**“Rock”-“Paper”-“Scissors” Identifier (June 2014)**

Tan Shuo and Shirley Qi

***Abstract-*Digital visual identifiers have wide applications from home front operations to consumer electronics. They can be used to ease visual communication in low-light situations, improve security, and involve the human sense of sight in games. Our product engages all three areas through identifying “rock”, “paper”, and “scissors” hand motions in an input picture. The algorithm pre-processes a bitmap format (BMP) image before identifying the hand gesture using a tree diagram. Although product only identifies a certain few gestures right now, it paves the way to more complex and reliable visual identifiers.**

***Index Terms-*Data preprocessing, digital circuits, digital filters, digital integrated circuits, digital signal processors, electrical engineering, filtering algorithms, image analysis, image denoising, image edge detection, image quality, image recognition, image sequence analysis, information filtering, machine vision, object detection, sensor systems.**

I. INTRODUCTION

*A. Societal Context*

T

HE concept of a digital visual identifier has been around for years. The “rock”-“paper”-“scissors” identifier is a simple program that only identifies three specific hand motions (the “rock”, “paper”, and “scissors” motions), but these three motions are applicable across multiple fields.

The identifier is most useful in communication, security, and in games. In communication, each hand gesture translates to a different predetermined message. Two people can exchange short messages with each other using this method in low-light situations and in situations in which traditional methods of communication are inappropriate so users have a need to be discreet.

Visual identifiers can be used as a second level of security clearance. Most security systems now involve typing passphrases. A visual identifier checks identity by identifying the predetermined hand gesture that the user puts up.

The third field that visual identifiers are highly applicable with is games. As the name suggests, this visual identifier works well in a “rock”-“paper”-“scissors” game. Implementing this technology in a robot opens up another field possibilities.

*B. Main Technical Issues*

An identifier works by identifying unique characteristics from each particular item. Unique characteristics from the same item will not look the same in every picture though. Different lighting and background noise will distort the object. It is hard to correctly identify objects that can change minutely in infinitely many ways.

An identifier that identifies many objects does not identify objects as well as one that identifies only a few objects. The objects may have similar characteristics that the identifier has problems distinguishing between.

To correctly get a good visual, the picture should have relatively high quality. A higher quality picture takes up more space in memory though, and takes a long time to process.

*C. Report Outline*

I. Introduction

*A. Societal Context*

*B. Main Technical Issues*

*C. Report Outline*

II. Technical Background

*A. Theory*

*B. Standards Employed*

III. Design Description

*A. Design to Constraints*

*B. Approach/Justification*

IV. Testing/Validation

*A. Experimental Procedures*

*B. Results*

V. Conclusion

*A. Successes/Non-successes*

*B. Future Products/Publications*

VI. References/Credits

II. TECHNICAL BACKGROUND

*A. Theory*

Our algorithm is divided into two separate parts: pre-processing the image, and then identifying the processed image. Identifying the image after pre-processing it increases the accuracy of the identification of the picture. Pre-processing the image includes filtering and finding the connected graph of the picture. The transformations we used were (in order): two-dimensional median filter, RGB colorspace to YDbDr colorspace mapping, binary color graph, erode, dilate, and bwareaopen.

Two-dimensional median filter: This algorithm blurs the picture by setting the value of each point to the mathematical average of the 4 points around it (top, right, bottom, left) [1].

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |
|  | 5 | 5 | 5 |  |
|  | 5 | 1 | 5 |  |
|  | 5 | 5 | 5 |  |
|  |  |  |  |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |
|  | 5 | 5 | 5 |  |
|  | 5 | 5 | 5 |  |
|  | 5 | 5 | 5 |  |
|  |  |  |  |  |

Fig 1. Median filter.

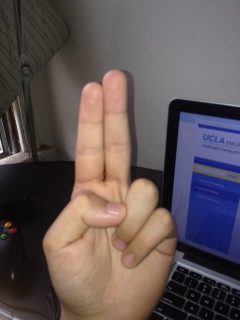
 

Fig 2. Effects of the filter on actual pictures.

