Lab 4: Calculating Blob Statistics

TRC3500 Sensors and Artificial Perception

Lab members

Kun Zhang	22701478
Seow Sheng Lun	22724281
Jiapeng Zou	24531308

1. Introduction

This laboratory practice explores some of the fundamental concepts in computer vision. Images were captured by a USB camera and could be viewed on a PC with the aid of OpenCV. A piece of C program was developed to process the images to find its statistics such as centres, orientation of the long axis, etc. These features were displayed on the original images in real time. Additionally, spread and elongation were calculated as a means of obtaining Euclidian distance which was a rudimentary way to classify different object.

2. Equipment and Software

Equipment used in the lab prac:
1) a USB camera
2) a PC

Software used in the lab prac:
3) Microsoft Visual Studio 2010 Professional
4) OpenCV
5) The supplied skeleton program

3. Theoretical Background

The moments of area are a geometrical property of an area that indicates the distribution of its points. For a given area, the moments of area are defined by

$$m_{pq} = \sum_i \sum_j i^p j^q \tau_{ij}$$

Equation 1 Moments of Area

where p and q are the order of the moments, τ is the binary number of the pixel.

Knowing moments of area gives us the ability to obtain a number of statistic of the area (or image). The centre of area is found by

$$i_0 = \frac{m_{10}}{m_{00}} = \frac{1}{A} \sum_{i=0}^{n-1} \sum_{j=0}^{m-1} i \tau_{ij}$$

$$j_0 = \frac{m_{01}}{m_{00}} = \frac{1}{A} \sum_{i=0}^{n-1} \sum_{j=0}^{m-1} j \tau_{ij}$$

Equation 2 Centre of area

where A, the area of the black area, equals to m_{00}

The orientation of the long axis of a blob is given by

$$\theta_0 = \frac{1}{2} \tan^{-1} \left[\frac{2(m_{00}m_{11} - m_{10}m_{01})}{(m_{00}m_{20} - m_{10}{}^2) - (m_{00}m_{02} - m_{01}{}^2)} \right]$$

Equation 3 Orientation of the long axis

Spread an elongation are given by

$$spread = \frac{(a+c)}{m_{00}^2}$$

$$elongation = \frac{\sqrt{b^2 + (a-c)^2}}{m_{00}^2}$$

$$a = \sum_{i=0}^{n-1} \sum_{j=0}^{m-1} i^2 \tau_{ij} - A i_0{}^2$$

$$b = \sum_{i=0}^{n-1} \sum_{j=0}^{m-1} ij \tau_{ij} - A i_0 j_0$$

$$c = \sum_{i=0}^{n-1} \sum_{j=0}^{m-1} j^2 \tau_{ij} - A j_0{}^2$$

Equation 4 Spread and elongation

A simple way to classify objects is to use Euclidean distance between features. If we store the features (such as spread and elongation) in to a vector, then we are able to calculate its Euclidian distance by

$$\text{Feature Vector } n = (fa_n, fb_n, \ldots)$$

$$Euclidean\ Distance = \sqrt{(fa_1 - fa_2)^2 + (fb_1 - fb_2)^2 + \ldots}$$

Equation 5 Euclidian distance

If the distance between images is comparably small, we can conclude that they represent the same object; otherwise, they are assumed to be different.

4. Procedures

1) Modify the skeleton code provided so that the image capturing program can display images.
2) Find the threshold of the image by averaging the maximum and minimum pixel values.
3) Calculate moments of areas with which we can find the centres of the blobs and its axis of minimum moment of inertia.
4) Show the calculated results on the console window and display the centre and orientation by showing a pointing arrow superimposed on the original image.
5) Calculate spread and elongation. Then, find Euclidian distance to differentiate different objects.

5. Experimental procedure

Part One – Image Import, Picture Processing

In this part, we import the image data into the PC and change the image to grey.

```
// Initializes capturing video from camera
 capture = cvCaptureFromCAM( -1 );
 if( !capture ) {
   fprintf(stderr,"Could not initialize capturing...\n");
   return -1;
 }
```

We initialize the camera setting, which may give out the camera USB already be connect to the PC or not.

```
 cvNamedWindow( "Camera image", 1 );
```

We use this function to create a window to show the new image.

```
 // Print details of image
 cout<<"image width ="<<frame->width<<" height ="<<frame->height;
 cout<<" depth ="<<frame->depth<<" channels ="<<frame->nChannels<<endl;
```

For this part, we use those functions to find out the width and the height for the picture. The width and the height value will be used for changing every pixel to grey, and then calculate the corresponding the value m00, m01, m10, m11, m20, m02.

```
int gray = ( row[ x*3 ] + row[ x*3+1 ] + row[ x*3+2 ] ) / 3;
        row[ x*3     ] = gray;
        row[ x*3 + 1 ] = gray;
         row[ x*3 + 2 ] = gray;
```

The code, which shows above, will be used for changing the original colour to grey.

