Face Detection: a simple approach

WANG Jia-ping (200228013202862)

Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100871, China

Abstract: This paper presents a simple method of detecting frontal faces on still image and estimating rotate angle. Based on vector similarity, I build a 24-class classifier to estimate face position and a Face/non-face classifier to finally verify a located frontal face. In additional I use image segmentation based on human skin color model to generate candidate face area by which greatly reduce the total detecting time.

1. Overview

The process of face detection on still image can divide into two stages: Preprocess and Searching. Preprocess stage product two result, an array of candidate face areas and a grayscale image of the original one for the succeed stage, Searching. At Searching stage, it search all candidate face areas in the grayscale image and mark the faces that found on the corresponded position in the original color image. Searching stage is relatively more time-consuming then Preprocess stage.

2. Detailed Process

2.1 Preprocess stage

Preprocess stage includes the following step:

1. Generate a binary map to represent skin/non-skin pixels. Initially the binary map contains black dot only. According a skin color model that trained beforehand, I scan all pixels in the original color image and set the corresponded dot in the binary map into a white dot if color of the pixel is human skin color.
2. Generate an array of rectangles of candidate skin areas. I use a method like clustering to generate a rectangle for every block of white dots in the binary map coming from step 1. The method does not need the pixels of the block to be tightly connected. Pixels can be incompact in some degree. After that, eliminate all rectangles that small than 19X19 (size of my average face template).
3. Convert the color image into a grayscale image. I straightforward scan pixels in the color image and set the bright value as (2R+G+B)/4 in every pixels of grayscale image.

2.2 Searching stage

Within every candidate rectangle, there may be a face in some place and in some size. Search stage is aimed to find it. I slide a 19X19 (the same size of my average face template) observing-window pixel-by-pixel and line-by-line in all sub-images of an image-pyramid build from the incoming grayscale image clip to check faces in all possible place and size. The image-pyramid contain sub-images of the original clip and every scaled down images from that one, with the scaling rate of 0.87, down to an image small than 19X19 (The last image is not included). For reducing memory cost, every layer of sub-image is temporarily generated, searched and then discard. Only the interested rectangle position is kept.

The image-pyramid is build as the following steps:

1. Keeping original proportion, if the greater of image width and image height is longer than 100 pixels, scale it into 100 pixels.

2. Then the image-pyramid contains sub-images of the original clip and every scaled image from that one, with the scaling rate of 0.87, down to an image small than 19X19 (The last one is not included).

In every time of observing, there is an evaluation of that the image in observing-window is a face or not. I record the maximum one while searching. After all sub-images are searched, the maximum one, if exist, is marked as a face.

The evaluation is generated as the follow steps:

1. Auto-level the image clip in the observing-window (it is somehow like histogram equation, but more fast). If the original level[1] is smaller than a valve (currently 100) kick the current window and continue to the next position.
2. Calculate edges of the auto-leveled image. If edge pixels sums less than a valve (currently 7000) kick the current window and continue to the next position.
3. No matter it is a face or not, estimate the rotate angle. If the estimation failed[2], kick the current window for it cannot be a face. Otherwise record the angle.
4. According to the angle estimation, choose corresponded average face template (I rotate the template in every 15 degree beforehand.) to compare with. It is to say evaluating similarity of the two vectors of 361 dimensions. If the similarity less than a valve (currently 230) kick it.
5. The similarity is the evaluation.

3. Detailed algorithm

3.1 Skin color model

Skin color model is used to separate color of human skin from the other.

Definition. Let

1) C be all natural numbers in [0,255].
2) X be the discrete vector space of $C \times C \times C$
3) F be a fuzzy set of $\{(m_A(x), x) | x \in X\}$

Training. Supports there are an array of vectors $V_{1\ldots N} \in X$ that every vector is of human skin color ($V_{1\ldots N}$ may contain same vectors) Then the membership function is

$$m_A(x) = \begin{cases} \dfrac{Count(x, V_{1\ldots N})}{N} & x \in \{V_{1\ldots N}\} \\ 0 & x \notin \{V_{1\ldots N}\} \end{cases}$$

The function ***Count(x, V)*** is defined as the number of elements that equal to x in the array of V.

Classify. Let *m* be the minimum value that confirms the following inequality.

$$\sum_{x \in \{y | m_A(y) > m\}} m_A(x) < 0.9$$

Then, $\forall x \in X$ if $m_A(x) > m$ x is color of human skin, otherwise not.

3.2 Rectangle clustering

After image segmentation, the result (a binary image) contains white dots that represent skin color. Rectangle clustering is used to generate some rectangles that encloses big block of white

[1] Defined in 3.3
[2] Defined in 3.7

dots to form candidate boxes for farther process (Searching).