RGB colorspace to YDbDr colorspace mapping: Each cell of a color picture matrix has three values corresponding to the amount of red, green, and blue used on that point in the picture. Mapping to YDbDr colorspace consists of taking the red, green, and blue values of each point and converting them to values for Y, Db, and Dr values using the following equation[2].

begin{align}
Y   &= +0.299 R +0.587 G +0.114 B\\
D_B &= -0.450 R -0.883 G +1.333 B\\
D_R &= -1.333 R +1.116 G +0.217B\\
\begin{bmatrix} Y \\ D_B \\ D_R \end{bmatrix} &=
\begin{bmatrix} 0.299 & 0.587 & 0.114 \\ 
-0.450 & -0.883 & 1.333 \\ 
-1.333 & 1.116 & 0.217 \end{bmatrix}
\begin{bmatrix} R \\ G \\ B \end{bmatrix}\end{align}

Fig 3. Equation mapping RGB colorspace to RDbDr colorspace.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 3 | 0 |
| 0 | 0 | 0 | 3 | 3 | 3 |
| 0 | 0 | 3 | 3 | 3 | 3 |

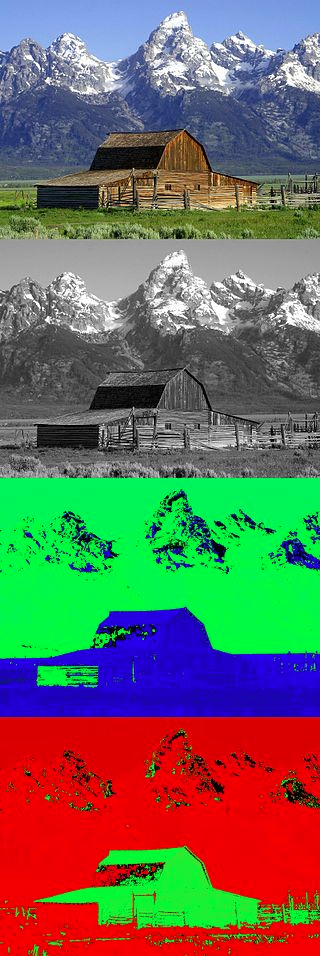
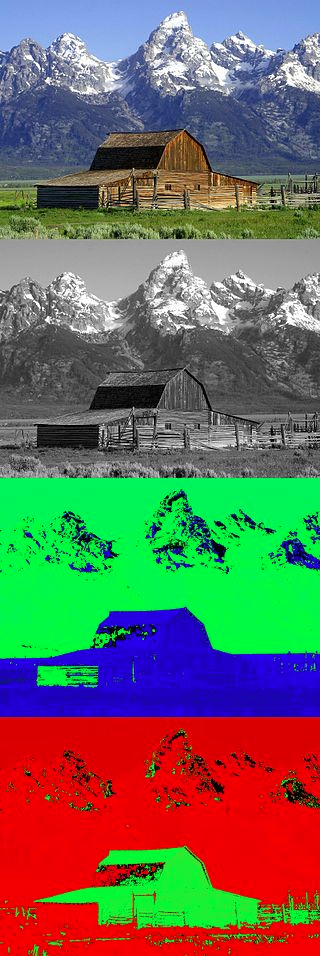
 

Fig 4. Picture mapped from RGB to Y space (top right), Db space (right middle), and Dr space (right bottom) [3].

binary color graph: changes picture from color to black and white.

erode(): This function reduces the size of a white connected graph by changing every edge cell of the connected graph to black [4]. This is useful for edge clean-up and eliminating small white dots.

dilate(): This function increases the size of a white connected graph by changing every cell touching an edge cell of the connected graph to white [4]. This function is the sister function to erode and eliminates small black dots and enlarges small white dots.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 1 | 1 | 1 | 1 | 1 | 1 |
| 1 | 1 | 1 | 1 | 1 | 1 |
| 1 | 1 | 0 | 1 | 1 | 1 |
| 1 | 1 | 1 | 1 | 1 | 1 |
| 1 | 1 | 1 | 1 | 1 | 1 |
| 1 | 1 | 1 | 1 | 1 | 1 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 1 | 1 | 1 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 | 1 | 1 |
| 1 | 0 | 0 | 0 | 1 | 1 |
| 1 | 0 | 0 | 0 | 1 | 1 |
| 1 | 1 | 1 | 1 | 1 | 1 |
| 1 | 1 | 1 | 1 | 1 | 1 |