Part Two – Use keyboard to control the stop for the image renew

We use keyboard for input. When CPU find that the keyboard have “q” input, the picture will stop renew.

```
while ('q'!=cvWaitKey(10));
```

At this part, we use do loop. When the keyboard find there have a “q” as a input the loop will stop then jump out. Which mean that images renew will be stopped.

Part Three – Find the Centre for the Image

From this part, we use equation 2 to calculate the moments of area.

$$m_{pq} = \sum_i \sum_j i^p j^q \tau_{ij}$$

Equation 2 Moments of Area

The code for this calculation shows below.

```
for( int y = 0; y < frame->height; y++) {
      for( int x = 0; x < frame->width; x++) {
         double tau = 0;
         if(gray>thresh){
            tau = 0;
         }else{
            tau = 1;
         }
         m00 = m00+tau;
         m01 = m01+tau*x;
         m10 = m10+tau*y;
         m11 = m11+tau*x*y;
         m02 = m02+tau*x*x;
         m20 = m20+tau*y*y;
      }
    }
long i0 = (long)(m10/m00);
long j0 = (long)(m01/m00);
printf("centre of area, Y = %d, X = %d\n",i0,j0);
```

At the beginning, we compare the threshold with every each pixel. If the value for the grey is greater than the threshold, the programming will consider that the pixel is white then set the value to 0. If the value is lower than the threshold, the value will be set to 1. Every pixel will be used to calculate the m00, m10, m01, m11, m20 and m02, which will be used to calculate the orientation, spread and elongation as follow.

Part Four – Determine the Orientation for the Graph

By using the equation 3 shows below, we can find the orientation for the graph.

$$\theta_0 = \frac{1}{2}\tan^{-1}\left[\frac{2\left(m_{00}m_{11} - m_{10}m_{01}\right)}{\left(m_{00}m_{20} - m_{10}{}^{2}\right) - \left(m_{00}m_{02} - m_{01}{}^{2}\right)}\right]$$

Equation 3 Orientation of the long axis

The calculation code for the orientation shows below.

```
double X_direction = 2*(m00*m11-m10*m01);
double Y_direction = (m00*m20-m10*m10)-(m00*m02-m01*m01);
double theta = 0.5 * atan2(X_direction , Y_direction);
theta = theta*180/3.1415926575897;
int AREA = 0;
double theta_360 = 0;
if (X_direction > 0){
        if (Y_direction > 0){
                        AREA = 1;
                        theta_360 = theta;
            }else{
                        AREA = 4;
                        theta_360 =  180 + theta;}
}else{
            if (Y_direction > 0){
                        AREA = 2;
                        theta_360 = theta ;
            }else{
                        AREA = 2;
                        theta_360 = 180 + theta;}
}
printf("Orientation  = %f\n",theta_360);
printf("Old_angle  = %f\n",theta);
```

Function ‘atan2()’ will be used to calculate the angle for the orientation. With the additional if conditions, the orientation can be read from 0 to 360 degree instead of 0 to 180 degree. All the value for m00, m10, m01, m11, m20 and m02 already has been calculate from above. We used function ‘atan2()’ can also distinguish the different angle from same tangent value by distinguish the negative and positive value for the numerator and denominator.

Part Five – Calculate the Euclidian Distance

For this part, we use function 4 to calculate the value for spread and elongation. And read the value shows on the screen to calculate the Euclidian distance.

$$spread = \frac{(a+c)}{m_{00}^2} \qquad a = \sum_{i=0}^{n-1}\sum_{j=0}^{m-1} i^2\tau_{ij} - Ai_0^2$$

$$b = \sum_{i=0}^{n-1}\sum_{j=0}^{m-1} ij\tau_{ij} - Ai_0j_0$$

$$elongation = \frac{\sqrt{b^2+(a-c)^2}}{m_{00}^2} \qquad c = \sum_{i=0}^{n-1}\sum_{j=0}^{m-1} j^2\tau_{ij} - Aj_0^2$$

Equation 4 Spread and elongation

The corresponding code shows the calculation for spread and elongation.

```
double  a = m20 - m00*i0*i0;
double  b = m11 - m00*i0*j0;
double  c = m02 - m00*j0*j0;
double speed = (a+b)/(m00*m00);
double elo = sqrt(b*b-(a-c)*(a-c))/(m00*m00);
```

