Aerorobot/Robot
Vision-based Navigation

Yong Jia

Luleå University of Technology
Master Thesis, Continuation Courses
Space Science and Technology
Department of Space Science, Kiruna
YONG JIA

AEROBOT/ROBOT
VISION-BASED NAVIGATION

SCHOOL OF ENGINEERING

MSc in ASTRONAUTICS AND SPACE ENGINEERING

MSc THESIS
SCHOOL OF ENGINEERING

MSc in Astronautics and Space Engineering
MSc Thesis

Academic Year
2007/08

YONG JIA

AEROBOT/ROBOT
VISION-BASED NAVIGATION

Supervised by: Dr. Stephen Hobbs

June 2008

This thesis is submitted in partial fulfilment (75%) of the requirements for the Degree of MSc in Astronautics and Space Engineering

© Cranfield University
Abstract

Vision-base technique is an effective approach for aerobot/robot navigation in the close region. The aim of this thesis is to identify the position of a series of reference target autonomously from the image for close-range navigation. This project reviews some current work about videogrammetry in Cranfield University, develops an image processing algorithm to calculate the position of each label in the image. The main feature of this algorithm is to merge the fragments into clusters according their relevant property, and calculate the position of each unique label for robot/aerobot navigation.

The experiments are performed on a single SONY digital camera and typical PC, to evaluate the feasibility and reliability of this method. By comparing the images in three conditions (ideal image draw by Windows Paint, real image in experiment and image with pepper noise), the results illustrate the performance and limitation of this algorithm. Finally, it also provides some ideas to improve this navigation technique.
I would like to express my gratitude to my supervisor, Dr. Stephen Hobbs, for his kind help and suggestions on this thesis.

I also want to thank the supports of SpaceMaster Programme that give me the opportunity to study in Cranfield University.

Additionally, I would like to give my thanks to my parents and friends whose encouragements enabled me to complete this thesis.
# Contents

ABSTRACT I

ACKNOWLEDGEMENTS II

CONTENTS III

LIST OF FIGURES IV

LIST OF TABLES V

NOTATION VI

1. INTRODUCTION 1

2. LITERATURE 3
   2.1. Image acquisition 3
   2.2. Coordinate system 4
   2.3. Image plane 6
   2.4. Image calibration 7
   2.5. Pattern recognition and cluster analysis 8

3. METHOD 10
   3.1. Introduction of BMP image file 12
      3.1.1. File header 12
      3.1.2. Information header 12
      3.1.3. Colour palette 14
      3.1.4. Image data 14
   3.2. Colour threshold 14
   3.3. Pattern recognition 16
List of Figures

Figure 2.1: Imaging system with CCD sensor 3  
Figure 2.2: Model of vision system 4  
Figure 2.3: Video axes and angles in imaging system 5  
Figure 2.4: The image plane in video system 6  
Figure 2.5: The lens distortion: Barrel Shape (left) and Pincushion Shape (right) 7  
Figure 3.1: The flowchart of image data processing 10  
Figure 3.2: The calibration of camera for navigation 11  
Figure 3.3: BMP data structure 12  
Figure 3.4(a): The original image (b): The image after red filter 15  
(c): The image after green filter (d): The image after blue filter  
Figure 3.5: The original image for pattern recognition (size: 512x384) 16  
Figure 3.6(a): The image zoomed in size of 32x24 16  
(b): The image after colour threshold for pattern recognition (size: 32x24)  
Figure 3.7: The 0/1 matrix after filter for calculation (‘1’ is in the region and ‘0’ is not) 18  
Figure 3.8: The image for merging fragments 19  
Figure 3.9: The coordinates and size of all fragments after pattern recognition. 20  
Figure 3.10: The distance defined in merging fragments 21  
Figure 3.11: The number of pair within the different clusters 21  
Figure 3.12: The results of fragments list in each cluster 22  
Figure 3.13: The results of fragments in each cluster 23  
Figure 3.14: The image for cluster identification 24  
Figure 3.15: The list of each cluster with coordinate (mean column and row) and size (the number of pixels) 24  
Figure 3.16: The results after cluster identification in each label 25  
Figure 3.17: The list of labels and the cluster list in each label 25  
Figure 4.1: The ‘ideal’ image for the contrast with experiment (Image A) 26  
Figure 4.2: The fragment list (Image A) 26  
Figure 4.3: The cluster list (Image A) 27
Figure 4.4: The label list and the clusters in each label (Image A) 27
Figure 4.5: The label list with unique cluster number 1~9 (Image A) 28
Figure 4.6: The real image in the experiment (Image B) 28
Figure 4.7: The image after colour threshold (Image B) 29
Figure 4.8: The fragment list (Image B) 29
Figure 4.9: The fragment list where ‘spurious fragments’ occur (Image B) 30
Figure 4.10: The process of merging fragments and the list of clusters (Image B) 30
Figure 4.11: The label list and the clusters in each label (Image B) 31
Figure 4.12: The label list with unique cluster number 1~9 (Image B) 32
Figure 4.13: The real image with pepper noise (Image C) 32
Figure 4.14: The image after colour threshold (Image C) 33
Figure 4.15: The fragment list (Image C) 33
Figure 4.16: The fragment list where ‘spurious fragments’ occur (Image C) 34
Figure 4.17: The process of merging fragments and the list of clusters (Image C) 34
Figure 4.18: The label list and the clusters in each label (Image C) 35
Figure 4.19: The label list with unique cluster number 1~9 (Image C) 36
Figure 5.1: The Image B (left) in real experiment and Image C (right) with pepper noise 37
Figure 5.2: The Image B (left) and Image C (right) after colour threshold 38
Figure A.1: The specification of Sony DCR-TR7000E 44
Figure A.2: Main window of DVgate Still 45
Figure A.3: Still window of DVgate Still 45
Figure B.1: The image to illustrate the BMP file format 46
Figure B.2: The file header and information header in image 46
Figure B.3: The ASCII format (HEX) of image 47
Figure C.1: Flowchart of image processing for cluster identification 48
List of Tables

Table 5.1: The number of fragments, clusters and labels in Image A, B and C, respectively 38
Table 5.2: The list of labels with unique cluster number 39
Notation

**p**  position of target in *World Reference System*

**a**  position of single camera in *World Reference System*

**O**  the origin of *World Reference System*

**x,y,z**  coordinates of position in *World Reference System*

**O_c**  the origin of *Camera Reference System*

**e_1,e_2,e_3**  the unit axes in *Camera Reference System*, defined as view axis, inclination and azimuth respectively

**θ,φ,ψ**  the pose angles of camera in *World Reference System*, defined as inclination, azimuth and view axis respectively

**α,β**  inclination and azimuth of the target measured by camera

**x_{est},y_{est}**  the corrected coordinates of a pixel on image plane

**x_i,y_i**  the coordinates of a pixel on image plane

**a_i,b_j**  the coefficients of polynomial

**D**  the distance between the camera and grid
1. Introduction

Navigation is an important task for mobile robots. It covers a large range of theories and technologies. In terms of scales of application, typically the navigation comprises of two fields: global navigation and local navigation (Batalin, 2004).

For global navigation, the GPS (Global Positioning System) and INS (Inertial Navigation Systems) are extensively used in various areas.

For local navigation, vision-base techniques are an effective way to deal with the close-range navigation problem. There are many applications in space such as autonomous vehicles and planetary rovers. Vision-based navigation contains a variety of tasks that include pattern recognition, target identification, etc. In practical applications, the optical sensor and image processing algorithms are also critical factors to affect the accuracy of navigation.

The current project refers to some previous work about videogrammetry in Cranfield University. The main feature of this system is that using low cost commercial electronics devices (e.g. digital video and PC's) obtains good performance on measurements. In Hobbs (2003), the report documents the algorithms, data processing software and applications for videogrammetry system. In this report, several methods and algorithms are provided for camera position and pose calibration.

In order to determine the camera position and pose, imaging a set of reference points with known positions in user defined coordinate system are necessary. So how to measure the position of targets in the image and the accuracy of measurements are essential for calibration. One approach is to extract data from image by human’s eyes; another is to calculate the positions by computer.

This project reviews some work in Cranfield’s videogrammetry system and develops some image processing algorithms, which are used to identify the labels autonomously from a series of reference targets.

To test and evaluate the feasibility of the algorithms, some simple experiments are performed. The experiments are based on commercial off-the-shelf equipment, a single
SONY digital camera and typical PC. In this project, some main factors that affect the accuracy of calculation are discussed. The results of this project are expected to be applied to more space applications such as rendezvous, docking and planetary rover navigation.

**This thesis is composed of five chapters:**

The chapter *literature* reviews the background and current work on this project. Some techniques and methods are discussed to develop efficient vision-based applications for robot/aerobot navigation.

The chapter *method* investigates the methodology for autonomous calibration to calculate the pose and position of camera. The main feature of the method is to merge the fragments into clusters according their relevant property, and identify the unique label as reference targets for calibration of camera.

The chapter *results*: experiments are performed to assess the algorithms and code. Comparing the images in three conditions (ideal image draw by Windows Paint, real image in experiment and image with pepper noise), the results are provided to illustrate the performance of the method and algorithm.

The chapter *discussion* analyses the results from experiments and discusses the limitation of method mentioned in this thesis.

The chapter *conclusion* summarises the achievements from experiments and results. It also suggests some future work to improve this navigation technique.
2. Literature

2.1. Image Acquisition

In digital video cameras, a CCD sensor is used for image acquisition. The principal of CCD sensor is shown in Figure 2.1. The energy from an illumination source is reflected from a scene element. The imaging system collects the incoming energy and focuses it on to an image plane. The digital and analogue circuits sweep the sensor array on the image plane and convert the image into a digital form (Gonzalez, 2002).

Figure 2.1: Imaging system with CCD sensor (Gonzalez, 2002)
2.2. Coordinate System

The model of a video system for a single camera is shown in Figure 2.2. The coordinate system $(O,x,y,z)$ is chosen as the World Reference System. $P$ is the target in the World Reference System.

The coordinate system $(O, e_1, e_2, e_3)$ is defined as the Camera Reference System which comprises three orthogonal unit vectors: the viewing axis ($e_1$), the inclination direction ($e_2$) and the azimuth direction ($e_3$) (Endemano, 2000).