Fig 5. Erode function.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 1 | 1 | 0 | 0 | 0 |
| 1 | 1 | 1 | 0 | 0 | 0 |
| 1 | 1 | 1 | 0 | 1 | 1 |
| 0 | 0 | 0 | 0 | 1 | 1 |
| 0 | 0 | 0 | 0 | 1 | 1 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 0 | 0 | 0 | 0 |

Fig 6. Dilate Function

8-connected neighborhood bwareaopen() function: This function goes through the image array and finds a point in every white connected graph. Then, it flags each point as a different number and finds the area of each connected graph. Finally, it fills every white connected graph except the largest one, which it leaves white [5].

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 1 | 0 | 0 | 0 | 0 |
| 1 | 1 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 0 | 1 | 1 | 1 |
| 0 | 0 | 1 | 1 | 1 | 1 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 2 | 0 | 0 | 0 | 0 |
| 2 | 2 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 3 | 0 |
| 0 | 0 | 0 | 3 | 3 | 3 |
| 0 | 0 | 3 | 3 | 3 | 3 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 0 | 1 | 1 | 1 |
| 0 | 0 | 1 | 1 | 1 | 1 |

Fig 6. Bwareaopen function.

The higher resolution the picture is, the more correct the identification is. The higher the resolution, the slower it takes for the DSK to process the picture. One of the tradeoffs we had was whether to reduce the picture size to optimize speed. We finally settled with a picture size of about 240x320. The DSK can finish processing this picture within a few seconds, with relatively high accuracy. This size also works the best with the memory size of the DSK.

*B. Standards Employed*

Input images in 24-bit bitmap image format (BMP). This report is written in the December 2013 revision of IEEE Transactions format [6]. Code Composer Studio is connected to the DSK 6416 with standard embedded JTAG support via USB connection.

III. DESIGN DESCRIPTION

*A. Design to Constraints*

The ideal input picture is a high quality picture of the gesture that has a plain white background. The hand is the brightest part of the picture and is upright. The identifier would identify the motion and give results without any delay. This would require a large memory to store the input picture. It would also need an extremely fast processor. The picture would most likely have to be edited or professionally taken in a specialized room.

These constraints are not pragmatic. It would uneconomic to use high-end machines for this task. We used the DSK 6416, Code Composer Studio and MATLAB as development environments. The DSK does not have enough memory for a high quality photo, so we used a 240x320 photo. It is difficult to constantly edit photos or take professional photos, so instead of having a specific environment in which the photos would need to be taken, we would use pre-processing code to “clean” the photos that do not have completely clean backgrounds. It would also take a long time to go through a very high quality picture, which is also not practical.

*B. Approach/Justification*

We are trying to be as consumer-oriented as possible though. The consumer should be able to use this product anywhere and anytime. Chaos in the background should affect the product as minimally as possible. The time it takes for the algorithm to run should be at most a minute. They do not have to buy any materials other than an inexpensive camera (phone cameras will work), and do not have to go to a certain location for shooting photos.

As a result, our design has two parts. First, editing the environment around the hand; second, identifying the gesture. The hand still must be upright in the picture, but that does not pose a significant economic or environmental hurdle. The ideal photo size is 240x320, but sizes similar to that will work.

First, we apply a median filter to the given bitmap format image (BMP image) to clear small noise. This speeds up the bwareaopen() function later. Then, we map the image from the RGB colorspace to the YDbDr colorspace. This colorspace is not as affected by changes in lighting. Using thresholds in the Db and Dr colorspaces, we determined which parts were the hand and which parts were background noise.

Then, we map the image to a binary color graph and erode() and dilate() the image. These erase small white holes and small black holes respectively. Together, they clean the boundary and plug all smaller holes.

Next, we use the bwareaopen() function in the 8-connected neighborhood form to plug every white connected graph except the largest white connected graph. The remaining white connected graph is the shape of the hand.