Part Six – Show the Result on the Picture

For this part, we show a dot at the centre of the image. And then show the line at the right orientation. And then print the picture and the orientation line on the picture.

```
double red_angle = (theta_360) / 180 * 3.1415926575897;
centre.x = j0;
centre.y = i0;
p1 = centre;
p2.x = centre.x + 200*cos(red_angle);
p2.y = centre.y - 200*sin(red_angle);
cvCircle(frame, centre, circleRadius, color, lineThickness, lineType, lineShift);
cvLine(frame, p1, p2, color, lineThickness, lineType, lineShift);
cvShowImage( "Camera image", frame );
```

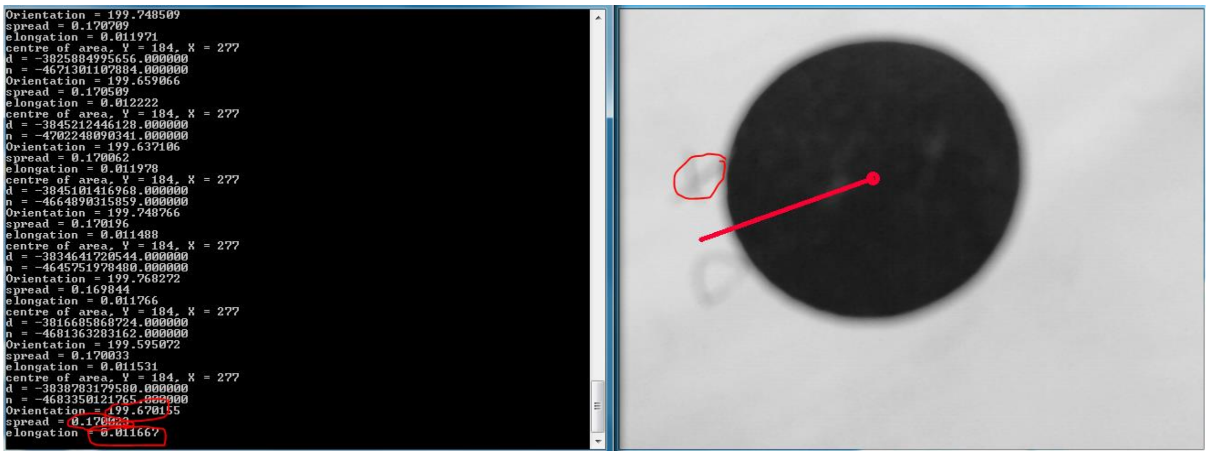
We use three functions in this part.
"cvCircle (img, center, radius, color, thickness, lineType, shift)" may show a circle at a center with different radius, color and line thickness at image.

"cvLine (img, pt1, pt2, color, thickness, lineType, shift)" may show a line from point1 to point2 with different color and thickness at image.

"cvShowImage("Camera image", frame)" shows frame at "Camera image" window.

Center point already been calculated before. In this part, just recall the previous value. The end for the line will be calculates by using 'sin' and 'cos' function.

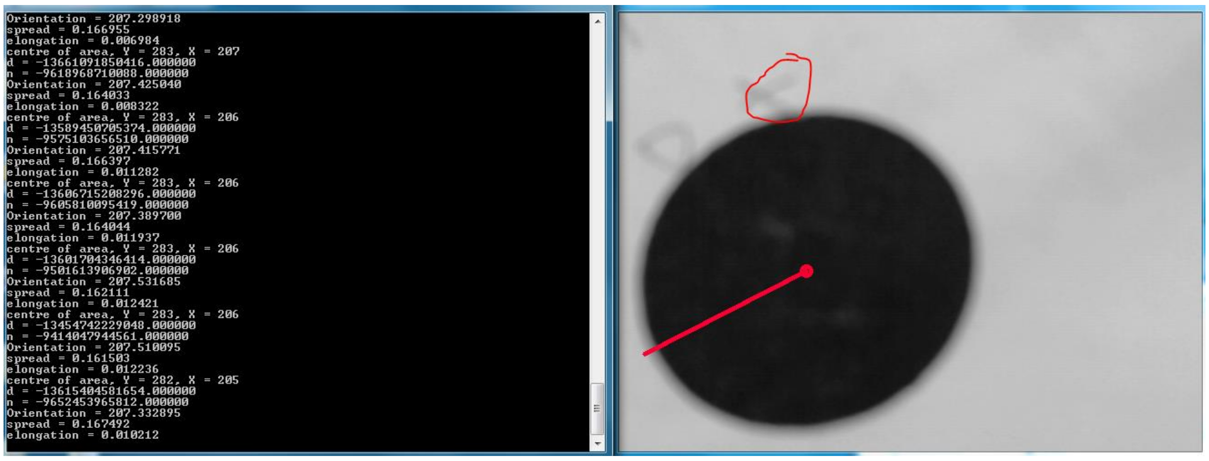
6. Result



```
Orientation = 199.748509
spread = 0.170709
elongation = 0.011971
centre of area, Y = 184, X = 277
d = -3825884995656.000000
n = -4671301107884.000000
Orientation = 199.659066
spread = 0.170509
elongation = 0.012222
centre of area, Y = 184, X = 277
d = -3845212446128.000000
n = -4702248090341.000000
Orientation = 199.637106
spread = 0.170062
elongation = 0.011978
centre of area, Y = 184, X = 277
d = -3845101416968.000000
n = -4664890315859.000000
Orientation = 199.748766
spread = 0.170196
elongation = 0.011488
centre of area, Y = 184, X = 277
d = -3834641720544.000000
n = -4645751978480.000000
Orientation = 199.768272
spread = 0.169844
elongation = 0.011766
centre of area, Y = 184, X = 277
d = -3816685868724.000000
n = -4681363283162.000000
Orientation = 199.595072
spread = 0.170033
elongation = 0.011531
centre of area, Y = 184, X = 277
d = -3838783179580.000000
n = -4683350121765.000000
Orientation = 199.670155
spread = 0.170023
elongation = 0.011667
```