![Figure 2.2: Model of vision system (Endemano, 2000)](image)
The unit axes \((e_1, e_2, e_3)\) of the Camera Reference System can be expressed in terms of three pose angles \((\theta, \phi, \psi)\) relative to the World Reference System. In Figure 2.3, these pose angles are defined as:

The inclination \((\theta)\) and azimuth \((\phi)\) of the viewing axis \((e_1)\), and the angle \((\psi)\) about the viewing axis of the camera’s azimuth direction from the plane containing the view axis and the vertical. The rotation order is (1) rotate by \(\phi\) about \(Oz\), (2) rotate by \(\theta\) about \(Oy'\), the (3) rotate by \(\psi\) about \(Ox'\). (Seynat, 2000)

![Figure 2.3: Video axes and angles in imaging system (Seynat, 2000)](image)

In terms of three pose angles \((\theta, \phi, \psi)\), the unit vectors \((e_1, e_2, e_3)\) can be written as follows:

\[
e_1 = (\cos \theta \cos \phi, \cos \theta \sin \phi, \sin \theta) \tag{2.1}
\]

\[
e_2 = (-\sin \theta \cos \phi \cos \psi + \sin \phi \sin \psi, \\
-\sin \theta \sin \phi \cos \psi - \cos \phi \sin \psi, \cos \theta \cos \psi) \tag{2.2}
\]

\[
e_3 = (\sin \theta \cos \phi \sin \psi + \sin \phi \cos \psi, \\
-\sin \theta \sin \phi \sin \psi - \cos \phi \cos \psi, \cos \theta \sin \psi) \tag{2.3}
\]
2.3. Image plane

The projection of the target on the image plane is shown in Figure 2.4. The target is at a vector position \( p \), and the camera is at \( a \) in the *World Reference System*. By the projection of target (\( P' \)) within the image plane, the camera can measure the angles of inclination (\( \alpha \)) and azimuth (\( \beta \)) in the *Camera Reference System*. In this way, the inclination (\( \alpha \)) and azimuth (\( \beta \)) of the target can be determined in terms of camera unit vectors (\( e_1, e_2, e_3 \)), camera position (\( a \)) and target position (\( p \)) as follows (Braunwart, 2001):

\[
\tan \alpha = \frac{(p-a) \cdot e_2}{(p-a) \cdot e_1}
\]  

(2.4)
\[ \tan \beta = \frac{(p-a) \cdot e_j}{(p-a) \cdot e_i} \] (2.5)

2.4. Image Calibration

The lens distortion is an important effect in the image. Usually there are two types of distortion: ‘Barrel Shape’ and ‘Pincushion Shape’ shown in Figure 2.5 (Endemano, 2000). In this case, the video lens distorts the image in a characteristic ‘Barrel’ distortion (Seynat, 2000).

In order to calculate the distortion, a 3rd order polynomial is selected for best solution (Seynat, 2000).

For the horizontal direction (\( a_i \) is coefficient of polynomial):

\[
x_{\text{est}}(x_i, y_i) = a_0 + a_1 x_i + a_2 y_i + a_3 x_i^2 + a_4 x_i y_i + a_5 y_i^2
+ a_6 x_i^3 + a_7 x_i^2 y_i + a_8 x_i y_i^2 + a_9 y_i^3
\] (2.6)

For the vertical direction (\( b_i \) is coefficient of polynomial):

\[
y_{\text{est}}(x_i, y_i) = b_0 + b_1 x_i + b_2 y_i + b_3 x_i^2 + b_4 x_i y_i + b_5 y_i^2
+ b_6 x_i^3 + b_7 x_i^2 y_i + b_8 x_i y_i^2 + b_9 y_i^3
\] (2.7)
Hence, the angles of inclination ($\alpha$) and azimuth ($\beta$) are expressed in terms of corrected coordinates $x_{est}$ and $y_{est}$ as follows:

$$\tan \alpha = \frac{y_{est}(x, y)}{D}$$  \hspace{1cm} (2.8)

$$\tan \beta = \frac{x_{est}(x, y)}{D}$$  \hspace{1cm} (2.9)

Where $D$ is the distance between the camera and grid (Seyna, 2000)

If a minimum of 3 reference points are known in the image plane and their coordinates in World Reference System, the position $(x, y, z)$ and pose $(\theta, \phi, \psi)$ of the camera can be calculated for calibration. In practice, it is better to choose more reference points. In Hobbs (2003), this calibration method is discussed in detail.

### 2.5. Pattern recognition and cluster analysis

**Pattern recognition**

The book of Gonzalez (2002) introduces some approaches to pattern recognition. The approaches are divided mainly into two categories: decision theoretic and structural. The first area uses quantitative classifier such as length, area and texture. The second area is described by qualitative classifier such as a relational classifier.

In practice, when the distribution of each class about its mean is in the spherical form and largely separated from other classes, the minimum distance classifier yields good results. This method can also be used in cluster analysis in GIS (Geographic Information System).

Matching by correlation is another basic concept that is applied in pattern recognition. It needs to calculate the correlation coefficient between the image and a mask, then find the best match. However, if the image or mask is distorted and rotated, it is more difficult to look for the best match.
Optimum statistical classifier is a probabilistic approach to pattern recognition. Because the randomness generated by pattern classes, probability consideration is more realistic in measuring physical events.

Neural network is a structural model and is the basis of powerful algorithms in pattern recognition currently. It is inspired by biological neural networks and changes its own structure according to external and internal information. Neutral networks can be used in pattern recognition such as radar systems, face identification, etc. (Gonzalez, 2002)

**Cluster analysis**

Cluster analysis is an important problem in many applications such as pattern recognition and GIS. Bezdek (2005) and Nabney (2002) discussed some methods and algorithms for pattern recognition and imaging processing. They presented a large number of examples to illustrate the methods and previous works about this problem.

In GIS, point pattern analysis is a useful tool to research the distribution of target. Woodall (2004) discusses applications of point pattern analysis in forest and ecology surveys.

Fung (2001) describes many basic algorithms in cluster analysis. They are categorised in two main classes: *parametric clustering* and *non-parametric clustering*. In this project, the non-parametric clustering algorithms are focused on due to the requirements of image processing. A representative algorithm in non-parametric clustering is hierarchical clustering. This method will be discussed further in cluster identification in the next chapter.
3. Methods

The flowchart in Figure 3.1 shows the image data processing developed for this vision-based system:

![Flowchart of image data processing](image)

**Term Notes:**

Fragment: A fragment is the minimal set of pixels which is composed of contiguous pixels with the same property in pattern recognition.

Cluster: A cluster is a set of fragments which is the basic element to identify each unique target.
Label: A label is a unique target which includes a set of clusters in order to calculate the positions of targets in the image.

The main procedures are given as follows:

1. Record the reference targets and download the frame in PC. Use DVgate Still to capture the image in Bitmap format which contains the reference targets.

2. Read the Bitmap file and extract the colour data. Use colour threshold methods to filter the image. After filtering, write the image in Bitmap format, and also write the data into memory.

3. Recognise pattern of each fragment, and calculate the centroid (column, row) and size of each fragment.

4. Merge the fragments with nearby fragments which result from noise or pixel deficiency in CCD array. Combine them into a cluster that will be used next.

5. Identify the clusters by distance matrix; compare the distance of each two clusters to find the cluster in the same label. Print all clusters in one label and calculate the centroid (column, row) and size of the label.

6. Assume the label as reference target to calibrate the pose and position of camera.

The main steps in Figure 3.2 to calibrate the camera for navigation are shown below

Extract reference targets in the image (at least three targets in each image)   Read out the image coordinates of reference targets

Record the true position of reference targets in the defined coordinated system   Solve the camera position and pose in the defined coordinated system

Figure 3.2: The calibration of camera for navigation
3.1. Introduction of BMP Image File

In this case, the 24 bit true colour no compression BMP file format (.bmp) are used to store bitmap digital images on Microsoft Windows operation systems (Microsoft, 2005). A typical BMP file contains 3 or 4 blocks of data as shown on Figure 3.3. The detail of BMP file in ASCII format is presented in Appendix A.

3.1.1. File Header

The first section is BMP file header. We assume short of 2 bytes, int of 4 bytes and long of 8 bytes. The length of header is 14 bytes. The structure of file header as followed in C programme.

```c
typedef struct               /* BMP file header structure ****/
{
    unsigned short bfType;   /* Magic number for file */
    unsigned int   bfSize;    /* Size of file */
    unsigned short bfReserved1; /* Reserved */
    unsigned short bfReserved2; /* ... */
    unsigned int   bfOffBits; /* Offset to bitmap data */
} BITMAPFILEHEADER;
```

Figure 3.3
BMP data structure

3.1.2. Information Header

This block is used to tell the detailed information in the image. The info header is 40 bytes in length and the structure in C code is described below.
typedef struct                         /**** BMP file info structure ****/
{
    unsigned int    biSize;  /* Size of info header */
    int            biWidth; /* Width of image */
    int            biHeight; /* Height of image */
    unsigned short biPlanes; /* Number of color planes */
    unsigned short biBitCount; /* Number of bits per pixel */
    unsigned int   biCompression; /* Type of compression to use */
    unsigned int   biSizeImage; /* Size of image data */
    int            biXPelsPerMeter; /* X pixels per meter */
    int            biYPelsPerMeter; /* Y pixels per meter */
    unsigned int   biClrUsed; /* Number of colors used */
    unsigned int   biClrImportant; /* Number of important colors */
} BITMAPINFOHEADER;

The most useful fields are image width and height, the number of bits per pixel (typical values are 1, 4, 8, 16, 24 and 32) and the compression type. Here the number of colour plane is set to 1.

The 24 bit true colour image is simple to read. There is no colour palette such that the image data follows after info header immediately. The colour data consist of 3 bytes values per pixel in B (blue), G (green), and R (red) order. Each byte gives the colour intensity from 0 to 255. So here only 24 bit true colour image will be discussed.

The constants for compression type are listed below:

0    /* No compression - straight BGR data */
1    /* 8-bit run-length compression */
Here the compression type is assumed to be only type 0 (no compression).

### 3.1.3. Colour Palette (optional)

The colour palette is not used when the bitmap is 16-bit or higher, so there is no colour palette block in this case due to 24-bit image.

### 3.1.4. Image Data

Here 24-bit colour image data are stored in this block, pixel by pixel. Pixels are scanned starting in the lower left corner, going from left to right, and then row by row from the bottom to the top of the image. The number of bytes matching a row should be a multiple of 4 bytes. If in each row the number of bytes is not divisible by 4, the line is padded with additional bytes of unspecified value so that the next row will start on a multiple of 4 bytes location in the file.

### 3.2. Colour Threshold

The aim of colour threshold method is to find whether each pixel’s colour matches a target colour. For instance, in Figure 3.4 the three different colour thresholds are used to filter the original image.

The image of Figure 3.4 is 256 colours, and the size is 512x384.

Colour band: The threshold of red filter \((b<=20 \&\& g<=20 \&\& r>=240)\)

- green filter \((r<=20 \&\& b<=20 \&\& g>=240)\)
- blue filter \((r<=20 \&\& g<=20 \&\& b>=240)\)
Read the original BMP image and write the BMP image after filtering. For red, green and blue filter they are shown below.

