAHWND)hwnd);
pVidWin->put_WindowStyle(WS_CHILD | WS_CLIPSIBLINGS);
RECT rc;
GetClientRect(hwnd,&rc);
pVidWin->SetWindowPosition (0,0,rc.right-100,rc.bottom );
pVidWin->Release ();

```

```
// Run the graph
hr = pControl->Run();
return hr;
}
```

Before COM can be used within a DirectShow application, the COM facilities must be initialized.

Now, we create the filter graph manager. The filter graph manager implements an interface that enables an application to build a filter graph.

Get the media control object by using QueryInterface on the graph builder. This media control interface enables the graph to be run, paused and stopped.

Create the video input filter from the webcam by using *GetVideoInputFilter* function and add it to the filter graph.

After adding video input filter to the filter graph, we add proxy transform filter created by OpenCV group to the filter graph. Proxy transform filter would call a function as a callback on every frame. The IProxyTransform interface is queried from proxy transform filter. The *process* callback function is set through IProxyTransform interface.

```
////////////////////////////////////
void process(void* img) {
    IplImage* image = reinterpret_cast<IplImage*>(img);
    cvErode( image, image, 0, 2 );
}
}
```

This function changes non-IplImage into IplImage data type. IplImage data type is an image data type supported by OpenCV group. Finally, it erodes image and sends it back to caller.

After adding video input filter and proxy transform filter, we connect both filters by using *ConnectFilters* function.

After connecting video input filter and proxy transform filter, we find an output pin on the proxy transform filter and render it.

Finally, we send the video to the main window and run the graph.

6. Examining releaseWebcamCap function

```
////////////////////////////////////  
int releaseWebcamCap(){  
    HRESULT hr;  
    hr = pControl->Stop();  
    pProxyFilter->Release();  
    pVideoInputFilter->Release();  
    pControl->Release ();  
    pGraph->Release ();  
    CoUninitialize();  
    return hr;  
}
```

This function releases all the things and cleans up.

7. Examining OnOK and OnCancel functions

```
void CDX_CU2D1g::OnOK()  
{  
    HRESULT hr;  
    // Start the webcam  
    hr=webcamCap(this->m_hWnd);  
    if (FAILED(hr)) CDialog::OnCancel();  
}  
  
void CDX_CU2D1g::OnCancel()  
{  
    // Release the webcam  
    releaseWebcamCap();  
    CDialog::OnCancel();  
}
```

The OnOK function calls the webcamCap function to implement the webcam capture operation. The OnCancel function calls the releaseWebcamCap function to release all global variables and exit the main program.

6.3. Simple Controlling PTZ Camera through RS232

This application controls Sony EVI D-70 PTZ camera through RS 232 by using VISCA protocol that developed by Sony.

1. Create a win32 console application

This application can easily be created using the Win32 Console application wizard. The name of the application is RS232_PTZ. VC++ should create a simple “Hello World” console application for you..

2. Include header files

Add the following header file to the RS232_PTZ.cpp file

```
#include <windows.h> //for serial communications device
```

3. Declare Function Prototype

Declare the following function prototypes in the RS232_PTZ.cpp file

```
void CameraOn(HANDLE* port);  
void CameraOff(HANDLE* port);  
void Home(HANDLE* port);  
void ADDRESS_SET(HANDLE* port);  
void IF_CLEAR(HANDLE* port);  
void sendCommand(HANDLE* port, BYTE *command);  
int getCom(HANDLE* port);  
void getAckAck(HANDLE* port);
```

4. Examining main function

```

int main(int argc, char* argv[])
{
    HANDLE comPort;
    DCB comPortSettings;
    COMMTIMEOUTS comPortTimeouts;

    // Open COM1 serial port with read and write access.
    // Access to the object cannot be shared
    // If the port was opened successfully the function returns
    // the handle comPort descriptor to work with further on.
    // If the port wasn't opened successfully the function will return INVALID_HANDLE_VALUE.
    comPort = CreateFile("COM1:", GENERIC_READ | GENERIC_WRITE,
        0, NULL, OPEN_EXISTING, 0, NULL);
    if (comPort == INVALID_HANDLE_VALUE) {
        printf("ERROR - Could not open com port\n");
        return 0;
    }

    // To set the size of the receiving and transmitting buffers.
    // Transmit buffer > 16
    // Receive buffer > ?
    if (!SetupComm(comPort, 40, 20)) {
        printf("ERROR - Could not set com port buffer sizes\n");
        return 0;
    }