Figure 1: Result obtained from circle shape

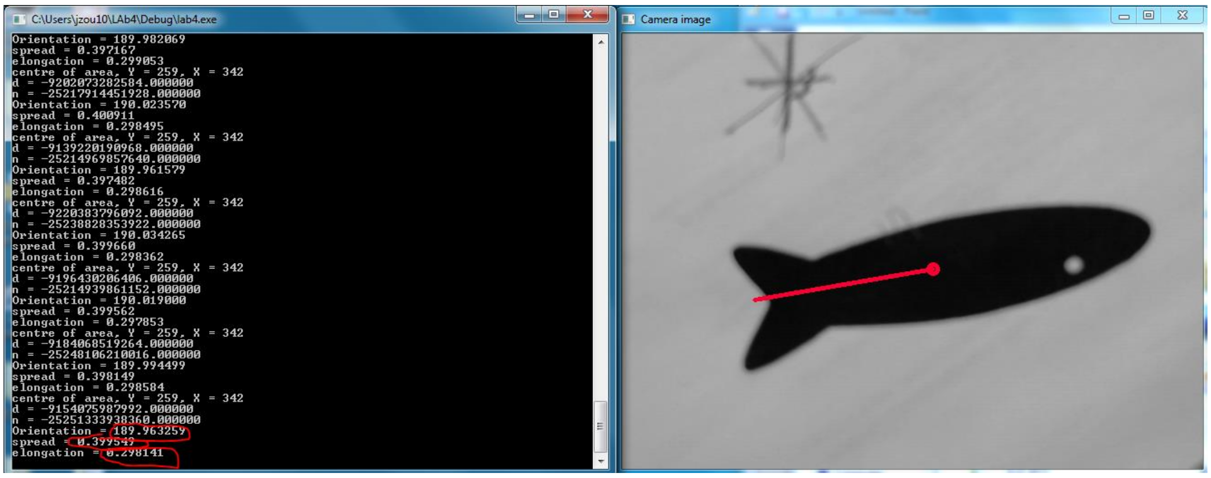
Orientation =199.67 degree, Spread = 0.17, Elongation = 0.012



```
Orientation = 207.298918
spread = 0.166955
elongation = 0.006984
centre of area, Y = 283, X = 207
d = -1366109185041 6.000000
n = -961896871008 8.000000
Orientation = 207.425040
spread = 0.164033
elongation = 0.008322
centre of area, Y = 283, X = 206
d = -1358945070537 4.000000
n = -957510365651 0.000000
Orientation = 207.415771
spread = 0.166397
elongation = 0.011282
centre of area, Y = 283, X = 206
d = -1360671520829 6.000000
n = -960581009541 9.000000
Orientation = 207.389700
spread = 0.164044
elongation = 0.011937
centre of area, Y = 283, X = 206
d = -1360170434641 4.000000
n = -950161390690 2.000000
Orientation = 207.531685
spread = 0.162111
elongation = 0.012421
centre of area, Y = 283, X = 206
d = -1345474222904 8.000000
n = -941404794456 1.000000
Orientation = 207.510095
spread = 0.161503
elongation = 0.012236
centre of area, Y = 282, X = 205
d = -1361540458165 4.000000
n = -965245396581 2.000000
Orientation = 207.332895
spread = 0.167492
elongation = 0.010212
```