Figure 3.4(a): The original image
Figure 3.4(b): The image after red filter
Figure 3.4(c): The image after green filter
Figure 3.4(d): The image after blue filter
3.3. Pattern Recognition

To simplify the processing of image and show the pattern recognition clearly, software ACDSee 7.0 is used to make the size of original image smaller. The size of new image is $1/16$ of original one as shown in the Figure 3.6(a).

Figure 3.5: The original image for pattern recognition (size: 512x384)

Figure 3.6(a): The image zoomed in size of 32x24

Figure 3.6(b): The image after colour threshold for pattern recognition (size: 32x24)
The Figure 3.6(b) is the image after filter for pattern calculation. The colour threshold is chosen as \( b \leq 40 \) \&\& \( g \leq 40 \) \&\& \( r \geq 230 \).

The Figure 3.7 shows the 0/1 matrix and results of pattern recognition for each fragment. The 0/1 matrix is correspond to the Figure 3.6(b) after colour filter. Here ‘1’ represent this pixel is in the recognized fragment after colour filter, ‘0’ is not in fragment.

This search algorithm, as used in AVI1.exe (Hobbs, 2003), detects a list of set pixels contiguous starting in the lower left corner, going from left to right, and then row by row from the bottom to the top of the image. When the first pixel matches the target colour threshold is found and equal to ‘1’, the coordinates of this pixel are stored and this pixel is defined as initial pixel.

After that, the search works around the edges enclosing the initial pixel, checks the edge coordinates to make sure they are still inside the image. If the edges are still within the image then the pixel in these edges is set to ‘1’; if the edges have had to be adjusted back to the image limits then the pixel in these edges is set to ‘0’.

If the fragment is found, the next step is to calculate and store the centroid coordinates of each fragment. Finally, all the points just found from the image are blanked such that the search for other clusters can resume. In Figure 3.7, ‘npixel’ is the size of each fragment. The ‘cmean’ and ‘rmean’ are the centroid coordinates in column and row respectively.

This algorithm does not strictly check for contiguous pixels but instead finds a box of clear pixels enclosing the original pixel. This should work fine if the blocks of contiguous pixels are well-separated.
Figure 3.7: The 0/1 matrix after filter for calculation (‘1’ is in the region and ‘0’ is not)
3.4. Merge Fragments

The process to merge fragments can be seen as form of low-pass filter on the image after colour threshold to associate outlier pixels with nearby clusters (Hobbs, 2003). In practice, it is useful to adjust the results caused by noise, vibration, pixel deficiency and other factors.

The image in Figure 3.8 is used to illustrate the algorithm to merge fragments.

![Figure 3.8: The image for merging fragments](image)
The Figure 3.9 shows the coordinates and size of all fragments after the step of pattern recognition.

```
<table>
<thead>
<tr>
<th>nfrag</th>
<th>list</th>
<th>frag_c</th>
<th>frag_w</th>
<th>frag_h</th>
<th>frag_size</th>
</tr>
</thead>
<tbody>
<tr>
<td>32</td>
<td>0</td>
<td>273.000000</td>
<td>112.500000</td>
<td>598</td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>1</td>
<td>344.000000</td>
<td>110.000000</td>
<td>437</td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>2</td>
<td>153.000000</td>
<td>134.000000</td>
<td>475</td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>3</td>
<td>183.500000</td>
<td>139.500000</td>
<td>468</td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>4</td>
<td>380.500000</td>
<td>141.000000</td>
<td>136</td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>5</td>
<td>362.000000</td>
<td>148.000000</td>
<td>405</td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>6</td>
<td>171.000000</td>
<td>165.500000</td>
<td>306</td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>7</td>
<td>183.500000</td>
<td>169.000000</td>
<td>44</td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>8</td>
<td>158.000000</td>
<td>172.000000</td>
<td>33</td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>9</td>
<td>171.500000</td>
<td>179.000000</td>
<td>70</td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>10</td>
<td>73.000000</td>
<td>194.000000</td>
<td>405</td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>11</td>
<td>73.000000</td>
<td>212.000000</td>
<td>435</td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>12</td>
<td>72.000000</td>
<td>223.000000</td>
<td>105</td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>13</td>
<td>197.000000</td>
<td>244.500000</td>
<td>570</td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>14</td>
<td>227.000000</td>
<td>245.000000</td>
<td>651</td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>15</td>
<td>212.000000</td>
<td>247.000000</td>
<td>63</td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>16</td>
<td>347.000000</td>
<td>270.500000</td>
<td>304</td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>17</td>
<td>104.500000</td>
<td>282.500000</td>
<td>140</td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>18</td>
<td>85.500000</td>
<td>284.500000</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>19</td>
<td>346.500000</td>
<td>283.500000</td>
<td>72</td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>20</td>
<td>371.000000</td>
<td>303.500000</td>
<td>596</td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>21</td>
<td>311.500000</td>
<td>279.500000</td>
<td>40</td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>22</td>
<td>321.000000</td>
<td>299.500000</td>
<td>156</td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>23</td>
<td>84.500000</td>
<td>298.500000</td>
<td>96</td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>24</td>
<td>100.500000</td>
<td>298.000000</td>
<td>84</td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>25</td>
<td>388.500000</td>
<td>300.500000</td>
<td>48</td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>26</td>
<td>374.000000</td>
<td>319.500000</td>
<td>102</td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>27</td>
<td>138.500000</td>
<td>328.500000</td>
<td>128</td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>28</td>
<td>76.000000</td>
<td>333.000000</td>
<td>231</td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>29</td>
<td>130.000000</td>
<td>343.500000</td>
<td>132</td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>30</td>
<td>146.000000</td>
<td>344.000000</td>
<td>143</td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>31</td>
<td>69.500000</td>
<td>346.500000</td>
<td>80</td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>32</td>
<td>85.500000</td>
<td>346.500000</td>
<td>160</td>
<td></td>
</tr>
</tbody>
</table>
```

Figure 3.9: the coordinates and size of all fragments after pattern recognition.
In Figure 3.10, the parameters in this algorithm are defined as:

$d$ is distance between core fragment and peripheral fragments,

$d_{\text{cluster}}$ is defined as the control area of each fragment which is equal to $\sqrt{N}$ ($N$ is size of each fragment).

If $d \leq d_{\text{cluster}}$, it indicates the peripheral fragment and the core fragment belong to the same cluster.

![Figure 3.10: The distance defined in merging fragments](image)

![Figure 3.11: The number of pair within the different clusters](image)
In this way to merge fragments, 14 pairs are found in this image as shows in Figure 3.11. In some pairs there are several same fragments, so the program is developed to remove the repeated fragments in each cluster. After data processing, the list and number of fragments in each cluster are given as followed in Figure 3.12. The fragment without merging any other fragments will be seen as a cluster.

This algorithm has various criteria to combine the spurious pixels with real target, while each criterion will lead to different results. A well designed experiment the criteria can be chosen straightforwardly to give unambiguous results. This method to identify the list of fragments in a cluster can be used in cluster identification as well.

The Figure 3.13 shows the results of fragments in each cluster (in green circle):
3.5. Cluster Identification

After the step of merging fragments, the clusters are listed in the sequences with coordinates (mean value of column and mean value of row) and size (the number of pixels). The process of cluster identification can be seen as a hierarchical clustering that combines the clusters into a label according to specific criterion. It is a global search algorithm which needs to compute the distance matrix among each cluster.

In this thesis, distance matrix is defined as a symmetrical two dimensional NxN array (N is number of clusters). Each element represents the distance between each two adjacent clusters. In this algorithm, the distance threshold (D) is a critical parameter that should be initialised before the running of program. Comparing the elements in distance matrix with the initialised threshold allows all clusters belonging to the same label to be found.
Figure 3.15 presents the list of clusters in the image Figure 3.14 as follows

**Figure 3.14: The image for cluster identification**

**Figure 3.15: The list of each cluster with coordinate (mean column and row) and size (the number of pixels)**
In this case, the distance threshold is $D^2 < 6000$. The Figure 3.16 and Figure 3.17 show the results of labels (in green circle) after cluster identification.

![Figure 3.16: The results after cluster identification in each label](image)

![Figure 3.17: The list of labels and the cluster list in each label](image)
4. Results

To test and evaluate the performance of the algorithm mentioned above, Image A (see in Figure 4.1) is drawn in Windows Paint and it includes 9 unique labels which correspond to their number of clusters. It is assumed that there is no noise or interference in the image, so the number of fragments (see in Figure 4.2) is equal to clusters number (see in Figure 4.3). The size of Image A is 768x576 which is the same as the Image B and Image C in real experiment. The colour threshold: $b \leq 50 \&\& g \leq 50 \&\& r = 130$

Image A

![Image A](image.png)

Figure 4.1. The 'ideal' image for the contrast with experiment

![Fragment List](fragment_list.png)

Figure 4.2. The fragment list (Image A)
Figure 4.3: The cluster list (Image A)

Figure 4.4: The label list and the clusters in each label (Image A)
The distance threshold: $D^2 < 7000$

![Image A](image)

**Figure 4.5:** The label list with unique cluster number 1~9 (Image A)

**Image B**

Image B (Figure 4.6) is the real image that is extracted from the experiment. The background uses the white board to enhance the contrast of image and eliminate the noise and distortion in the image.

The colour threshold and distance threshold of Image B are the same as Image A.

![Image B](image)

**Figure 4.6:** The real image in the experiment (Image B)
Because of some internal and external noise in the imaging system, some spurious pixels occur in Figure 4.9. So the number of fragments in real image (see in Figure 4.8) is more than the ideal Image A. In practice, it is necessary to investigate an effective method to merge fragments which can be considered as a ‘filter’ to make the algorithm more robust and reliable.
In Figure 4.10, it shows the process of merging fragments and calculates the coordinate and size of new cluster for the cluster identification. As is seen that after merging fragments, the number of clusters in equal to the Image A.
In Figure 4.11, it lists each label and the clusters in it. The number of total cluster represents unique label. So from this results in Figure 4.12, the label is identified and the position of label in the image is obtained that can be used for navigation.
In this project, the colour threshold is the essential method to extract the target from the image. Therefore, some ‘pepper noise’ is added to evaluate the performance of algorithm and variation of target position in the image. ‘Pepper noise’ is a form of noise typically on the image which occurs as different colour pixels decreasing the quality of original image. In this experiment, the ‘pepper noise’ is added by software ACDSee 7.0. The accuracy is taken into account for navigation application.
From the results printed out in Figure 4.15 and Figure 4.16, the number of fragment, the centroid and size of each fragment are changed due to some noise added into the image.