Next, we start to identify the picture. We first identify the widest point of the hand. Anything above the widest horizontal line is the upper half of the hand, and anything below and is the lower half. We define *length* of the hand is the length from the tip of the longest finger to the line separating top half from lower half and *width* of the hand as the width of the hand at the line separating top half from lower half.



Top Half

Bottom Half

Fig 7. The widest part of the hand divides the top half and the bottom half of the hand.



Fig 8. The length and width of the hand are defined as seen in the figure.

First, we take the *ratio* of the width of the hand to the length of the hand. Then, we go through the actual heights of each point from left to right. If the actual height is less than half of the longest height, then change the actual height to 0. If it is more than half of the longest height, increase the actual height to the largest height. This creates *segments*, which are separate blocks of the hand when looking only at the top half.

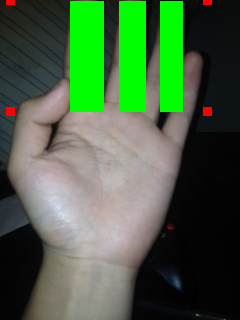
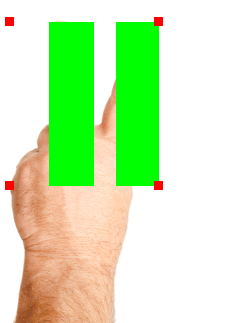
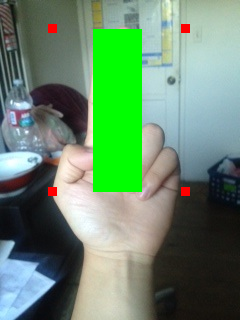
  

Fig 9. Examples of different numbers of segments.

The *fill %* of the hand is the % of the rectangle that just encloses the top half of the hand that is filled by the segments.



Fig 10. Rectangle that encloses the top half of the hand. The % fill is a % of this rectangle.

These are the components that form the identification tree that is shown below.

Fig 2. Identification tree.

IV. TESTING/VALIDATION

*A. Experimental Procedures*

To test the algorithm, we took pictures in many different situations, and sent them through the identifier. Then, we compared the result with the gesture in the original picture. There were the following pictures:

Lighting: Bright afternoon sunlight

Even outdoor light

Uneven indoor light (bright spots)

Even indoor light

Yellow background (similar to skin tone)

Background: Bright computer screen

In lab with machines/knobs

Bedroom in background

Colorful background

Hand texture: Nail polish

Hair on hand

Thin female hand

Thick female hand

Male hand

Hand motion: Vertical hand

Slightly tilted hand

Twisted hand, so side of hand is showing

Palm of hand facing camera

Back of hand facing camera

*B. Results*

Lighting makes a huge difference. If there is anything somewhat large (finger sized) in the background that is as bright as the hand itself, it may be mistaken for the hand. If anything has a similar hue as the hand itself, it may be mistaken as well. It is best to use flash in the picture to clearly identify the hand in the foreground of the picture. Small bits of colorful material or having many things in the background do not change the picture very much. The hand may be read incorrectly if it is slightly tilted. We did not find any other situations that significantly affected the results.

V. CONCLUSION

*A. Successes/Non-successes*

The algorithm we wrote works with pictures that do not have a background very similar to the hand color. It also works with the limited memory space in the DSK. It works the best when the photo has been taken with flash. The only non-success we had was that we could not make the identifier a real-time identifier, because we do not have the correct equipment to work with.

*B. Future Products/Publications*

In the future, we may look into using a camera to turn it into a real-time identifier. We may also expand the identification to other hand gestures, or to identifying other body parts.