Figure 2: Result obtained from circle shape

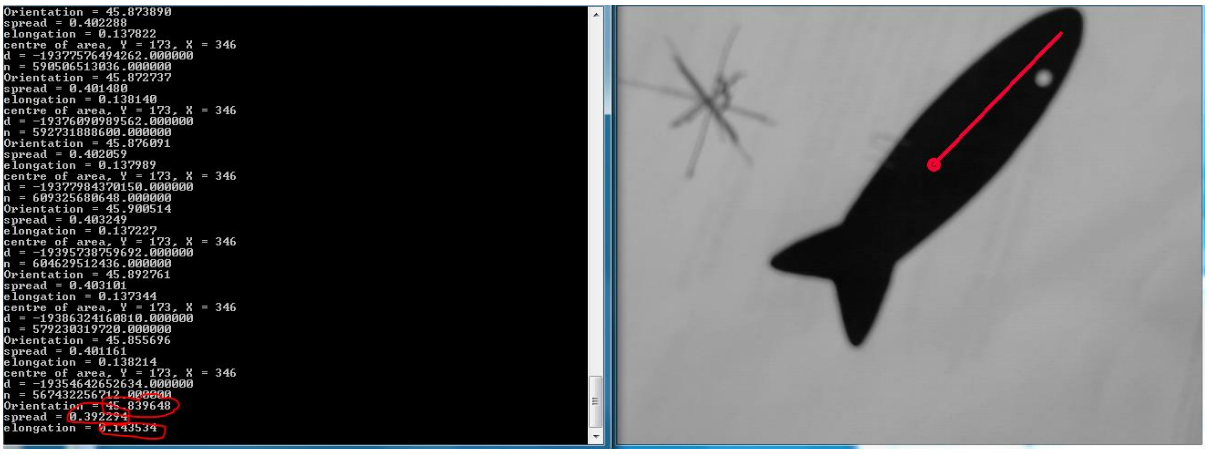
Orientation = 207.33 degree, Spread = 0.17, Elongation = 0.010



```
Orientation = 189.982069
spread = 0.397167
elongation = 0.299053
centre of area, Y = 259, X = 342
d = -9202073282584.000000
n = -25217914451928.000000
Orientation = 190.023570
spread = 0.400911
elongation = 0.298495
centre of area, Y = 259, X = 342
d = -9139220190968.000000
n = -25214969857640.000000
Orientation = 189.961579
spread = 0.397482
elongation = 0.298616
centre of area, Y = 259, X = 342
d = -9220383796092.000000
n = -25238828353922.000000
Orientation = 190.034265
spread = 0.399660
elongation = 0.298362
centre of area, Y = 259, X = 342
d = -9196430206406.000000
n = -25214939861152.000000
Orientation = 190.019000
spread = 0.399562
elongation = 0.297853
centre of area, Y = 259, X = 342
d = -9184068519264.000000
n = -25248106210016.000000
Orientation = 189.994499
spread = 0.398149
elongation = 0.298584
centre of area, Y = 259, X = 342
d = -9154075987992.000000
n = -25251333938360.000000
Orientation = 189.963259
spread = 0.399549
elongation = 0.298141
```

Figure 3: Result obtained from fish shape

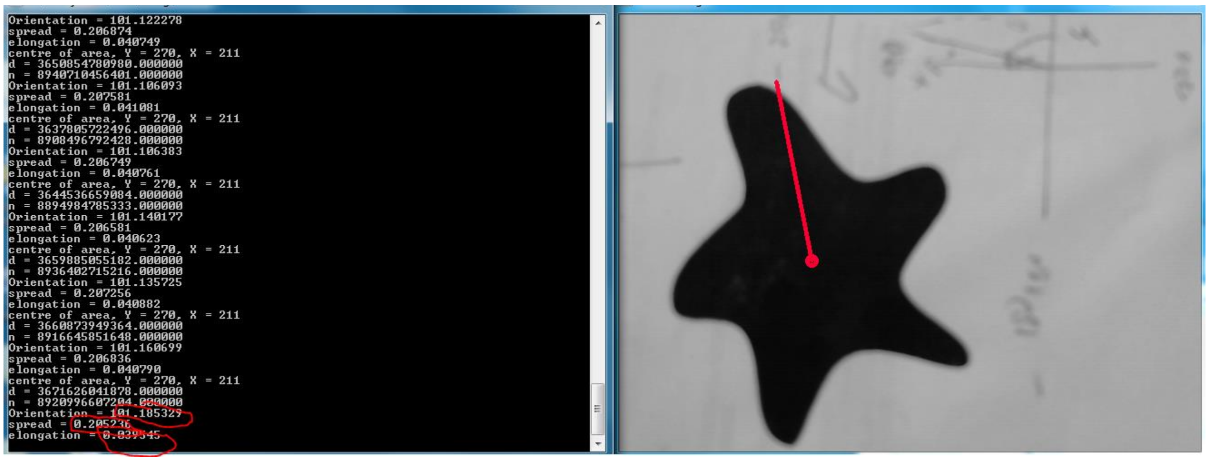
Orientation = 189.96 degree, Spread = 0.40, Elongation = 0.298



```
Orientation = 45.873890
spread = 0.402288
elongation = 0.137822
centre of area, Y = 173, X = 346
d = -19377576494262.000000
n = 590506513036.000000
Orientation = 45.872737
spread = 0.401480
elongation = 0.138140
centre of area, Y = 173, X = 346
d = -19376090989562.000000
n = 592731888600.000000
Orientation = 45.876091
spread = 0.402059
elongation = 0.137989
centre of area, Y = 173, X = 346
d = -19377984370150.000000
n = 609325680648.000000
Orientation = 45.900514
spread = 0.403249
elongation = 0.137227
centre of area, Y = 173, X = 346
d = -19395738759692.000000
n = 604629512436.000000
Orientation = 45.892761
spread = 0.403101
elongation = 0.137344
centre of area, Y = 173, X = 346
d = -19386324160810.000000
n = 579230319720.000000
Orientation = 45.855696
spread = 0.401161
elongation = 0.138214
centre of area, Y = 173, X = 346
d = -19354642652634.000000
n = 567432256712.000000
Orientation = 45.839648
spread = 0.392294
elongation = 0.143534
```