In Figure 4.17, after merging fragment, the number of clusters in Image C is equal to Image B and Image C. It illustrates the function of merging fragments in the whole algorithm.
Figure 4.16: The fragment list where ‘spurious fragments’ occur (Image C)

Figure 4.17: The process of merging fragments and the list of clusters (Image C)
Figure 4.16, 4.17 and 4.18 show the same process that are used in Image A and Image C.

The label list after cluster identification is printed in Figure 4.19.
Figure 4.19: The label list with unique cluster number 1~9 (Image C)
5. Discussion

The aim of this project is to identify the position of a series of reference target from the image in some simple experiments. For close-range navigation, this technique is based on the vision system and image processing algorithm, so the accuracy and reliability of this method is fundamental for application.

In practice, there are many factors affect the accuracy of results. In this project, some main factors are discussed as follows:

**Vision System**

Figure 5.1: The Image B (left) in real experiment and Image C (right) with pepper noise

In this vision system, the CCD digital video is used as imaging sensor. So the resolution and sensitivity of CCD sensor is very important for image acquisition. The intensity of light determines the value of three principal colours, red(R), green (G) and blue (B). The random noise caused by electric circuit and environment also has an influence on the quality of image. Figure 5.1 shows the difference between real image in experiment and image with pepper noise.
Colour threshold

How to choose the colour threshold is critical for pattern recognition in this method. The colour threshold depends on the colours value in target region compared with surrounding pixels. The contrast between target and background is a difficult problem to concern. In order to enhance the accuracy of target position in the image, a high contrast with background is essential for experiment. Figure 5.2 show the images after colour threshold.

![Figure 5.2: The Image B (left) and Image C (right) after colour threshold](image)

Merging fragment

The function of merging fragment can think as a ‘filter’ which has various criterions corresponding to different context. In this experiment, the main task of merging fragment is to associate the ‘spurious’ small pixels within the real clusters. Table 5.1 lists the number of clusters before and after this step.

<table>
<thead>
<tr>
<th></th>
<th>nfrag</th>
<th>ncluster</th>
<th>nlabel</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>45</td>
<td>45</td>
<td>9</td>
</tr>
<tr>
<td>B</td>
<td>47</td>
<td>45</td>
<td>9</td>
</tr>
<tr>
<td>C</td>
<td>48</td>
<td>45</td>
<td>9</td>
</tr>
</tbody>
</table>

Table 5.1: The number of fragments, clusters and labels in Image A, B and C respectively
Cluster identification

This step is based on a global search algorithm and compared the element of distance matrix with threshold. In practice, if the distance between two labels is near to the distance from a cluster to another in one label, it is not reliable. In some cases, after colour threshold filtering, some spurious fragments occur which might lead to false label and cluster number.

<table>
<thead>
<tr>
<th>List</th>
<th>Label_cmean</th>
<th>Label_rmean</th>
<th>Label_size</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Image B</td>
<td>Image C</td>
<td>Image B</td>
</tr>
<tr>
<td>No.1</td>
<td>144.31</td>
<td>144.56</td>
<td>479.65</td>
</tr>
<tr>
<td>No.8</td>
<td>355.39</td>
<td>355.33</td>
<td>122.72</td>
</tr>
<tr>
<td>No.9</td>
<td>566.24</td>
<td>565.67</td>
<td>126.78</td>
</tr>
<tr>
<td>No.7</td>
<td>154.87</td>
<td>154.68</td>
<td>134.62</td>
</tr>
<tr>
<td>No.6</td>
<td>562.80</td>
<td>563.05</td>
<td>310.17</td>
</tr>
<tr>
<td>No.5</td>
<td>348.34</td>
<td>348.39</td>
<td>331.55</td>
</tr>
<tr>
<td>No.4</td>
<td>147.36</td>
<td>147.58</td>
<td>324.63</td>
</tr>
<tr>
<td>No.3</td>
<td>565.23</td>
<td>565.21</td>
<td>470.21</td>
</tr>
<tr>
<td>No.2</td>
<td>344.94</td>
<td>345.29</td>
<td>487.30</td>
</tr>
</tbody>
</table>

Table 5.2: The list of labels with unique cluster number
Table 5.2 presents the final results of the imaging process and lists the position of each label in column and row (origin at the low left corner). 'cmean' and 'rmean' represent the mean value of column and row in each label.

Compare the coordinates of each label in image B and image C, it indicates that the pepper noise does not significantly affect the accuracy of position. The pepper noise causes the quality of the image to blur, so the number of pixels in the target is decreased.
6. Conclusion

This project reviews some current work, develops and tests the technique which is applied in vision-based navigation. Some simple experiments are performed based on digital video camera. The results show the algorithm of imaging process step by step to evaluate the feasibility of this method.

The accuracy of this technique is mainly based on colour threshold filter and cluster analysis. So the contrast between targets and background, the distribution of clusters are two critical factors for final results. From this point, this algorithm is not robust to find and read labels in a more complicated image.

If the contrast between targets and background is strong, clusters distribute densely within each label as well as labels scattered separately, it is a simple and effective method to identify a label autonomously.

In this project, the method to read each label depends on counting the number of clusters. In some experiments, the position of each label in the image is accurate, but the number of clusters in each label is wrong due to some noise or colour interference. So it needs other methods to identify the labels, such as the geometry relation or possibility distribution, to make the results reliable.

In practice, there are many limitations in this technique which need to be improved by updated techniques. The future work includes:

1. Implication of optical imaging sensor. This is fundamental requirement in vision-based technique to improve the accuracy for navigation. In order to reduce the cost of equipment, the more elaborate experimental conditions need to be considered so as to enhance the quality of image.
2. In this project, only one algorithm is used for merging fragment. Some other algorithms need to be developed to deal with different tasks. Cluster analysis is a difficult problem in pattern recognition. In reality, the background of target is complicated. The more features of target are extracted by image processing; the results are more accurate and reliable.

3. In the experiments mentioned above, only the still image is extracted from video. For mobile target and vision system, the implication of motion needs to be considered. In real-time system, it also needs to be extended for further application.
7. References


Fung, G. A comprehensive overview of basic clustering algorithm. 2001


Hobbs, S.E., Target position and trajectory measurements by videogrammetry. College of Aeronautics report 0208, Cranfield University , November 2003.


Woodall, C.W, Graham, G.M. A technique for conducting point pattern analysis of cluster plot stem-maps. Forest ecology and management. 2004
Appendix A. Equipments in Experiment

A.1. Digital Video Camera

The digital video camera (Sony DCR-TR7000E) is used for measurement in the experiment. The image device is CCD sensor, and the interface is used with IEEE 1394 FileWire to connect the DV and PC. More technical specifications in Braunwart (2001) as follows:

---

**Specifications**

<table>
<thead>
<tr>
<th>Video camera recorder</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>System</strong></td>
</tr>
<tr>
<td>Video recording system</td>
</tr>
<tr>
<td>2rotary heads</td>
</tr>
<tr>
<td>Helical scanning system</td>
</tr>
<tr>
<td>Audio recording system</td>
</tr>
<tr>
<td>Rotary heads, PCM system</td>
</tr>
<tr>
<td>Quantization: 12 bits (Fs 32 kHz, stereo 1, stereo 2), 16 bits (Fs 48 kHz, stereo)</td>
</tr>
<tr>
<td>Video signal</td>
</tr>
<tr>
<td>PAL, colour, CCIR standards</td>
</tr>
<tr>
<td>Recommended cassette</td>
</tr>
<tr>
<td>118 video cassette</td>
</tr>
<tr>
<td>Recording/playback time (using 90 min. cassette)</td>
</tr>
<tr>
<td>1 hours</td>
</tr>
<tr>
<td>Fastforward/rewind time (using 90 min. cassette)</td>
</tr>
<tr>
<td>Approx. 8 min.</td>
</tr>
<tr>
<td>Image device</td>
</tr>
<tr>
<td>1/4 inch CCD (Charge Coupled Device)</td>
</tr>
<tr>
<td>Approx. 800,000 pixels</td>
</tr>
<tr>
<td>(Effective: Approx. 400,000 pixels)</td>
</tr>
<tr>
<td>Lens</td>
</tr>
<tr>
<td>Combined power zoom lens</td>
</tr>
<tr>
<td>Filter diameter 37 mm (1 1/16 in) (2x: Optical, 8x: Digital)</td>
</tr>
<tr>
<td>Focal length 36 - 72 mm (5/32 - 2 7/8 in)</td>
</tr>
<tr>
<td>When converted to a 25 mm still camera 46 - 92 mm (1 5/8 - 3 3/4 in)</td>
</tr>
<tr>
<td>Colour temperature</td>
</tr>
<tr>
<td>Auto</td>
</tr>
<tr>
<td>Minimum illumination</td>
</tr>
<tr>
<td>3 lux (F 1.4)</td>
</tr>
<tr>
<td>0 lux (in the NightShot mode)*</td>
</tr>
<tr>
<td>* Objects unable to be seen due to the dark can be shot with infrared lighting.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input and output connectors</th>
</tr>
</thead>
<tbody>
<tr>
<td>5 video output</td>
</tr>
<tr>
<td>4-pin mini DIN</td>
</tr>
<tr>
<td>Luminance signal: 1 Vpp, 75 ohms, unbalanced</td>
</tr>
<tr>
<td>Chrominance signal: 0.3 Vpp, 75 ohms, unbalanced</td>
</tr>
<tr>
<td>Video output</td>
</tr>
<tr>
<td>Phono jack, 1 Vpp, 75 ohms, unbalanced</td>
</tr>
<tr>
<td>Audio output</td>
</tr>
<tr>
<td>Phono jacks (2 stereo L and R)</td>
</tr>
<tr>
<td>327 mV, (at output impedance 47 kilohms) impedance less than 2.2 kilohms</td>
</tr>
<tr>
<td>RFU DC OUT</td>
</tr>
<tr>
<td>Special minilack, DC 5V</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>General</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Power requirements 7.2 V (battery pack)</td>
</tr>
<tr>
<td>8.4 V (AC power adaptor)</td>
</tr>
<tr>
<td>Average power consumption (when using the battery pack)</td>
</tr>
<tr>
<td>During camera recording 3.2 W</td>
</tr>
<tr>
<td>Operating temperature 0 °C to 40 °C (32 °F to 104 °F)</td>
</tr>
<tr>
<td>Storage temperature -20 °C to +60 °C (-4 °F to +140 °F)</td>
</tr>
<tr>
<td>Dimensions (Approx.) 101 x 106 x 195 mm (4 x 4 1/4 x 7 3/4 in.) (w/h/d)</td>
</tr>
</tbody>
</table>

**Mass (approx.)**

790 g (1 lb 11 oz) excluding the battery pack, lithium battery, cassette and shoulder strap DCR-TR7000E: 930 g (2 lb)
including the battery pack NF-F530, lithium battery CR2025, 98 mm. cassette shoulder strap DCR-TR7000E: 950 g (2 lb 1 oz) including the battery pack NF-F530, lithium battery CR2025, 90 mm cassette and shoulder strap

**Supplied accessories**

See page 5.