VI. APPENDIX

#ifndef ALG

#define ALG

#define MAX\_CONNECTED\_GRAPH 255

#include "m\_mem.h"

#include "imatrix.h"

/\* stack, used for graph expanding \*/

typedef struct stack

{

axis\_t data;

struct stack\* next;

struct stack\* pre;

} stack\_t;

stack\_t\* table = NULL;

stack\_t\* current = NULL;

stack\_t\* previous = NULL;

unsigned int count = 0; /\* how many pixels in the stack \*/

axis\_t temp; /\* the temporary pixel poped out from the stack \*/

int l0 = 0; /\* the top boundry line of the hand we want to focus on \*/

int l1 = 0; /\* the bot boundry line of the hand we want to focus on \*/

int l2 = 0; /\* twist line, never used \*/

int h0 = 0; /\* never used \*/

int h1 = 0; /\* never used \*/

int n0 = 0; /\* the left boundry line of the hand we want to focus on \*/

int n1 = 0; /\* the right boundry line of the hand we want to focus on \*/

/\* statics on areas of connected graphs in a binary image \*/

unsigned int stat[MAX\_CONNECTED\_GRAPH];

/\* parameters for the identifier \*/

int seg\_num = 0;

double ratio = 0.0;

double percent = 0.0;

/\* print out the matrix \*/

void debug(imt\_c\* m);

void push(unsigned int row,unsigned int col)

{

if(table == NULL)

{

table = (stack\_t\*)s\_malloc(sizeof(stack\_t));

table->data.row = row;

table->data.col = col;

table->next = NULL;

table->pre = NULL;

current = table;

count ++;

}

else

{

current->next = (stack\_t\*)s\_malloc(sizeof(stack\_t));

current->next->data.row = row;

current->next->data.col = col;

current->next->next = NULL;

current->next->pre = current;

current = current->next;

count ++;

}

}

void pop()

{

if(table == NULL)

{

return;

}

if(table == current)

{

temp.row = current->data.row;

temp.col = current->data.col;

s\_free();

table = NULL;

count--;

}

else

{

temp.row = current->data.row;

temp.col = current->data.col;

current = current->pre;

s\_free();

current->next = NULL;

count --;

}

}

/\* median filter \*/

void medfilt2(imt\_c\* m,imt\_c\* result)

{

int i = 0;

int j = 0;

double temp = 0.0;

temp = ((double)get\_pixel\_c(m,1,2) + (double)get\_pixel\_c(m,2,2) + (double)get\_pixel\_c(m,2,1))/3;

set\_pixel\_c(result,1,1,temp);

temp = ((double)get\_pixel\_c(m,1,m->col - 1) + (double)get\_pixel\_c(m,2,m->col - 1) + (double)get\_pixel\_c(m,2,m->col))/3;

set\_pixel\_c(result,1,m->col,temp);

temp = ((double)get\_pixel\_c(m,m->row,2) + (double)get\_pixel\_c(m,m->row - 1,2) + (double)get\_pixel\_c(m,m->row - 1,1))/3;

set\_pixel\_c(result,m->row,1,temp);

temp = ((double)get\_pixel\_c(m,m->row,m->col - 1) + (double)get\_pixel\_c(m,m->row - 1,m->col) + (double)get\_pixel\_c(m,m->row - 1,m->col - 1))/3;

set\_pixel\_c(result,m->row,m->col,temp);

for(i = 2;i < m->col;i ++)

{

temp = ((double)get\_pixel\_c(m,1,i - 1) + (double)get\_pixel\_c(m,1,i + 1) + (double)get\_pixel\_c(m,2,i) + (double)get\_pixel\_c(m,2,i + 1) + (double)get\_pixel\_c(m,2,i - 1))/5;

set\_pixel\_c(result,1,i,temp);

temp = ((double)get\_pixel\_c(m,m->row,i - 1) + (double)get\_pixel\_c(m,m->row,i + 1) + (double)get\_pixel\_c(m,m->row - 1,i - 1) + (double)get\_pixel\_c(m,m->row - 1,i) + (double)get\_pixel\_c(m,m->row - 1,i + 1))/5;

set\_pixel\_c(result,m->row,i,temp);

}

for(i = 2;i < m->row;i ++)