Figure 4: Result obtained from fish shape

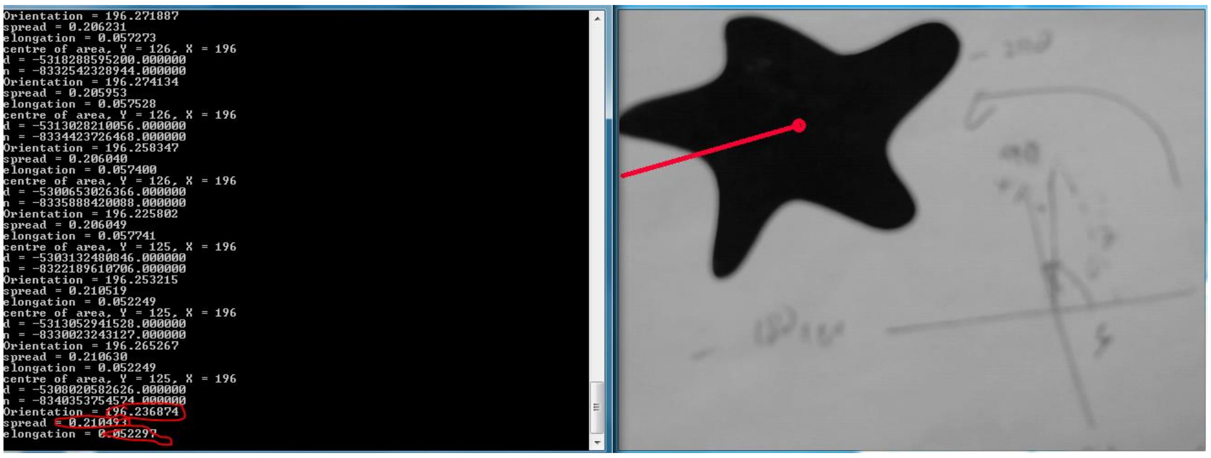
Orientation = 45.84 degree, Spread = 0.40, Elongation = 0.144



```
Orientation = 101.122278
spread = 0.206874
elongation = 0.040749
centre of area, Y = 270, X = 211
d = 3650854780980.000000
n = 8940710456401.000000
Orientation = 101.106093
spread = 0.207581
elongation = 0.041081
centre of area, Y = 270, X = 211
d = 3637805722496.000000
n = 8908496792428.000000
Orientation = 101.106383
spread = 0.206749
elongation = 0.040761
centre of area, Y = 270, X = 211
d = 3644536659084.000000
n = 8894984785333.000000
Orientation = 101.140177
spread = 0.206581
elongation = 0.040623
centre of area, Y = 270, X = 211
d = 3659885055182.000000
n = 8936402715216.000000
Orientation = 101.135725
spread = 0.207256
elongation = 0.040882
centre of area, Y = 270, X = 211
d = 3660873949364.000000
n = 8916645851648.000000
Orientation = 101.160699
spread = 0.206836
elongation = 0.040790
centre of area, Y = 270, X = 211
d = 3671626041878.000000
n = 8920996607204.000000
Orientation = 101.185329
spread = 0.205236
elongation = 0.039545
```