The detail algorithm is stated as the following:

0. (Input)

   A binary image of $W \times H$: $M_{W \times H}$

1. (Initialization)

   1) An array of submitted rectangles, initially the length is zero, S
   2) An array of growing rectangles, initially the length is zero, G

2. (Forward Inclusion)

   For every line y in M: ($y \in [0,H-1]$)

   Scan line y form the left to the right, if, at the offset of x:

   1) Reach a line segment of the length equal to 8 that contains white dots then 6 and the line is not within any growing rectangles in G. Then create a rectangle that has top and bottom equal to y, left equal to x and right equal to x+8. Add this rectangle to G.
   2) Reach the leftedge of a growing rectangle (5 dots to the left of the rectangle) in G. If there are more then 2 white dots on the road to the rectangle. Grow it by setting the left parameter to x.
   3) Reach internal part of a growing rectangle in G.

      i. Check the bottom parameter of it: if y – bottom>6 then stop glowing the rectangle by move it from G to S. (The rectangle has not grew for a long time)

      ii. Count dots from the left to the right of the rectangle in line y: if there contains 1/32 white dot then glow it by setting the bottom parameter to y.
   4) Reach right edge of a growing rectangle (4 dots from the right of the rectangle) in G. If there are more then 2 white dots on road from the rectangle. Grow it by setting the right parameter into x.
   5) Reach a growing rectangle that is not far a way from the previous one (less then 6 dots), left to it. Combine the two rectangles into one.

   After all lines is scanned, move all rectangles in G to S.

   For every rectangle in S:

   Slightly enlarge the rectangle that encloses more than 1/16 white dot in its area.

3. (Conditional Exclusion)

   For every rectangle in S:

   1) Remove the rectangle that small then 19 X 19 (Width <19 or Height<19)
   2) Remove the rectangle that most area is enclosed by another rectangle.

4. (Output)

   All rectangles in S.

3.3 Grayscale image Auto-level

Auto-leveling an image is aimed to wipe out the difference of lighting and physical sampling.

Definition. Let,

1) $H_i \quad i \in [0,255]$ be the histogram of the input image $M_{X \times Y}$.

2) $A_{X \times Y}$ be the auto-leveled image.

3) V be the maximum value that confirming $\sum_{i=0}^{L} H_i \Big/ (X \cdot Y) < 0.013$

4) U be the minimum value that confirming $\sum_{i=U}^{255} H_i \Big/ (X \cdot Y) < 0.013$

5) Original level L is defined as $U - V$.

Then $A_{ij} = \begin{cases} 0 & M_{ij} < V \\ 255 \cdot (M_{ij} - V)/L & M_{ij} > V, M_{ij} < U \\ 255 & M_{ij} > U \end{cases}$

3.4 Sum edge pixels

Summing edge pixels is used to eliminate images that contain little detail.

Definition. Let, $M_{X \times Y}$ be the input image.

Then edge pixel summing S:

$$S = \sum_{i=0}^{y} \sum_{j=0}^{x} \max\left(\left| M_{i,j} - M_{(i+1),j} \right|, \left| M_{i,j} - M_{i,(j+1)} \right| \right)$$

3.5 Evaluate vector similarity

I define *similarity* of two vectors as the following:

Definition. Let

1) $A=(a_1, a_2 \ldots a_n)$ and $B=(b_1, b_2 \ldots b_n)$ to be vectors of n dimensions.
2) $P=(p_1, p_2 \ldots p_n)$ is a static vector of n dimension. P represents different weights of different components in calculating similarity. P is a statistical result of many vectors of same specific pattern. A more stable component has a greater p_i.

Then similarity S of A and B is

$$S = \left(\sum_{i=1}^{n} p_i (a_i - b_i) \right)^2 - \sum_{i=1}^{n} [p_i (a_i - b_i)]^2 .$$

3.6 Rotate feature extracting

As showed in figure 1, I divide circularity into 24 sectors and sum grayscale values in every sector to form the Rotating-Feature vector of 24 dimensions.

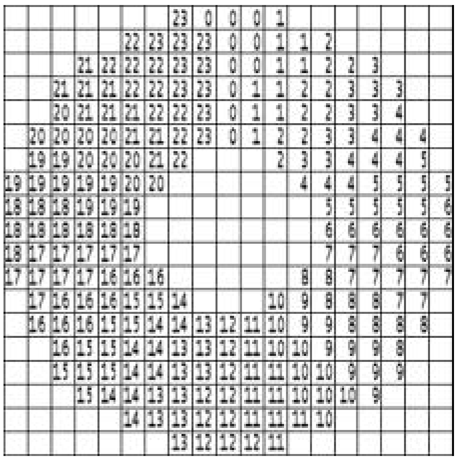
Take 19x19 image for example.

