A research of applying image warping technique to polarization parameter fitting

Yanhong Zhou
Lehigh University

Follow this and additional works at: http://preserve.lehigh.edu/etd

Recommended Citation
A Research of Applying Image Warping Technique to Polarization Parameter Fitting

January 14, 1995
A Research of Applying Image Warping Technique to Polarization Parameter Fitting

by
Yanhong Zhou

A Thesis Presented to the Graduate and Research Committee of Lehigh University in Candidacy for the Degree of Master of Science

in
EECS Department

Lehigh University

December 7, 1995
Approved and recommended for acceptance as a dissertation in partial fulfillment of the requirements for the Master of Science.

December 7, 1995

/Terrance Boult
Thesis Advisor

Alastair McAulay
Chairperson of
Department
# Table of Contents

Abstract 1

Chapter One
   Introduction 2

Chapter Two
   Overview of Reflection Components Separation, Digital Image Warping, Image Distortion Correction and Image Intensity Linearization 4
      2.1 Reflection Components Separation 4
         2.1.1 Reflection Components 4
         2.1.2 Polarization 4
         2.1.3 Separation of Reflection Components Using Color and Polarization 7
      2.2 Digital Image Warping 7
         2.2.1 Spatial Transformations 7
         2.2.2 Global Splines 8
         2.2.3 Semi-reproducing Kernal Spline 9
      2.3 Image Distortion Correction 10
         2.3.1 Camera Calibration 10
         2.3.2 Distortion Correction 13
      2.4 Image Intensity Linearization 13
         2.4.1 The Linear Relation Between Luminous Reflectance Factor and Image Brightness 13
         2.4.2 The Algorithm of Intensity Linearization 14

Chapter Three
   Programs Used in the Project 15
      3.1 Fitting Polarization Parameters 15
         3.1.1 Algorithm of Fitting 15
         3.1.2 Program Flow 15
         3.1.3 Usage 17
      3.2 Getting Centroids of Circle Chart 17
         3.2.1 Algorithm of Getting Centroids 18
         3.2.2 A Tutorial to the Program of Getting Centroids 19
      3.3 Linearizing Image Intensity 27
         3.3.1 Algorithm of Linearizing Image Intensity 27
         3.3.2 Usage of the Programs 29

Chapter Four
   Experiments 31
      4.1 Introduction 31
      4.2 Experiment Setup 31
      4.3 Checking Equipments 32
      4.4 Experiment One 33
Abstract

The diffuse image can be recovered from strong highlights using an algorithm of separating diffuse and specular components of brightness with polarization and color. The algorithm is implemented by fitting of polarization parameters. In order to apply polarization parameter fitting to multiple cameras, image warping technique is used. Some experiments are done to analyze weather and how warping affects fitting. Objects with different shapes and surface roughness are included in the experiments. Analysis is made according to different regions of images. Experiment of linearizing images are also taken and its affect towards polarization parameter fitting is analyzed.
Chapter One

Introduction

Specular reflections and interreflections produce strong highlights in brightness images. An algorithm, presented by Shree K. Nayar, Xi-Sheng Fang and Terrance Boult[1], of separating the diffuse and specular components of brightness from images using polarization and color information can be used to recover the diffuse image. The algorithm is implemented by placing a polarization filter in front of the camera and rotating the filter to obtain a sequence of images. The polarization parameters are calculated using a fitting technique.

To apply the algorithm in a broader area, multiple cameras are used by putting a polarization filter in front of each camera. Since the angles of the filters are different, rotating is no longer necessary. One of the cameras is used as a reference camera, an image warping technique, presented by Terrance Boult[3], is then applied to warp the images taken by the other cameras to the reference camera and the polarization parameters are calculated with the warped images.

In this project, experiments are done to study whether and how the fitting results are affected by image warping. The question of whether the warping technique can be used with fitting polarization parameters is expected to be answered with the results and analysis of the experiments.

In order to do the image warping, a chart consisting of black circles on a white background is included in the scene and the centroids of the circles for the warping is calculated by a program. The experiments are done by taking a group of eight images taken with four cameras for a fitting, then comparing the fitting results with the group of eight images taken with only the reference camera. Because warping may affect different regions of an image in different ways, objects with different shapes and surface roughness are included in the experiment and result analysis is made according to different regions.

Since the illumination and the image sensors used in the experiments are not perfect, the image intensity is not linear originally. Steps are taken to linearize the images before image warping and polarization parameter fitting, and a compare between the fitting results of linearized images and the original images is made to show how much linearization helps. The imperfection of cameras also cause distortion of the images and corrections are supposed to produce better warping and fitting results. But since the cameras used in this project are not too bad and the objects are neither very close to the cameras, image distortion correction is only talked about theoretically.

Some background knowledge of reflection components separation, digital image warping, image distortion correction and image intensity linearization is briefly introduced in
Chapter Two.