Figure 5: Result obtained from star shape

Orientation = 101.19 degree, Spread = 0.21, Elongation = 0.040



```
Orientation = 196.271887
spread = 0.206231
elongation = 0.057273
centre of area, Y = 126, X = 196
d = -5318288595200.000000
n = -8332542328944.000000
Orientation = 196.274134
spread = 0.205953
elongation = 0.057528
centre of area, Y = 126, X = 196
d = -5313028210056.000000
n = -8334423726468.000000
Orientation = 196.258347
spread = 0.206040
elongation = 0.057400
centre of area, Y = 126, X = 196
d = -5300653026366.000000
n = -8335888420088.000000
Orientation = 196.225802
spread = 0.206049
elongation = 0.057741
centre of area, Y = 125, X = 196
d = -5303132480846.000000
n = -8322189610706.000000
Orientation = 196.253215
spread = 0.210519
elongation = 0.052249
centre of area, Y = 125, X = 196
d = -5313052941528.000000
n = -8330023243127.000000
Orientation = 196.265267
spread = 0.210630
elongation = 0.052249
centre of area, Y = 125, X = 196
d = -5308020582626.000000
n = -8340353754574.000000
Orientation = 196.236874
spread = 0.210493
elongation = 0.052297
```

Figure 6: Result obtained from star shape

Orientation = 196.24 degree, Spread = 0.21, Elongation = 0.052

7. Discussion

In the result section, blobs provided for this lab were shown from figure 1 to 6 with each of their centre being marked with circle and their orientation of long axis being marked with a line.

For circle, the centre would be located at the origin of the circle as shown in figure 1 and 2. Circle doesn't really have a fixed orientation of long axis due to its nature of equal length from every angle. Hence explain why when the camera was rotated (notice the X beside the circle in figure 1 and 2), the orientation, spread and elongation still remain the same in both results for circle.

For fish shape, two different orientation of long axis were obtained as shown in figure 3 and 4. This is due to fish shape which has two legit orientations of long axis where one located at the head and the other located at the tail. This also depends on the how we position the camera. Therefore, the obtained results are acceptable. To further prove both the results were from the same fish shape, calculated spread from both results was exactly the same. Thus, the results were obtained from the same fish shape.

As for the star shape, again two different orientation of long axis were obtained. This can be seen from figure 5 and 6. From the first result, the orientation of long axis pointed to the top of the star and to the bottom of the star from the second result. Both the results have the same spread value which determines that both were the same star shape.

To investigate further, we can compare the result obtained from figure 1, 3 and 6. These three results obtained the same orientation of long axis but all the results were capturing different shape of blob. Therefore, in order to differentiate further, spread and elongation values were computed. With these values, we can uniquely determine that which value belong to which shape.

Improvement

There is vagueness in inverse tangent calculation. This is due to the sign ambiguity that "atan" function in c++ cannot determine in which quadrant the angle falls with only its tangent value. Therefore, to overcome that issue, "atan2" function was used in the code where the function takes into account the sign of both arguments in order to determine the right quadrant.

Besides, the resolution of the image captured from the USB camera was set to 640x480 which did not use the full capability of the USB camera provided that could perform 1280x720. With higher resolution, more data can be obtained from the image and the calculation for moments of area, centre of area, orientation or etc. can be more precise and accurate.

Other than that, the threshold of captured image was fixed to 115 in the code because the back lightning provided to the blob was always around 115. We could have computed the threshold for every image that are capturing and use the computed threshold value to obtain a more accurate calculation.

Finally, a narrower field of view camera would be good so that we could really focus on the object that we wanted to find out the centre of its area and its orientation.

Application

Binary image processing has been widely used in industry for part identification, location and quality control. For example in quality control, end product should give a constant centre of area, spread and elongation values. With these data, computer can determine whether the end product meet the requirement set by the manufacturer. The procedures are about the same for part identification application.

8. Conclusion

Following the procedure explained in the lecture note, a program was successfully written to determine the centre of captured object/blob marked with a circle and followed by the orientation of the long axis of the object/blob marked with a line using the library provided from OpenCV. Moreover, spread and elongation values were computed to further determine the features derived from the images.