---

**AC power adaptor**

<table>
<thead>
<tr>
<th>Power requirements</th>
</tr>
</thead>
<tbody>
<tr>
<td>180 - 240 V AC, 50/60 Hz</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Power consumption</th>
</tr>
</thead>
<tbody>
<tr>
<td>23 W</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Output voltage</th>
</tr>
</thead>
<tbody>
<tr>
<td>DC OUT, 8.4 V, 1.5 A in operating mode</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Operating temperature</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 °C to 40 °C (32 °F to 104 °F)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Storage temperature</th>
</tr>
</thead>
<tbody>
<tr>
<td>-20 °C to +60 °C (-4 °F to +140 °F)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Dimensions (approx.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>125 x 30 x 62 mm (5 x 19/16 x 2 1/2 in.) (w/h/d)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Mass (approx.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>280 g (0.8 oz) excluding power cord</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Cord length (approx.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Main lead: 2 m (6.6 feet) Connecting cord: 1.5 m (5.2 feet)</td>
</tr>
</tbody>
</table>

Design and specifications are subject to change without notice.

---

Figure A.1: The specification of Sony DCR-TR7000E
A.2. DVgate Still

In the experiment, DVgate Still is used for capturing video images from DV devices as still images. The image is stored as Bitmap format (.bmp file). The process for capturing still image is shown below (Sony, 2002):

- Main Window

![Main Window](image)

Figure A.2: Main window of DVgate Still (Sony, 2002)

- Still Window

![Still Window](image)

Figure A.3: Still window of DVgate Still (Sony, 2002)
Appendix B. BMP image file format

To describe the format of BMP image file, the image is chosen for size of 5x5, 24bit true colour and no compression which is shown in Figure B.1.

![Figure B.1: The image to illustrate the BMP file format](image)

After code written in Appendix C is run and results are read out, it can be seen that the file header and information header of bitmap file in Figure B.2. The height and width of this image is 5 respectively, and the image is 24 bits true colour and no compression.

![Figure B.2: The file header and information header in image](image)
Figure B.3 shows the ASCII file in HEX which is opened by software UltraEdit. The first 2 bytes in BMP file is ‘BM’ (ASCII code 0x424d). The red block is file header and account for 14 bytes. The green block is information header which is account for 40 bytes. In this case, there is no colour palette, so the colour data is followed the information header.

The pixels are scanned starting in the lower left corner, going from left to right, and then row by row from the bottom to the top of the image. The number of bytes matching a row should be a multiple of 4 bytes. If the number of bytes in each row is not divisible by 4, the line is padded with additional bytes of unspecified value so that the next row will start on a multiple of 4 bytes location in the file.
Appendix C. Source Code

This program is written in Microsoft Visual C++ 6.0. Within the program, the aim of functions are to read and write image file (bitmap 24bit no compression format), pattern recognition, merge fragments and clusters identification. Chapter 3 (Methods) shows the data processing sequences and the results of each step.

Figure C.1: Flowchart of image processing for cluster identification
```c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

/*
Sample data structure of BMP file
*/

typedef struct                       /**** BMP file header structure ****/
{
    unsigned short bfType;               /* Magic number for file */
    unsigned int   bfSize;                  /* Size of file */
    unsigned short bfReserved1;      /* Reserved */
    unsigned short bfReserved2;      /* ... */
    unsigned int   bfOffBits;               /* Offset to bitmap data */
} BITMAPFILEHEADER;

typedef struct                       /**** BMP file info structure ****/
{
    unsigned int   biSize;                   /* Size of info header */
    int            biWidth;                       /* Width of image */
    int            biHeight;                      /* Height of image */
    unsigned short biPlanes;            /* Number of color planes */
    unsigned short biBitCount;         /* Number of bits per pixel */
    unsigned int   biCompression;    /* Type of compression to use */
    unsigned int   biSizeImage;        /* Size of image data */
    int            biXPelsPerMeter;       /* X pixels per meter */
    int            biYPelsPerMeter;       /* Y pixels per meter */
    unsigned int   biClrUsed;            /* Number of colors used */
    unsigned int   biClrImportant;      /* Number of important colors */
} BITMAPINFOHEADER;

/******/ Constants for the biCompression field...
*/

#define BI_RGB       0             /* No compression - straight BGR data */
#define BI_RLE8      1             /* 8-bit run-length compression */
#define BI_RLE4      2             /* 4-bit run-length compression */
#define BI_BITFIELDS 3            /* RGB bitmap with RGB masks */

typedef struct                       /**** Colormap entry structure ****/
{
    unsigned char  rgbBlue;          /* Blue value */
    unsigned char  rgbGreen;         /* Green value */
```
unsigned char rgbRed;        /* Red value */
unsigned char rgbReserved;   /* Reserved */
} RGBQUAD;

typedef struct                       /**** Bitmap information structure ****/
{  
BITMAPINFOHEADER bmiHeader; /* Image header */  
RGBQUAD bmiColors[256]; /* Image colormap */  
} BITMAPINFO;

typedef struct {
  unsigned char r,g,b;      
} COLOURINDEX;

/
* 'read_word()' - Read a 16-bit unsigned short.  
*/
static unsigned short read_word(FILE *fp) {
  unsigned char b0, b1; /* Bytes from file */
  b0 = getc(fp);
  b1 = getc(fp);
  return ((b1 << 8) | b0);
}

/*
* 'read_dword()' - Read a 32-bit unsigned int.  
*/
static unsigned int read_dword(FILE *fp) {
  unsigned char b0, b1, b2, b3; /* Bytes from file */
  b0 = getc(fp);
  b1 = getc(fp);
  b2 = getc(fp);
  b3 = getc(fp);
  return (b0 << 24) | (b1 << 16) | (b2 << 8) | b3;
}
return ((((((b3 << 8) | b2) << 8) | b1) << 8) | b0);
}

/*
 * 'read_long()' - Read a 32-bit signed integer.
 */
static int                        /* O - 32-bit int */
read_long(FILE *fp)               /* I - File to read from */
{
    unsigned char b0, b1, b2, b3; /* Bytes from file */
    b0 = getc(fp);
    b1 = getc(fp);
    b2 = getc(fp);
    b3 = getc(fp);
    return ((int)(((((b3 << 8) | b2) << 8) | b1) << 8) | b0);
}

/*
 * 'write_word()' - Write a 16-bit unsigned short integer.
 */
static int                     /* O - 0 on success, -1 on error */
write_word(FILE *fp, /* I - File to write to */
           unsigned short w)   /* I - Integer to write */
{
    putc(w, fp);
    return (putc(w >> 8, fp));
}

/*
 * 'write_dword()' - Write a 32-bit unsigned integer.
 */
static int                    /* O - 0 on success, -1 on error */
write_dword(FILE *fp, /* I - File to write to */
            unsigned int dw)  /* I - Integer to write */
{
    putc(dw, fp);
    putc(dw >> 8, fp);
}
putc(dw >> 16, fp);
return (putc(dw >> 24, fp));
}

/*
* 'write_long()' - Write a 32-bit signed integer.
*/

static int           /* O - 0 on success, -1 on error */
write_long(FILE *fp, /* I - File to write to */
int  l)   /* I - Integer to write */
{
putc(l, fp);
putc(l >> 8, fp);
putc(l >> 16, fp);
return (putc(l >> 24, fp));
}

int main(int argc,char **argv)
{
    FILE *fp,*fp1;       //input and output image file
    int i,j;
    unsigned char  r,g,b;
    unsigned char  junk;
    BITMAPFILEHEADER bmfHeader;   // bitmap file header
    BITMAPINFOHEADER bmiHeader;   //bitmap info header
    int Width, Height;
    int Width_4;
    unsigned char *Image,*Image1;  //pointer to colour values
    char filter;
    int *array;
    int *matrix;
    int *pixel_col, *pixel_row;
    int max_pixel;
    int n_set, n_inc, n_inc0;
    int edge1, edge2, edge3, edge4;  // test whether or not in the edge
int col0, row0, ncol,nrow, col, row;
int col_up, col_dn, row_up, row_dn;
int del_col_dn, del_col_up, del_row dn, del_row up;

float *frag_c, *frag_r;       // data structure of fragments
int *frag_n, frag_inc, nfrag;
int *frag, frag_long;

int k, n, npoint;
float cmean, rmean;

float d2,d2_lable;            // distance threshold
int *pair_1, *pair_2, *pair;  // data structure of pair
int pair_inc, npair;
int *pair_list;
int pair_long;

int *section_1, *section_2, *section; // data structure of pairs with common element
int sect_inc, nsection;
int *sect_list;
int sect_long;

int p,q,s,l;
int pair_count, sect_count;

int *single;  //data structure of single
int single_inc, nsingle;

int z,t,w;

float cluster_c, cluster_r;  // data structure of cluster
int cluster_n;

float *clust_c, *clust_r;
int *clust_n, clust_inc, ncluster;
int *clust, clust_long;

float label_cmean, label_rmean;  // data structure of labels
int label_size;

float *label_c, *label_r;
int *label_n, label_inc, nlabel;
int *label_list, list_inc;
/* load bitmap file (".bmp");*/

    if ((fp=fopen("real.bmp","rb"))==NULL)
    {
        printf("can not open source file\n");
        exit(-1);
    }

    fp1=fopen("out.bmp","wb");   // write the bitmap

    if ((fp1=fopen("out.bmp","wb"))==NULL)
    {
        printf("can not print out file\n");
        exit(-1);
    }

    /***************************************************************************/
    bmfHeader.bfType      = read_word(fp);           // read bitmap file header
    bmfHeader.bfSize      = read_dword(fp);
    bmfHeader.bfReserved1 = read_word(fp);
    bmfHeader.bfReserved2 = read_word(fp);
    bmfHeader.bfOffBits   = read_dword(fp);

    bmiHeader.biSize         = read_dword(fp);       //read bitmap info header
    bmiHeader.biWidth       = read_long(fp);
    bmiHeader.biHeight      = read_long(fp);
    bmiHeader.biPlanes      = read_word(fp);
    bmiHeader.biBitCount    = read_word(fp);
    bmiHeader.biCompression = read_dword(fp);
    bmiHeader.biSizeImage   = read_dword(fp);
    bmiHeader.biXPelsPerMeter = read_long(fp);
    bmiHeader.biYPelsPerMeter = read_long(fp);
    bmiHeader.biClrUsed     = read_dword(fp);
    bmiHeader.biClrImportant = read_dword(fp);
    /***************************************************************************/

    // check bitmap file if bfType is "BM"
    if (bmfHeader.bfType != 0x4D42)
    {
        printf("This is not Bitmap file !\n");
        exit(0);
    }