{

temp = ((double)get\_pixel\_c(m,i - 1,1) + (double)get\_pixel\_c(m,i + 1,1) + (double)get\_pixel\_c(m,i - 1,2) + (double)get\_pixel\_c(m,i,2) + (double)get\_pixel\_c(m,i + 1,2))/5;

set\_pixel\_c(result,i,1,temp);

temp = ((double)get\_pixel\_c(m,i - 1,m->col) + (double)get\_pixel\_c(m,i + 1,m->col) + (double)get\_pixel\_c(m,i - 1,m->col - 1) + (double)get\_pixel\_c(m,i,m->col - 1) + (double)get\_pixel\_c(m,i + 1,m->col - 1))/5;

set\_pixel\_c(result,i,m->col,temp);

}

for(i = 2;i < m->row;i ++)

for(j = 2;j < m->col;j ++)

{

temp = (double)get\_pixel\_c(m,i - 1,j - 1) + (double)get\_pixel\_c(m,i,j - 1) + (double)get\_pixel\_c(m,i + 1,j - 1) + (double)get\_pixel\_c(m,i - 1,j) + (double)get\_pixel\_c(m,i + 1,j) + (double)get\_pixel\_c(m,i - 1,j + 1) + (double)get\_pixel\_c(m,i,j + 1) + (double)get\_pixel\_c(m,i + 1,j + 1);

temp = temp/8;

set\_pixel\_c(result,i,j,temp);

}

}

/\* erode the image by the strel marix \*/

imt\_c\* imrode(imt\_c\* m,imt\_c\* strel)

{

int i;

int j;

imt\_c\* temp;

imt\_c\* result;

result = create\_zero\_c(m->row,m->col);

temp = create\_zero\_c(strel->row,strel->col);

for(i = 2;i <= m->row - 1;i ++)

for(j = 2;j <= m->col - 1;j ++)

{

/\* get the sub matrix concentrate on a pixel \*/

sub\_mt\_fast\_c(m,temp,i - 1,i - 2 + strel->row,j - 1,j - 2 + strel->col);

/\* set this pixel to be 255, if equal to the strel \*/

if(is\_equal\_c(temp,strel))

{

set\_pixel\_c(result,i,j,255);

}

}

return result;

}

/\* dilate the image by the strel marix \*/

imt\_c\* imdilate(imt\_c\* m,imt\_c\* strel)

{

int i;

int j;

int k;

int l;

imt\_c\* result;

result = create\_zero\_c(m->row,m->col);

/\* replace a pixel by the strel \*/

for(i = 2;i <= m->row - 1;i ++)

for(j = 2;j <= m->col - 1;j ++)

{

if(get\_pixel\_c(m,i,j) == 255)

{

for(k = i - 1;k <= i - 2 + strel->row;k ++)

for(l = j - 1;l <= j - 2 + strel->col;l ++)

set\_pixel\_c(result,k,l,255);

}

}

return result;

}

/\* mark flags on each connected grapg by using 8-direction flood algorithm \*/

int mark(imt\_c\* data,imt\_c\* target,int row,int col,unsigned char flag)

{

/\* if the pixel is marked or black then jump over \*/

if(get\_pixel\_c(data,row,col) == 0 || get\_pixel\_c(target,row,col) != 0 || flag >= MAX\_CONNECTED\_GRAPH)

return 0;

/\* push the pixel into the stack \*/

push(row,col);

/\*while the stack is not empty, push its around white pixels into the stack and fill them with the same flag \*/

while(count > 0)

{

pop();

if(get\_pixel\_c(data,temp.row - 1,temp.col - 1) != 0 && get\_pixel\_c(target,temp.row - 1,temp.col - 1) == 0)

{

set\_pixel\_c(target,temp.row - 1,temp.col - 1,flag);

push(temp.row - 1,temp.col - 1);

stat[flag]++;

}

if(get\_pixel\_c(data,temp.row - 1,temp.col) != 0 && get\_pixel\_c(target,temp.row - 1,temp.col) == 0)

{

set\_pixel\_c(target,temp.row - 1,temp.col,flag);

push(temp.row - 1,temp.col);

stat[flag]++;

}

if(get\_pixel\_c(data,temp.row - 1,temp.col + 1) != 0 && get\_pixel\_c(target,temp.row - 1,temp.col - 1) == 0)

{

set\_pixel\_c(target,temp.row - 1,temp.col + 1,flag);

push(temp.row - 1,temp.col + 1);

stat[flag]++;