9. Appendix

```
/////////////////////////////////////////////////////////////////
// Skeleton program for TRC3500
// Grabs images from a USB camera using OpenCV
// Written by Andy Russell 09th February 2006
// Modified by Michael Curtis 2011-2012 - updated for newer OpenCV
/////////////////////////////////////////////////////////////////
#include "cv.h"
#include "highgui.h"
#include <iostream>
#include <stdio.h>
#include <math.h>

using namespace std;

/////////////////////////////////////////////////////////////////
// main - initialises OpenCV and captures an image and changes it
/////////////////////////////////////////////////////////////////
int main( )
{
  CvCapture* capture = 0;
  IplImage* processedImage = 0;

  int thresh = 115;
  CvPoint centre;
  CvPoint p1;
  CvPoint p2;
  CvScalar color = { 50, 0, 250 };

  double m00=0,m01=0,m10=0,m11=0,m20=0,m02=0;
  const int circleRadius = 5;
  const int lineColor = 50;
  const int lineThickness = 3;
  const int lineType = 8;
  const int lineShift = 0;

  cout<<"Andy's USB camera program"<<endl<<"Press 'q' to quit"<<endl;

  // Initializes capturing video from camera
  capture = cvCaptureFromCAM( -1 );
  if( !capture ) {
    fprintf(stderr,"Could not initialize capturing...\n");
    return -1;
  }
```

```
// Creates window
cvNamedWindow( "Camera image", 1 );

// Camera image
IplImage* frame = 0;

// Grabs and returns a frame from camera
frame = cvQueryFrame( capture );

// Print details of image
cout<<"image width ="<<frame->width<<" height ="<<frame->height;
cout<<" depth ="<<frame->depth<<" channels ="<<frame->nChannels<<endl;

do {

  // Grabs and returns a frame from camera
  frame = cvQueryFrame( capture );
  if( !frame ) {
    break;
  }
        m00=0;
        m01=0;
        m10=0;
        m11=0;
        m20=0;
        m02=0;
  // Convert half of the image to gray
  for( int y = 0; y < frame->height; y++) {
    for( int x = 0; x < frame->width; x++) {
        // This is a pointer to the start of the current row.
        //  Note: The image is stored as a 1-D array which is mapped back
        //  into 2-space by multiplying the widthStep (the image width rounded to
        //  a "nice" value, eg a multiple of 4 or 8 depending on the OS and CPU)
        //  by the row number.
                uchar *row = (uchar*)(frame->imageData + frame->widthStep * y );

        int gray = ( row[ x*3 ] + row[ x*3+1 ] + row[ x*3+2 ] ) / 3;

        row[ x*3     ] = gray;
        row[ x*3 + 1 ] = gray;
        row[ x*3 + 2 ] = gray;
        double tau = 0;
        if(gray>thresh){
                tau = 0;
        }else{
                tau = 1;
        }
        m00 = m00+tau;
        m01 = m01+tau*x;
        m10 = m10+tau*y;
        m11 = m11+tau*x*y;
        m02 = m02+tau*x*x;
        m20 = m20+tau*y*y;
  }
}

        // centre of area
        long i0 = (long)(m10/m00);
        long j0 = (long)(m01/m00);
        printf("centre of area, Y = %d, X = %d\n",i0,j0);

        double X_direction = 2*(m00*m11-m10*m01);
        double Y_direction = (m00*m20-m10*m10)-(m00*m02-m01*m01);

        double theta = 0.5 * atan2(X_direction , Y_direction) ;

        theta = theta*180/3.1415926575897 + 90;
        double A = m20 - m00*i0*i0;
        double B = m11 - m00*j0*i0;
        double C = m02 - m00*j0*j0;

        double spread = (A+C)/(m00*m00);
        double ele = sqrt(B*B+(A-C)*(A-C))/(m00*m00);


        int AREA = 0;
        double theta_360 = 0;
        if (X_direction > 0){
                if (Y_direction > 0){
                        AREA = 1;
                        theta_360 = theta;
                }else{
                        AREA = 4;
                        theta_360 =  180 + theta;
                }

        }else{
                if (Y_direction > 0){
                        AREA = 2;
                        theta_360 = theta ;
                }else{
                        AREA = 2;
                        theta_360 = 180 + theta;
                }
        }

        printf("d = %f\n",X_direction);
        printf("n = %f\n",Y_direction);
        printf("Orientation = %f\n",theta_360);
        //printf("Old_angle = %f\n",theta);
        printf("spread = %f\n",spread);
        printf("elongation = %f\n",ele);

        //Calculate moments

        double red_angle = (theta_360) / 180 * 3.1415926575897;

        centre.x = j0;
        centre.y = i0;
        p1 = centre;

        p2.x = centre.x + 200*cos(red_angle);
        p2.y = centre.y - 200*sin(red_angle);

        cvCircle(frame, centre, circleRadius, color, lineThickness, lineType, lineShift);
        cvLine(frame, p1, p2, color, lineThickness, lineType, lineShift);

  // Shows the resulting image in the window
  cvShowImage( "Camera image", frame );

} while ('q'!=cvWaitKey(10));
//tidy up
// Releases the CvCapture structure
cvReleaseCapture( &capture );
// Destroys all the HighGUI windows
cvDestroyAllWindows( );

return 0;
```