}
// print file header
printf("Print file header:\n");
printf("bfType=%x\n",bmfHeader.bfType);
printf("bfSize=%d\n",bmfHeader.bfSize);
printf("bfReserved1=%d\n",bmfHeader.bfReserved1);
printf("bfReserved2=%d\n",bmfHeader.bfReserved2);
printf("bfOffBits=%d\n",bmfHeader.bfOffBits);

printf("\n");

// print information header
printf("Print information header:\n");
printf("biSize=%d\n",bmiHeader.biSize);
printf("biWidth=%d\n",bmiHeader.biWidth);
printf("biHeight=%d\n",bmiHeader.biHeight);
printf("biPlanes=%d\n",bmiHeader.biPlanes);
printf("biBitCount=%d\n",bmiHeader.biBitCount);
printf("biCompression=%d\n",bmiHeader.biCompression);
printf("biSizeImage=%d\n",bmiHeader.biSizeImage);
printf("biXPelsPerMeter=%d\n",bmiHeader.biXPelsPerMeter);
printf("biYPelsPerMeter=%d\n",bmiHeader.biYPelsPerMeter);
printf("biClrUsed=%d\n",bmiHeader.biClrUsed);
printf("biClrImportant=%d\n",bmiHeader.biClrImportant);

/**************************************************************************/
write_word( fp1, bmfHeader.bfType);          /* bfType */        //write bitmap file header
write_dword(fp1, bmfHeader.bfSize);          /* bfSize */
write_word( fp1, bmfHeader.bfReserved1);     /* bfReserved1 */
write_word( fp1, bmfHeader.bfReserved2);     /* bfReserved2 */
write_dword(fp1, bmiHeader.biSize);             //write bitmap info header
write_long( fp1, bmiHeader.biWidth);
write_long( fp1, bmiHeader.biHeight);
write_word( fp1, bmiHeader.biPlanes);
write_word( fp1, bmiHeader.biBitCount);
write_dword(fp1, bmiHeader.biCompression);
write_dword(fp1, bmiHeader.biSizeImage);
write_long( fp1, bmiHeader.biXPelsPerMeter);
write_long( fp1, bmiHeader.biYPelsPerMeter);
write_dword(fp1, bmiHeader.biClrUsed);
write_dword(fp1, bmiHeader.biClrImportant);

/**************************************************************************/
printf("\n");
printf("Print the color data: \n");

switch (bmiHeader.biBitCount) {               //judge the bitmap file is 24bit or not
    case 1:
        break;
    case 4:
        break;
    case 8:
        break;
    case 24:
        Width  = bmiHeader.biWidth;      // read image width
        Height = bmiHeader.biHeight;     // read image height
        Width_4 = 4*((3*Width-1)/4+1);   //choose a number is 4 times of image width
        junk = Width_4 - 3 * Width;

        // allocate the memory for image data
        Image = (unsigned char *)malloc(Width_4*Height*sizeof(unsigned char));
        Image1 = (unsigned char *)malloc(Width_4*Height*sizeof(unsigned char));

        // image 0/1 matrix
        array  = (int *)malloc(Width*Height*sizeof(int));
        matrix = (int *)malloc(Width*Height*sizeof(int));

        max_pixel = Width * Height;
        pixel_col = (int *)malloc(max_pixel*sizeof(int));
        pixel_row = (int *)malloc(max_pixel*sizeof(int));

        // allocate the memory for fragments
        frag_c = (float *)malloc(max_pixel*sizeof(float));
        frag_r = (float *)malloc(max_pixel*sizeof(float));
        frag_n = (int *)malloc(max_pixel*sizeof(int));
        frag   = (int *)malloc(max_pixel*sizeof(int));

        // allocate the memory for pair list
        pair_1 = (int *)malloc(max_pixel*sizeof(int));
        pair_2 = (int *)malloc(max_pixel*sizeof(int));
        pair = (int *)malloc(max_pixel*sizeof(int));
        pair_list = (int *)malloc(max_pixel*sizeof(int));

        // allocate the memory for fragment pair with common elements
section_1 = (int *)malloc(max_pixel*sizeof(int));
section_2 = (int *)malloc(max_pixel*sizeof(int));
section = (int *)malloc(max_pixel*sizeof(int));
sect_list = (int *)malloc(max_pixel*sizeof(int));
single = (int *)malloc(max_pixel*sizeof(int));

//allocate the memory for clusters
clust_c = (float *)malloc(max_pixel*sizeof(float));
clust_r = (float *)malloc(max_pixel*sizeof(float));
clust_n = (int *)malloc(max_pixel*sizeof(int));
clust = (int *)malloc(max_pixel*sizeof(int));

//allocate the memory for labels
label_c = (float *)malloc(max_pixel*sizeof(float));
label_r = (float *)malloc(max_pixel*sizeof(float));
label_n = (int *)malloc(max_pixel*sizeof(int));
label_list = (int *)malloc(max_pixel*sizeof(int));

fseek(fp,bmfHeader.bfOffBits,0); // seek the head of colour date
fseek(fp1,bmfHeader.bfOffBits,0);

printf(" Input filter type: ");
scanf("%c",&filter);
printf("\n");
switch (filter) { // choose filter
	
}/**************************************************************************/
Print the raw bitmap image
**************************************************************************/
case 'a': // print the pixel values of whole image
	printf("Print all image pixels:\n");
for(i=0; i<Height; i++)
{
    fread(Image, sizeof(unsigned char), Width_4, fp);
    for(j=0;j<Height;j++)
    {
        b = *(Image+ 3*j);
        g = *(Image+ 3*j +1);
        r = *(Image+ 3*j +2);
    }
printf("(%d ",i);
printf("%d ",j);
printf("%d ",r);
printf("%d ",g);
printf("%d ",b);

*(Image1+ 3*j) = b; //write the colour value into memory
*(Image1+ 3*j +1) = g ;
*(Image1+ 3*j +2) = r;

// write the pixel value into bitmap file
fwrite(Image1+ 3*j, sizeof(unsigned char),1,fp1);
fwrite(Image1+ 3*j +1,sizeof(unsigned char),1,fp1);
fwrite(Image1+ 3*j +2,sizeof(unsigned char),1,fp1);

}
printf("\n");

// write the junk in each row
fwrite(Image1+ 3*Width, sizeof(unsigned char), junk, fp1);

Image=Image+Width_4;
Image1 = Image1 + Width_4;
}

printf("\n");
break;

**************************************************************************
Using the colour threshold to filter the bitmap image
**************************************************************************
case 'r':                       // colour threshold filter in red colour
    printf("Print the red pixels:\n");

for(i=0; i<Height; i++)
{
    fread(Image, sizeof(unsigned char), Width_4, fp);

    for(j=0;j<Width;j++)
    {
        b = *(Image+ 3*j);

        *(Image1+ 3*j) = b; //write the colour value into memory
        *(Image1+ 3*j +1) = g ;
        *(Image1+ 3*j +2) = r;

        // write the pixel value into bitmap file
        fwrite(Image1+ 3*j, sizeof(unsigned char),1,fp1);
fwrite(Image1+ 3*j +1,sizeof(unsigned char),1,fp1);
fwrite(Image1+ 3*j +2,sizeof(unsigned char),1,fp1);
    }

    printf("\n");
    break;

/****************************************************************************
Using the colour threshold to filter the bitmap image
****************************************************************************/

g = *(Image+ 3*j +1);
r = *(Image+ 3*j +2);

/***************************************************************************/
//colour threshold
if((b<=140&& b>=70) && (g<=70 && g>=20) && (r>=50 && r<=120))
{
    printf("(%d ",i);
    printf("%d ",j);
    printf("%d ",r);
    printf("%d ",g);
    printf("%d ",b);

    *(Image1+ 3*j)    = b;
    *(Image1+ 3*j +1) = g ;
    *(Image1+ 3*j +2) = r;
    *(array +j)    = 1;
    *(matrix +j)   = 1;

}
else {

    *(Image1+ 3*j)   = 255;
    *(Image1+ 3*j +1) = 255;
    *(Image1+ 3*j +2) = 255;
    *(array +j)   = 0;    // write in 0/1 matrix
    *(matrix + j) = 0;    //write in 0/1 matrix for detect the targer

}

fwrite(Image1+ 3*j, sizeof(unsigned char),1,fp1);
fwrite(Image1+ 3*j +1,sizeof(unsigned char),1,fp1);
fwrite(Image1+ 3*j +2,sizeof(unsigned char),1,fp1);
}
printf("\n");
fwrite(Image1+ 3*Width, sizeof(unsigned char), junk, fp1);

Image=Image+Width_4;
Image1 = Image1 + Width_4;
array = array + Width;
matrix = matrix + Width;
}
printf("
");
break;
}
break;
}

/* *************************************************************************/
Print the target region after filter by using 0/1 matrix
/* *************************************************************************/

printf( "Print the 0/1 matrix after filter:\n");     // show the target region
printf("\n");
for(i=0; i<Height; i++){
    array = array - Width;
    for(j=0;j<Width;j++)
        {
            printf("%d ", *(array+j));
        }
    printf("\n");
}

matrix = matrix - Width*Height;                    // pointer to zero

/* *************************************************************************/
Pattern recognition for each fragments
/* *************************************************************************/

frag_inc = 0;
for(i=0; i<Height; i++){
    for(j=0;j<Width;j++)
        {
            if(*(*(array + j) > 0)

n_set = 1;
*(pixel_col + n_set)= j;
*(pixel_row + n_set)= i;

nrow = Height;
ncol = Width;

c0 = j;
row0 = i;

c0_up = c0+1;            // initialise search bounds
col_dn = c0-1;
row_dn = row0-1;
row_up = row0+1;

/************************* //check edges
if( row_up<nrow ){
    edge1 = 1;
} else{
    edge1 = 0;
    row_up = row_up - 1;
}

/****************************
if( col_up<ncol ){
    edge2 = 1;
} else{
    edge2 = 0;
    col_up = col_up - 1;
}

/****************************
if( row_dn >= 0 ){
    edge3 = 1;
} else{
    edge3 = 0;
    row_dn = 0;
}