In Chapter Three, the algorithms of fitting the polarization parameters, getting centroids of the chart, linearizing the image intensities are described in detail. The usages of the programs are also given.

Chapter Four describes three experiments taken for the purpose of this project. Experiment one is made to study the situation that the objects are closer to the cameras with a smaller chart, experiment two is taken to study the situation that the objects are farther to the cameras with a bigger chart. There are two parts in experiment three, one part is made to linearize images, the other is made to study how linearization affects fitting. Same error analysis is made for each of the experiment and some conclusion and discussion is derived at the end of the chapter.

Appendix A includes all the source codes of the programs used in this project, appendix B includes the references.
Chapter Two

Overview of Reflection Components Separation, Digital Image Warping, Image Distortion Correction and Image Intensity Linearization

2.1 Reflection Components Separation

In 1992, Shree K. Nayar, Xi-Sheng Fang, and Terrance Boult[1] presented an algorithm of separating the diffuse and specular components of brightness from images. The algorithm is to use polarization and color information simultaneously to obtain constraints on the reflection components at each image point.

2.1.1 Reflection Components

Reflection of light from a surface point is composed of a diffuse component and a specular component. The diffuse component results from light rays penetrating the surface, undergoing multiple reflections and refractions, and re-emerging at the surface. The specular component is a surface phenomenon and results from single reflection of incident light rays, which can either be direct illumination by a light source or diffuse-specular or specular-specular interreflections.

2.1.2 Polarization

Each image brightness value $I$ is the sum of diffuse component $I_d$ and specular component $I_s$, which can be written as:

$$I = I_d + I_s \quad [2.1]$$

If an idea polarization filter is placed in front of the image sensor, light energy may be partially polarized. The extent of polarization depends on several factors including the material of the reflecting surface element, its orientation with respect to the image sensor, and the type of reflection mechanisms at work.

Figure 1 shows a surface element illuminated by a source and imaged by a sensor. A polarization filter is placed in front of the sensor. $n$ is the surface normal, $\alpha$ the phase angle determined by the projection of the surface normal onto the plane of the filter. $\theta$ is the angle of the polarization filter.

The diffuse component tends to be unpolarized while the specular component tends to be partially polarized. In other words, if the surface element is perfectly diffuse in reflectance, rotating the polarization filter does not alter image brightness; if the surface element is not entirely diffuse, rotation of the filter varies the specular component as a cosine function as shown in figure 2.
Therefore, the specular component can be expressed as the sum of a constant $I_{sc}$ and a cosine function with amplitude $I_{sv}$:

Figure 2.1

Figure 2.2
\[ I = I_d + I_{sc} + I_{sv}\cos2(\theta - \alpha) \]  \hspace{1cm} [2.2]

We combine \( I_d \) and \( I_{sc} \) together as \( I_c = I_d + I_{sc} \).

When \( \cos2(\theta - \alpha) \) equals to 1, the maximum value of image brightness is determined as:

\[ I_{\text{max}} = I_c + I_{sv} \]  \hspace{1cm} [2.3]

when \( \cos2(\theta - \alpha) \) equals to -1, the minimum value of image brightness is determined as:

\[ I_{\text{min}} = I_c - I_{sv} \]  \hspace{1cm} [2.4]

From [2.2] we know, for every filter angle \( \theta \), an image brightness is determined. Since we have three unknown factors \( I_c, I_{sv} \), and \( \alpha \), three discrete filter angles are needed to solve the unknowns. But from [2.2] and figure 2, we see two solutions to \( \alpha \) exist, which are \( \alpha \) and \( \alpha + \pi \). This ambiguity cannot be resolved unless at least one more filter setting is used.

The degree the polarization at a scene point can be estimated as:

\[ \rho = \frac{I_{\text{max}} - I_{\text{min}}}{I_{\text{max}} + I_{\text{min}}} \]  \hspace{1cm} [2.5]

\( \rho \) lies between 0 and 1 and can be used in highlight removal to classify points into those that are only diffuse and those that include a specular component.

Another term called Fresnel ratio \( q \) is determined as:

\[ q = \frac{I_{sc} + I_{sv}}{I_{sc} - I_{sv}} = \frac{F_{\perp}(\eta, \psi)}{F_{\parallel}(\eta, \psi)} \]  \hspace{1cm} [2.6]

where \( F_{\perp}(\eta, \psi) \) and \( F_{\parallel}(\eta, \psi) \) are Fresnel coefficients.

From the above it shows, although \( I_c \) and \( I_{sv} \) can be computed, \( I_d \) and \( I_{sc} \) are still unknown unless we know Fresnel ratio \( q \).