Definition. Let

1) P be the matrix that showed in figure1.
2) G be the input image clip to be extracted.

Then the result feature vector RF is:

$$rf_{(d+1)} = \sum_{p_{ij}=d} g_{ij} \qquad d \in [0,23]$$



							23	0	0	0	1							
					22	23	23	23	0	0	1	1	2					
			21	22	22	22	23	23	0	0	1	1	2	2	3			
		21	21	21	22	22	23	23	0	1	1	2	2	3	3	3		
		20	21	21	21	22	22	23	0	1	1	2	2	3	3	4		
	20	20	20	20	21	21	22	23	0	1	2	2	3	3	4	4	4	
	19	19	20	20	20	21	22				2	3	3	4	4	4	5	
19	19	19	19	19	20	20						4	4	4	5	5	5	5
18	18	18	19	19	19								5	5	5	5	5	6
18	18	18	18	18	18								6	6	6	6	6	6
18	17	17	17	17	17								7	7	7	6	6	6
17	17	17	17	16	16	16						8	8	7	7	7	7	7
	17	16	16	16	15	15	14				10	9	8	8	8	7	7	
	16	16	16	15	15	14	14	13	12	11	10	9	9	8	8	8	8	
		16	15	15	14	14	13	13	12	11	10	10	9	9	9	8		
		15	15	15	14	14	13	13	12	11	11	10	10	9	9	9		
			15	14	14	13	13	12	12	11	11	10	10	10	9			
					14	13	13	12	12	11	11	11	10					
							13	12	12	12	11							

Figure 1. Sector matrix for 19x19 image

3.7 24-class rotate angle classifier

The 24-class classifier, as showed in figure2, is used to estimate the rotate angle of an image clip of specific pattern. Every estimator handles a Rotating-Feature (RF) vector[3] extracted from the image clip and tells the final classifier the similarity between the incoming RF vector and the built-in average vector. The final classifier regards the maximum one as the rotate angle. So to build the 24 classifiers is to build the 24 built-in average vectors. I build them as the following:

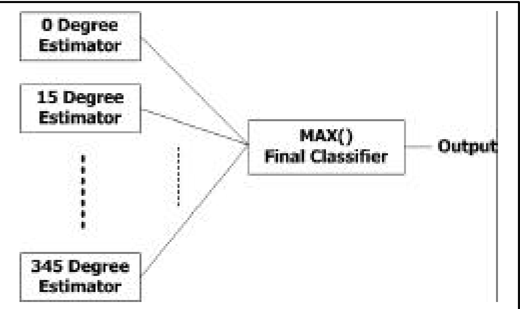


Figure 2. 24-class classifier

1) I sum all RF vectors generated from hundreds of non-rotated (0 degree) frontal face images and get a average RF vector (RF^0) and P vector (P^0) (mentioned in 3.5) for 0 degree estimator.
2) For other estimators (1 for 15 degree, 2 for 30 degree and so on) I round shifts the vectors:

   Let RF^i and P^i to be built-in RF and P vector for estimator i, then

$$\begin{cases} RF_j^i = RF_{j+1}^{i-1} & j \in [1,23] \\ RF_{24}^i = RF_1^{i-1} & Other \end{cases} \qquad \begin{cases} P_j^i = P_{j+1}^{i-1} & j \in [1,23] \\ P_{24}^i = P_1^{i-1} & Other \end{cases}$$

Finally, the classifier calculates every estimator and find the maximum one as the result.

3.8 Face/non-face classifier

The classifier is used to finally verify a frontal face. I use a 19X19 image clip as observing-window and I regard it as a vector of 361 dimensions. I use the method like in 3.7 step 1 to build an average frontal face vector and compare every input image with it. If the similarity is greater then a fix valve (currently 230), I regard the input image as a frontal face. This method is very fast but is not so robust. I am planning using other algorithm (such as SVM/Adaboost) instead of it in farther study.

4. Implementation

Based on previous project (Keyboard simulation), I use Visual C++.Net to develop all components. I use class CImage of ATL 7.0,based on GDI+, to read and write image data of BMP and JPEG format; class Matrix to perform affine transformation. All classes associated with face detection, implemented by me, are all in the namespace of **Vision**. In additional I add the following classes to the original project (Kernel) to maintain user interface for face detection.

1) CFaceDetPane

It is derived from CWnd of MFC. It is used to display the three panels on the PDA screen. It also creates a working thread for face detection and handles the mouse events for zooming.