/****************************
if( col_dn >= 0 ){

edge4 = 1;
}
else{
    edge4 = 0;
    row_dn = 0;
}

do {  // search along the four sides bounding the
    n_inc0 = 0;  // set counter for this search of the 3 edges to 0
    del_col_dn = 0;  // initialise values used to store changes in edge positions
    del_col_up = 0;
    del_row_dn = 0;
    del_row_up = 0;
    if ( edge1 > 0 ) {
        n_inc = 0;  // set counter for this edge to 0
        for (col=col_dn; col<=col_up; col++) {
            if (*(matrix + row_up*Width + col) > 0){
                n_inc++;
                *(pixel_col + n_set + n_inc) = col;
                *(pixel_row + n_set + n_inc) = row_up;
            }
        }
        if ( n_inc > 0) {
            del_row_up = 1;  // search further out next time if pixels were found
        }
    }
    n_set = n_set + n_inc;  // update count of set pixels found altogether
    n_inc0 = n_inc0 + n_inc;  // update count of set pixels found in this
                                // search of all edges

if ( edge2 > 0 ) {
    n_inc = 0;                      // set counter for this edge to 0
    for (row=row_dn+1; row<=row_up-1; row++) {    // set bounds to avoid double-counting corner pixels
        if (*(matrix + row*Width + col_up) > 0){
            n_inc++;
            *(pixel_col + n_set + n_inc) = col_up;
            *(pixel_row + n_set + n_inc) = row;
        }
    }
    if ( (n_inc > 0) || (*(matrix + row_up*Width + col_up) > 0) || (*(matrix + row_dn*Width + col_up) > 0) ) {
        del_col_up = 1; // need to check corners too, but not double-count them
    }
    n_set = n_set + n_inc;    // increment counters for number of set pixels found
    n_inc0 = n_inc0 + n_inc;
}
if ( edge3 > 0 ) {
    n_inc = 0;                      // set counter for this edge to 0
    for (col=col_dn; col<=col_up; col++) {
        if (*(matrix + row_dn * Width + col) > 0) {
            n_inc++;
            *(pixel_col + n_set + n_inc) = col;
        }
    }
}
*(pixel_row + n_set + n_inc) =  row_dn;
}
}
if ( n_inc > 0) {
    del_row_dn = -1;
}

n_set = n_set + n_inc;
n_inc0 = n_inc0 + n_inc;
}
if ( edge4 > 0 ) {
    n_inc = 0;                      // set counter for this edge to 0
    for (row=row_dn+1; row<=row_up-1; row++) {     // set bounds to avoid double-
        if (*(matrix + row * Width + col_dn) > 0) {
            n_inc++;
            *(pixel_col + n_set + n_inc) =  col_dn;
            *(pixel_row + n_set + n_inc) =  row;
        }
    }
    if ( (n_inc > 0) || (*(matrix + row*Width + col_dn) > 0) || (*(matrix + row_dn*Width + col_dn) > 0) ) {
        del_col_dn = -1; // need to check corners too, but not double-count them
    }
}
n_set = n_set + n_inc;  // increment counters for number of set pixels found
n_inc0 = n_inc0 + n_inc;

}
col_up += del_col_up;
col_dn += del_col_dn;
row_up += del_row_up;
row_dn += del_row_dn;

/****************/  // check edges
if( row_up<nrow ){
   edge1 = 1;
}
else{
   edge1 = 0;
   row_up = row_up - 1;
}

/*****************/
if( col_up<ncol ){  
   edge2 = 1;
}
else{
   edge2 = 0;
   col_up = col_up - 1;
}

/****************/  
if( row_dn>=0 ){  
   edge3 = 1;
}
else{  
   edge3 = 0;
   row_dn = 0;
}

/*****************/
if( col_dn>=0){  
   edge4 = 1;
}
else{
    edge4 = 0;
    row_dn = 0;
}

} while( n_inc0 > 0);

*(pixel_col) = n_set;
*(pixel_row) = n_set;

npoint = n_set;
cmean = 0.0;
rmean = 0.0;

for (k=1; k<=npoint; k++) {
    cmean = cmean + *(pixel_col + k);
    rmean = rmean + *(pixel_row + k);
}

cmean = cmean / npoint;              // calculate mean value of column index
rmean = rmean / npoint;              // calculate mean value of row index

/***********************************************************/
*(frag_c + frag_inc) =  cmean;      // store centroid coordinates
*(frag_r + frag_inc) =  rmean;
*(frag_n + frag_inc) =  npoint;       // store size of cluster
frag_inc++;

for (n=1; n<=npoint; n++)        //set the pixel value as zero after count
{
    *(matrix + (*(pixel_row + n)) * Width + *(pixel_col + n)) = 0;
    *(array - i * Width + (*(pixel_row + n)) * Width + *(pixel_col + n)) = 0;
}
array = array + Width;

printf("
");

nfrag = frag_inc;
printf("nfrag= : %d ", nfrag);
printf("n");

/***********************************************************************************/
for(i=0;i<nfrag;i++)                    //print the list of fragments
{
    printf("i= : %d ", i);
    printf("frag_c= : %f ", *(frag_c + i));
    printf("frag_r= : %f ", *(frag_r + i));
    printf("frag_size= : %d ", *(frag_n + i));
    printf("n");
}

/***********************************************************************************/

Merge fragment into clusters
***********************************************************************************/

pair_inc = 0;

for(i=0;i<nfrag;i++)
{
    for(j=0;j<nfrag;j++)
    {
        if(j!=i)                                      // the element value in distance matrix
        {
            d2 = (*(frag_c+j) - *(frag_c+i))*(*(frag_c+j) - *(frag_c+i))
            + (*(frag_r+j) - *(frag_r+i))*(*(frag_r+j) - *(frag_r+i));

            if(d2<=*(frag_n+i)){    // compare the distance with threshold
                *(pair_1 + pair_inc) = i;       // store the pair
                *(pair_2 + pair_inc) = j;
                pair_inc = pair_inc + 1;
            }
        }
    }
}
pair_inc++;
}
}
}

npair = pair_inc;

//****************************************************************************// print pair
for(i=0;i<npair;i++)
{
printf("No. = %d: ", i);
printf("pair_1 = : %d ", *(pair_1 + i));
printf("pair_2 = : %d ", *(pair_2 + i));
printf("\n");
}
//****************************************************************************// find the fragment without repeated
pair_long = 2*npair;
for(n=0;n<npair;n++)
{
*(pair + n) = *(pair_1 + n);
*(pair + npair + n) = *(pair_2 + n);
}
for(s=0;s<pair_long;s++)
{
*(pair + s) = *(pair + s);
}
for (p= 0 ; p < pair_long -1 ; p++)
{
for ( q = p + 1 ; q < pair_long ; q++) //check element whether repeated or not
{
if *(pair + q) == *(pair + p)  // if repeated element occur
{
   // put the element forward in temp array
   for (s = q + 1; s < pair_long; s++)
   {
      *(pair + s -1) = *(pair + s);
   }
//delete the number of temp array by one
pair_long = pair_long - 1;

//check again to avoid the omission
q = q - 1;
}
}
}

printf("n");
pair_count = 0;
for (l = 0; l < pair_long; l++)
{
 *(pair_list + l) = *(pair + l);
 pair_count++;
}

/***********************************************//pair_list
for(i=0;i<nfrag;i++){
 *(frag + i) = i;
}

frag_long = nfrag;

//delete the element of pair and get the single fragment
for(i=0;i<pair_count;i++){
 for(j=0;j<frag_long;j++){
 if(*(pair_list + i) == *(frag + j)){
 for(s = j+1; s<frag_long; s++){
 *(frag + s -1) = *(frag + s);
 }
 frag_long = frag_long -1;
 j = j -1;
 }
 }
}
single_inc = 0;

for(l=0;l<frag_long;l++){
    *(single + l) = *(frag + l);
    printf(" %d ", *(single + l));
    single_inc++;
}

printf("\n");
nsingle = single_inc;
printf(" single_num= : %d ", nsingle);
printf("\n");
//print single list
for(k=0;k<nsingle;k++){
    printf("cluster_c= : %f ", *(frag_c + *(single + k)));
    printf("cluster_r= : %f ", *(frag_r + *(single + k)));
    printf("cluster_size= : %d ", *(frag_n + *(single + k)));
    *(clust_c + k) = *(frag_c + *(single + k));
    *(clust_r + k) = *(frag_r + *(single + k));
    *(clust_n + k) = *(frag_n + *(single + k));
    printf("\n");
}

/***********************************************************/

clust_inc = 0;

for(i=0;i<npair;i++){
    sect_inc = 1;

    *section_1 = *(pair_1 + i);
    *section_2 = *(pair_2 + i);

    for(j=i+1;j<npair;j++){
        // check the common element in pair
        if((*(pair_1 + j)==*(pair_1 + i))||(*(pair_1 + j)==*(pair_2 + i))
            ||(*(pair_2 + j)==*(pair_1 + i))||(*(pair_2 + j)==*(pair_2 + i)))
        {
            *(section_1 + sect_inc) = *(pair_1 + j);
            *(section_2 + sect_inc) = *(pair_2 + j);
        }
sect_inc++;               //delete repeated element in pair
for(s=j+1;s<npair;s++){
    *(pair_1 + s -1) = *(pair_1 + s);
    *(pair_2 + s -1) = *(pair_2 + s);
}

npair = npair - 1;
    j  =  j  - 1 ;
}
}

nsection = sect_inc;
printf("nsection= %d:  ",nsection);
sect_long = 2*nsection;

if(nsection==1){                                  // only one pair

    printf("\n");
    printf(" %d ",*(section_1));
    printf(" %d ",*(section_2));
    printf(" sect_list= : %d ", sect_long);
    printf("\n");

    // calculate mean value of cluster
    cluster_c = (*(frag_c + *section_1)) * (*(frag_n + *section_1)) 
                 + (*(frag_c + *section_2)) * (*(frag_n + *section_2)) ;

    cluster_r = (*(frag_r + *section_1)) * (*(frag_n + *section_1)) 
                 + (*(frag_r + *section_2)) * (*(frag_n + *section_2)) ;

    cluster_n = (*(frag_n + *section_1)) + (*(frag_n + *section_2)) ;

    cluster_c = cluster_c / cluster_n;
    cluster_r = cluster_r / cluster_n;

    printf(" cluster_c: %f ", cluster_c);
    printf(" cluster_r: %f ", cluster_r);
    printf(" cluster_n: %d ", cluster_n);