}

if(get\_pixel\_c(data,temp.row,temp.col - 1) != 0 && get\_pixel\_c(target,temp.row,temp.col - 1) == 0)

{

set\_pixel\_c(target,temp.row,temp.col - 1,flag);

push(temp.row,temp.col - 1);

stat[flag]++;

}

if(get\_pixel\_c(data,temp.row,temp.col + 1) != 0 && get\_pixel\_c(target,temp.row,temp.col + 1) == 0)

{

set\_pixel\_c(target,temp.row,temp.col + 1,flag);

push(temp.row,temp.col + 1);

stat[flag]++;

}

if(get\_pixel\_c(data,temp.row + 1,temp.col - 1) != 0 && get\_pixel\_c(target,temp.row + 1,temp.col - 1) == 0)

{

set\_pixel\_c(target,temp.row + 1,temp.col - 1,flag);

push(temp.row + 1,temp.col - 1);

stat[flag]++;

}

if(get\_pixel\_c(data,temp.row + 1,temp.col + 1) != 0 && get\_pixel\_c(target,temp.row + 1,temp.col + 1) == 0)

{

set\_pixel\_c(target,temp.row + 1,temp.col + 1,flag);

push(temp.row + 1,temp.col + 1);

stat[flag]++;

}

}

return 1;

}

/\* iteratively calls mark() to devide the image to be several connected graphs \*/

imt\_c\* imboundary(imt\_c\* m)

{

int i;

int j;

int ret;

int flag = 1;

imt\_c\* result;

result = create\_zero\_c(m->row,m->col);

for(i = 1;i <= m->row;i ++)

{

for(j = 1;j <= m->col;j ++)

{

ret = mark(m,result,i,j,flag);

if(ret)

flag++;

}

}

return result;

}

/\* delete the connected graphs if the area are below n \*/

void bwareaopen(imt\_c\* m,unsigned long n)

{

int i;

int j;

int flag;

imt\_c\* result;

for(i = 0;i < MAX\_CONNECTED\_GRAPH;i ++)

{

stat[i] = 0;

}

result = imboundary(m);

for(i = 1;i <= m->row;i ++)

for(j = 1;j <= m->col;j ++)

{

flag = get\_pixel\_c(result,i,j);

if(flag == 0)

{

continue;

}

else

{

if(stat[flag] < n)

{

set\_pixel\_c(m,i,j,0);

}

}

}

m\_free(result);

}

/\* focus on the hand area we want to analysis, see details in our report \*/

imt\_c\* focus(imt\_c\* m)

{

int i;

int j;

int\* left;

int\* right;

int\* band;

int rtn = 0;

imt\_c\* result;

left = (int\*)m\_malloc(m->row \* sizeof(int));

right = (int\*)m\_malloc(m->row \* sizeof(int));

band = (int\*)m\_malloc(m->row \* sizeof(int));

for(i = 0; i < m->row;i ++)

{

left[i] = 0;

right[i] = 0;

band[i] = 0;

}

for(i = 1;i <= m->row;i ++)

{

for(j = 1;j <= m->col;j ++)

{

if(get\_pixel\_c(m,i,j) == 255)

{

l0 = i;

break;

}

}

if(l0 != 0)

{

break;

}

}

for(i = 1;i <= m->row;i ++)

{

for(j = 1;j <= m->col;j ++)

{

if(get\_pixel\_c(m,i,j) == 255)

{

left[i - 1] = j;

break;

}

}

for(j = 0;j <= m->col - 1;j ++)

{

if(get\_pixel\_c(m,i,m->col - j) == 255)

{

right[i - 1] = m->col - j;

break;

}

}

band[i - 1] = right[i - 1] - left[i - 1];

}

for(i = 1;i <= m->col;i ++)

{

for(j = 1;j <= m->row;j ++)

{

if(get\_pixel\_c(m,j,i) == 255)

{

n0 = i;

break;

}

}

if(n0 != 0)

{

break;

}

}

for(i = 0;i <= m->col - 1;i ++)

{

for(j = 1;j < m->row;j ++)

{

if(get\_pixel\_c(m,j,m->col - i) == 255)