    // set com port parameters
    if (!GetCommState(comPort, &comPortSettings)) {
        printf("ERROR - Could not get com port state\n");
        return 0;
    }
    comPortSettings.BaudRate = CBR_9600; // 9600 bps or 38400 bps
    comPortSettings.ByteSize = 8; //The number of bits in the bytes transmitted and received.
    comPortSettings.Parity = NOPARITY; // No parity scheme to be used
    comPortSettings.StopBits = ONESTOPBIT;
    comPortSettings.fAbortOnError = TRUE; // terminates all read and write operations
    // with an error status if an error occurs.
    if (!SetCommState(comPort, &comPortSettings)) {
        printf("ERROR - Could not set com port state\n");
        return 0;
    }

    // Clear the command buffer of the camera using broadcast function
    IF_CLEAR(&comPort);

    // VISCA can support a daisy chain of up to seven attached cameras.
    // Confirm the address for first time connection using broadcast function
    ADDRESS_SET(&comPort);

    // Turn on PTZ camera
    CameraOn(&comPort);

    // Set the PTZ camera in home position
    Home(&comPort);
    Sleep(10000);

    // Turn off PTZ camera
    CameraOff(&comPort);

    // close the port.
    CloseHandle(comPort);
    return 0;
}

```

First, we open COM1 serial port with read and write access. The access to the object cannot be shared by other applications. If the port was opened successfully the function returns the handle comPort descriptor to work with further on. Otherwise, returns INVALID_HANDLE_VALUE.

Next, we set the size of the receiving and transmitting buffers. The maximum transmit packet size is 16 bytes and the maximum receive packet size is unknown. We

assume that maximum receive packet size is 38 bytes. In order to guarantee the communication between two devices, the buffer size is larger than packet size.

After setting the buffer size of the com port, we set the baud rate as 9600 bps and the number of bits in the bytes transmitted and received as 8. No parity scheme to be used. The application terminates all read and write operation when error occurs. The number of stop bits to be used is one.

We clear the command buffer of the camera using *IF_CLEAR* function

```
,  
  
void IF_CLEAR(HANDLE* port)  
{  
    BYTE buf[] = {0x88, 0x01, 0x00, 0x01, 0xFF};  
    sendCommand(port,buf);  
}
```

This function sends the {0x88, 0x01, 0x00, 0x01, 0xFF} command to *sendCommand* function.

```
// Send a command and wait ACK or completion message from camera  
void sendCommand(HANDLE* port, BYTE *command)  
{  
    DWORD dummy;  
  
    WriteFile(*port, command, sizeof(command), &dummy, NULL);  
    FlushFileBuffers(*port);  
    getAckAck(port);  
  
    Sleep(100);  
}
```

This function sends the {0x88, 0x01, 0x00, 0x01, 0xFF} command to the PTZ by using *WriteFile* function. Then the transmit buffer will be flushed. Finally, the function waits ACK message and completion message from the PTZ through *getAckAck* function.

```
// Gets ACK message and Completion message  
void getAckAck(HANDLE* port)  
{  
    int packet = getCom(port);  
  
    while ((packet != 0xFF) && (packet != -1))  
        packet = getCom(port);  
}
```

This function reads a byte of data from camera through *getCom* function. This function leaves until get the ACK message, completion message, or fail to read data from the camera.

```

// Get a byte of data from the camera
int getCom(HANDLE* port)
{
    DWORD dummy;
    BYTE value;
    int rv;

    rv = ReadFile(*port, &value, 1, &dummy, NULL);
    if (rv == 0)
        return -1;
    else
        return (int)value;
}

```

This function reads a byte of data from camera through ReadFile function. If the ReadFile function successfully read a byte of data it returns value. Otherwise, returns -1.

```

void ADDRESS_SET(HANDLE* port)
{
    BYTE buf[] = {0x88, 0x30, 0x01, 0xFF};
    sendCommand(port,buf);
}
void Home(HANDLE* port)
{
    BYTE buf[] = {0x81, 0x01, 0x06, 0x04, 0xFF};
    sendCommand(port,buf);
}
void CameraOff(HANDLE* port)
{
    BYTE buf[] = {0x81, 0x01, 0x04, 0x00, 0x03, 0xFF};
    sendCommand(port,buf);
}
void CameraOn(HANDLE* port)
{
    BYTE buf[] = {0x81, 0x01, 0x04, 0x00, 0x02, 0xFF};
    sendCommand(port,buf);
}

```

If the *ADDRESS_SET*, *CameraOn*, *Home* and *CameraOff* functions are compared with *IF_CLEAR* function, the difference is the command only.

The *ADDRESS_SET* function finds the address of the camera for the first time connection by using broadcast method.

The PTZ camera can be turned on by using *CameraOn* function. The *Home* function sets the PTZ camera in home position.

Finally, the PTZ camera can be turned off by using *CameraOff* function and the port is closed by using *CloseHandle* function.