    *(clust_c + nsingle + clust_inc) = cluster_c ;
    *(clust_r + nsingle + clust_inc) = cluster_r ;
    *(clust_n + nsingle + clust_inc) = cluster_n ;
clust_inc++;

printf("\n");
printf("\n");
}

else{                                  // more than one pairs have common elements

for(n=0; n < nsection; n++){
    *(section + n) = *(section_1 + n);
    *(section + nsection + n) = *(section_2 + n);
}

for(s=0; s < sect_long; s++){
    *(section + s) = *(section + s);
}

for (p= 0; p < sect_long -1; p++)
{
    for ( q = p + 1; q < sect_long; q++)
    {
        if (*(section+q) == *(section+p))
        {
            for (s = q + 1; s < sect_long; s++)
            {
               *(section + s -1) = *(section + s);
            }
            sect_long = sect_long - 1;
            q = q - 1;
        }
    }
}

printf("\n");
sect_count = 0;
for (l = 0; l < sect_long; l++)
{
    *(sect_list + l) = *(section + l);
    printf(" %d ",*(sect_list + l));
    sect_count++;
}

for(z=0;z<sect_count;z++)
{
    for(t=i+1;t<npair;t++)
    {
        //check the repeated element again
        if(*(pair_1 + t)==*(sect_list + z)||(*(pair_2 + t)==*(sect_list + z))
        {
            for(w=t+1;w<npair;w++)
            {
                *(pair_1 + w -1) = *(pair_1 + w);
                *(pair_2 + w -1) = *(pair_2 + w);
            }
            npair = npair - 1;
            t = t -1;
        }
    }
}

printf(" total_frag= : %d ", sect_count);
printf("\n");
/*************************************/
cluster_c = 0.0;
cluster_r = 0.0;
cluster_n = 0;
for (k=0; k<sect_count; k++)
{
    cluster_c = cluster_c + (*((frag_c + *(sect_list + k)))) * (*((frag_n + *(sect_list + k)))) ;
    cluster_r = cluster_r + (*((frag_r + *(sect_list + k)))) * (*((frag_n + *(sect_list + k)))) ;
    cluster_n = cluster_n + *(frag_n + *(sect_list + k)) ;
}
cluster_c = cluster_c / cluster_n;   // calculate mean value of column index

cluster_r = cluster_r / cluster_n;   // calculate mean value of row index

printf(" cluster_c: %f ",  cluster_c);
printf(" cluster_r: %f ",  cluster_r);
printf(" cluster_n: %d ",  cluster_n);

*(clust_c + nsingle + clust_inc) = cluster_c ;
*(clust_r + nsingle + clust_inc) = cluster_r ;
*(clust_n + nsingle + clust_inc) = cluster_n ;
clust_inc++;

printf("\n");
printf("\n");

} 

/**********************************************************/  //print the list of clusters

ncluster = nsingle + clust_inc;

printf("ncluster= : %d ", ncluster);
printf("\n");

for(i=0;i<ncluster;i++)
{
   printf(" %d:  ", i);
   printf("cluster_c= : %f ", *(clust_c + i));
   printf("cluster_r= : %f ", *(clust_r + i));
   printf("cluster_size= : %d ", *(clust_n + i));
   printf("\n");
}

/***********************************************************/

Identify label and count the number of clusters in each label

/***********************************************************/

pair_inc = 0;
for(i=0;i<ncluster;i++){
    for(j=0;j<ncluster;j++)
    {
        if(j!=i) // distance threshold
        {
            d2_lable = (*(clust_c+j) - *(clust_c+i))*(*(clust_c+j) - *(clust_c+i))
            + (*(clust_r+j) - *(clust_r+i))*(*(clust_r+j) - *(clust_r+i));
            if(d2_lable < 5000){ //judge distance threshold for identify label
                *(pair_1 + pair_inc) = i; //store the pair
                *(pair_2 + pair_inc) = j;
                pair_inc++;
            }
        }
    }
}

npair = pair_inc;

/************************************************/ // print cluster pair
    for(i=0;i<npair;i++)
    {
        printf("No.= %d: ", i);
        printf("pair_1= : %d ", *(pair_1 + i));
        printf("pair_2= : %d ", *(pair_2 + i));
        printf("\n");
    }

/***************************************************/  //find the cluster without repeated
    pair_long = 2*npair;
    for(n=0;n<npair;n++){
        *(pair + n) = *(pair_1 + n);
        *(pair + npair + n) = *(pair_2 + n);
    }
    for(s=0;s<pair_long;s++){
        *(pair + s) = *(pair + s);
for (p = 0; p < pair_long - 1; p++)  // check element whether repeated or not
{
    for (q = p + 1; q < pair_long; q++)  // if repeated element occur
    {
        if (*(pair + q) == *(pair + p))
        {
            // put the element forward in temp array
            for (s = q + 1; s < pair_long; s++)
            {
                *(pair + s - 1) = *(pair + s);
            }

            // delete the number of temp array by one
            pair_long = pair_long - 1;

            // check again to avoid the omission
            q = q - 1;
        }
    }
}

printf("n");
pair_count = 0;

for (l = 0; l < pair_long; l++)
{
    *(pair_list + l) = *(pair + l);
    pair_count++;
}

// cluster pair list
for(i=0;i<ncluster;i++)
    *(clust + i) = i;

clust_long = ncluster;

// delete the element of element of pair and get the single cluster
for(i=0;i<pair_count;i++)
{
    for(j=0;j<clust_long;j++)
    {
        if(*(pair_list + i) == *(clust + j))
        {
            for(s = j+1; s<clust_long; s++)
            {
                *(clust + s -1) = *(clust + s);
            }
            clust_long = clust_long -1;
            j = j -1;
        }
    }
}

single_inc = 0;
for(l=0;l<clust_long;l++)
{
    *(single + l) = *(clust + l);
    printf("%d ",*(single + l));
    single_inc++;
}

printf("\n");
nsingle = single_inc;
printf("single_num= : %d ", nsingle);
printf("\n"); // print single list of cluster
for(k=0;k<nsingle;k++)
{
    printf("label_c= : %f ", *(clust_c + *(single + k)))
    printf("label_r= : %f ", *(clust_r + *(single + k)))
    printf("label_size= : %d ", *(clust_n + *(single + k)))

    *(label_c + k) = *(clust_c + *(single + k));
    *(label_r + k) = *(clust_r + *(single + k));
    *(label_n + k) = *(clust_n + *(single + k));
    *(label_list + k) = 1;

    printf("\n");
}
/************************************************************************************/
printf("n");
label_inc = 0;
list_inc = nsingle;

for(i=0;i<npair;i++){
    sect_inc = 1;
    *section_1 = *(pair_1 + i);
    *section_2 = *(pair_2 + i);

    for(j=i+1;j<npair;j++){
        // check the common element in pair cluster
        if((*(pair_1 + j)==*(pair_1 + i))||(*(pair_1 + j)==*(pair_2 + i))
            ||(*(pair_2 + j)==*(pair_1 + i))||(*(pair_2 + j)==*(pair_2 + i))}
        {
            *(section_1 + sect_inc) = *(pair_1 + j);
            *(section_2 + sect_inc) = *(pair_2 + j);
            sect_inc++;
        }
        // delete repeated element in pair
        for(s=j+1;s<npair;s++){
            *(pair_1 + s -1) = *(pair_1 + s);
            *(pair_2 + s -1) = *(pair_2 + s);
        }
        npair = npair - 1;
        j  =  j  - 1 ;
    }
}

nsection = sect_inc;
printf("cluster list= : ");
sect_long = 2*nsection;

if(nsection==1){
    only one pair cluster in each label
    printf("n");
    printf(" %d ",*(section_1));
    printf(" %d ",*(section_2));
    printf(" sect_list= : %d ", sect_long);
    printf("n");
// calculate mean value of each label
label_cmean = (*(clust_c + *section_1)) * (*(clust_n + *section_1))
  + (*(clust_c + *section_2)) * (*(clust_n + *section_2)) ;

label_rmean = (*(clust_r + *section_1)) * (*(clust_n + *section_1))
  + (*(clust_r + *section_2)) * (*(clust_n + *section_2)) ;

label_size = (*(clust_n + *section_1)) + (*(clust_n + *section_2)) ;

label_cmean = label_cmean / label_size;
label_rmean = label_rmean / label_size;

printf("lable_c: \%f ",  label_cmean);
printf("lable_r: \%f ",  label_rmean);
printf("lable_n: \%d ",  label_size);

*(label_c + nsingle + label_inc) = label_cmean ;
*(label_r + nsingle + label_inc) = label_rmean ;
*(label_n + nsingle + label_inc) = label_size ;
label_inc++;

printf("\n");
printf("\n");
}

else{                          // more than one pairs have common elements
for(n=0;n<nsection;n++){
  *(section + n) = *(section_1 + n);
  *(section + nsection + n) = *(section_2 + n);
}

for(s=0;s<sect_long;s++){
  *(section + s) = *(section + s);
}

for (p = 0; p < sect_long -1; p++)
{
  for ( q = p + 1; q < sect_long; q++)
  {

79
if (*(section+q) == *(section+p))
{
    for (s = q + 1; s < sect_long; s++)
    {
        *(section + s -1) = *(section + s);
    }
    sect_long = sect_long - 1;
    q = q - 1;
}

printf("\n");
sect_count = 0;
for (l = 0; l < sect_long; l++)
{
    *(sect_list + l) = *(section + l);
    printf("%d ",*(sect_list + l));
    sect_count++;
}

for(z=0;z<sect_count;z++){
    for(t=i+1;t<npair;t++){
        // check the repeated element again
        if((*(pair_1 + t)==*(sect_list + z))||(*(pair_2 + t)==*(sect_list + z)))
        {
            for(w=t+1;w<npair;w++){
                *(pair_1 + w -1) = *(pair_1 + w);
                *(pair_2 + w -1) = *(pair_2 + w);
            }
            npair = npair - 1;
            t = t -1;
        }
    }
}
printf(" total_cluster= : %d ", sect_count);
*(label_list + list_inc) = sect_count;
list_inc++;
printf("n");

label_cmean = 0.0;
label_rmean = 0.0;
label_size = 0;
for (k=0; k<sect_count; k++) {
  label_cmean = label_cmean + (*(clust_c + *(sect_list + k))) * (*(clust_n + *(sect_list + k))) ;
  label_rmean = label_rmean + (*(clust_r + *(sect_list + k))) * (*(clust_n + *(sect_list + k))) ;
  label_size = label_size + *(clust_n + *(sect_list + k)) ;
}
label_cmean = label_cmean / label_size;   // calculate mean value of column
label_rmean = label_rmean / label_size;    // calculate mean value of row

printf(" label_c: %f ",  label_cmean);
printf(" label_r: %f ",  label_rmean);
printf(" label_n: %d ",  label_size);
*(label_c + nsingle + label_inc) = label_cmean ;
*(label_r + nsingle + label_inc) = label_rmean ;
*(label_n + nsingle + label_inc) = label_size ;
label_inc++;
printf("n");
printf("n");

}
//print the list of labels

nlabel = nsingle + label_inc;
printf("nlabel= : %d ", nlabel);
printf("\n");

for(i=0;i<nlabel;i++)
{
    printf(" %d :  ", *(label_list + i));
    printf("label_c= : %f ", *(label_c + i));
    printf("label_r= : %f ", *(label_r + i));
    printf("label_size= : %d ", *(label_n + i));
    printf("\n");
}

/*****************************/

printf("\n");

fclose(fp);                   //close the bipmap file
fclose(fp1);

}                                /*XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX*/