{

n1 = m->col - i;

break;

}

}

if(n1 != 0)

{

break;

}

}

for(i = 1;i <= m->row;i ++)

{

if(band[i - 1] > rtn)

{

rtn = band[i - 1];

h0 = left[i - 1];

h1 = right[i - 1];

l1 = i;

}

}

rtn = m->col;

for(i = l1;i <= m->row;i ++)

{

if(band[i - 1] > rtn)

{

rtn = band[i - 1];

l2 = i;

}

}

result = sub\_mt\_c(m,l0,l1,n0,n1);

return result;

}

/\* hand identifier, see details in our report \*/

int identifier(imt\_c\* m)

{

int i = 0;

int j = 0;

int k = 0;

int t = 0;

int top = 0;

int area = 0;

int flag = 0;

seg\_num = 0;

ratio = (double)m->row / (double)m->col;

t = m->row \* 0.65;

for(j = 1;j <= m->col;j ++)

{

for(i = 1;i <= m->row;i ++)

{

if(get\_pixel\_c(m,i,j) == 255)

{

top = m->row - i;

if(top > t)

{

for(k = 1; k <= m->row;k ++)

{

set\_pixel\_c(m,k,j,255);

}

}

else

{

for(k = i; k <= m->row;k ++)

{

set\_pixel\_c(m,k,j,0);

}

}

break;

}

}

}

for(i = 1;i <= m->row;i ++)

{

for(j = 1;j <= m->col;j ++)

{

if(get\_pixel\_c(m,i,j) == 255)

area ++;

}

}

for(i = 1;i <= m->col;i ++)

{

if(get\_pixel\_c(m,1,i) == 255)

{

if(flag == 0)

{

flag = 1;

seg\_num ++;

}

}

else

{

if(flag == 1)

flag = 0;

}

}

percent = (double)area / (double)(m->row \* m->col);

if(seg\_num == 0)

{

return 0;

}

else if(seg\_num > 2)

{

if(ratio > 0.45)

return 3;

else

return 1;

}

else if(seg\_num == 2)

{

if(ratio > 0.45)

return 2;

else

return 1;

}

else if(seg\_num == 1)

{

if(ratio < 0.5)

{

return 1;

}

else if(ratio < 0.75)

{

if(percent > 0.60)

return 1;

else

return 3;

}

if(ratio > 1.2)

{

if(percent > 0.55)

return 3;

else

return 2;

}

if(ratio < 0.9)

{

return 3;

}

if(percent < 0.55)

return 2;

else

return 3;

}

return 0;

}

/\* print out parameters \*/

void characteristic()

{

printf("ratio: %f,filled: %f,segments: %d\n",ratio,percent,seg\_num);

}

void debug(imt\_c\* m)

{

int i;

int j;

for(i = 1;i < m->row;i ++)

{

for(j = 1;j < m->col;j ++)

{

printf("%d ",get\_pixel\_c(m,i,j));

}

printf("\n");

}

}

#endif

VII. REFERENCE/CREDITS

[1] R. Fisher, S. Perkins, A. Walker, and E. Wolfart. Median Filter. *HYPERMEDIA IMAGE PROCESSING REFERENCE.* [Online]. Available: <http://homepages.inf.ed.ac.uk/rbf/HIPR2/median.htm>

[2] Wikipedia contributors. (2013, December). YDbDr. *Wikipedia, The Free Encyclopedia.* [Online]. Available: <http://en.wikipedia.org/wiki/YDbDr>

[3] Wikipedia contributors. (2014, May). RGB color model. *Wikipedia, The Free Encyclopedia.* [Online].Available: <http://en.wikipedia.org/wiki/RGB_color_model> (Accessed: May 17, 2014).

[4] Stephen Ostermiller. Efficiently Implementing Dilate and Erode Image Functions. *Ostermiller.* [online]. Available: <http://ostermiller.org/dilate_and_erode.html>

[5] bwareaopen. *MathWorks.* [Online]. Available: <http://www.mathworks.com/help/images/ref/bwareaopen.html>

[6] *Preparation of Papers for IEEE Transactions and Journals*, December 2013.