2) CFaceDetProgress

It is a nested class of CFaceDetPane. It contains working state of face detection, via which the working thread is synchronized with UI.

3) CImgShowBox

It is derived from CDailog of MFC. It is used to display zoomed image on the panels in screen. Its instance is dynamically created by CFaceDetPane. It provides a function of saving the image that it displayed into a file of BMP or JPEG format.

There is two global functions newly added:

1) `UINT FD_Worker( LPVOID pParam )`, the function to executed in the working

[3] Defined in 3.6

thread. It performs the two stages of Preprocess & searching and keeps synchronization with UI.

2)
```
int HardTravel(Vision::CGreyScaleImage &Text,
               RECT * lpRet,
               Vision::CTemplate<OW_SIZE> Key[24],
               Vision::CTemplate<OW_SIZE> Mask[24],
               Vision::CRotateFeatureExtracter &RF_Ext,
               double * pMinDist = NULL)
```

The function is called by FD_Worker to search a possible frontal face in a candidate rectangle.

In Vision namespace, all classes are for digital image processing. Function FD_Worker and HardTravel use them. Figure 3 shows hierarchy of classes in the namespace.

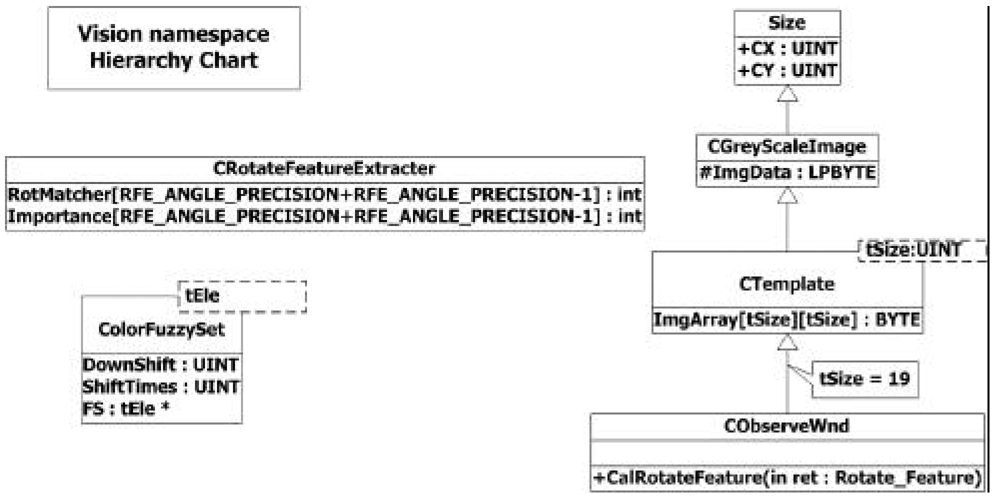

Vision namespace Hierarchy Chart
Size
+CX : UINT
+CY : UINT
CGreyScaleImage
#ImgData : LPBYTE
CRotateFeatureExtracter
RotMatcher[RFE_ANGLE_PRECISION+RFE_ANGLE_PRECISION-1] : int
Importance[RFE_ANGLE_PRECISION+RFE_ANGLE_PRECISION-1] : int
tSize:UINT
CTemplate
ImgArray[tSize][tSize] : BYTE
tEle
ColorFuzzySet
DownShift : UINT
ShiftTimes : UINT
FS : tEle *
tSize = 19
CObserveWnd
+CalRotateFeature(in ret : Rotate_Feature)

Figure 3. Class hierarchy of Vision (UML Class Diagram)

1) Class **Size** is a simple class that encapsulates width and height.
2) Class **CGreyScaleImage** is the core class of digital image processing. It implements:
   a) Import/Export from (or to) a Win32 HBITMAP handle.
   b) Affine transformation and re-sampling.
   c) Calculate image vector similarity.
   d) Grayscale image auto-leveling.
   e) Sum edge pixels.
   f) Image segmentation according to a given instance of class ColorFuzzySet.
   g) Rectangle cluster on a binary image.
3) Class **CTemplate** is derived from CGreyScaleImage. It handles image of fixed size only. It stores image pixels in memory statically allocated instead of dynamically. It is a template class. The fixed size is specified in template parameter.
4) Class **CObserveWnd** is derived from CTemplate with template parameter tSize=19. It implements calculating Rotating-Feature of a 19X19 image.
5) Class **CRotateFeatureExtracter** is used to recognize the rotate angle of a given Rotating-Feature. (Sorry for I misname it ^_^). It also provides a function for given an angle estimation of more precision by combine two output of maximum and second maximum estimator.
6) Class **ColorFuzzySet** represent a fuzzy set of 3D vector. It has the function of classifying a skin-color from other color.