In this project, research is only made towards the fitting of \( I_{\text{max}}, I_{\text{min}} \) and \( \alpha \) of each image point, the separation of \( I_d \) and is going to be reviewed very briefly as in 2.1.3.
2.1.3 Separation of Reflection Components Using Color and Polarization

The algorithm of separating reflection components from images are based on the following three assumptions:

(1) The scene consists of dielectric objects;

(2) Specular interreflections results from either the diffuse-specular mechanism or the specular-specular mechanism but not both;

(3) The Fresnel coefficients are independent of the wavelength of incident light.

Since there are not sufficient constraints to compute \( I_d \), neighboring image points are used to do the estimation. Eventually, \( I_d \) is computed by \( I_{\text{min}} \), \( I_{sv} \) and a number of neighboring image points.

2.2 Digital Image Warping

George Wolberg[4], in his book Digital Image Warping, talks about digital image warping very thoroughly. Digital image warping is an image processing that deals with the geometric transformation of digital images. A geometric transformation is an operation that redefines the spatial relationship between points in an image. A warp can be a translation, scale, rotation, convoluted transformation, etc..

Image warping is applied in many fields including remote sensing, medical imaging, computer vision and computer graphics.

2.2.1 Spatial Transformations

Spatial transformations is one of the three components that comprise all geometric transformations. The other two are resampling and antialiasing.

A spatial transformation defines a geometric relationship between each point in the input and output images. The basis of geometric transformation is the mapping of one coordinate system onto another. A mapping function establishes a spatial correspondence between all points in the input and output images.

A spatial transformation can be expressed as

\[
[x, y] = [X(u, v), Y(u, v)]
\]  \hspace{1cm} [2.7]

or

\[
[u, v] = [U(x, y), V(x, y)]
\]  \hspace{1cm} [2.8]
where \([u, v]\) refers to the input image coordinates and \([x, y]\) refers to the output image coordinates, and \(X, Y, U, V\) are arbitrary mapping functions that uniquely specify the spatial transformation.

There are many kinds of spatial transformations including affine transformations, perspective transformations, bilinear transformations, polynomial transformations, piecewise polynomial transformations, global splines, etc. The warping used in this thesis is one of the algorithms of global splines.

2.2.2 Global Splines

Global splines is a general solution to the problem posed by inferring a mapping function given only sparse scattered correspondence points.

Global splines consists of the following procedure:

1. Define a set of basis functions \(h_i(x, y)\), where \(i = 1, ..., K\).

2. Define a set of correspondence points \((x_j, y_j, u_j)\), where \(j = 1, ..., M\), and \(u_j\) refers to the surface height associated with point \((x_j, y_j)\).

3. Define the interpolating function to be a linear combination of these basis functions. The interpolation function is referred as a spline, it is given as:

\[
U(x, y) = \sum_{i=1}^{K} a_i h_i(x, y) \tag{2.9}
\]

for some \(a_i\).

4. Determine the unknown \(a_i\) coefficients by solving a system of linear equations to ensure that the function interpolates the data. The system of equations is given as \(U = HA\), or equivalently as

\[
\begin{bmatrix}
  u_1 \\
u_2 \\
\vdots \\
u_x
\end{bmatrix} = \begin{bmatrix}
h_1(x_1, y_1) & h_2(x_1, y_1) & \cdots & h_k(x_1, y_1) \\
h_1(x_2, y_2) & h_2(x_2, y_2) & \cdots & h_k(x_2, y_2) \\
\vdots & \vdots & \ddots & \vdots \\
h_1(x_M, y_M) & h_2(x_M, y_M) & \cdots & h_k(x_M, y_M)
\end{bmatrix} \begin{bmatrix}
a_1 \\
a_2 \\
\vdots \\
a_k
\end{bmatrix}
\tag{2.10}
\]

The matrix \(H\) is often called the design matrix or the Gram matrix of the problem.
Although the definition of this approach is rather simple, the choice of the basis functions is nontrivial.

In this project, we apply an algorithm called semi-reproducing kernel spline presented by Boult and Kender in 1986.

2.2.3 Semi-reproducing kernel spline

The following is the algorithm of same-reproducing kernel spline.

To interpolate \( M \) data points for mapping function \( V \), the expression is

\[
U(x, y) = \sum_{i=1}^{M} a_i h_i(x, y)
\]

where the basis functions \( h_i \) are

\[
h_i(x, y) = \theta \cdot [(x - x_i)^2 + (y - y_i)^2] \cdot \log[(x - x_i)^2 + (y - y_i)^2], \quad i = 1, \ldots, M
\]

\[
h_{M+1}(x, y) = 1
\]

\[
h_{M+2}(x, y) = x
\]

\[
h_{M+3}(x, y) = y
\]

for a constant \( \theta \).

The above expression for \( U \) has more basis functions than data points, thus the design matrix is insufficient to define the coefficients of the interpolation spline. Instead, the \( M+d \) basis function coefficients can be determined from the solution of \( (M + d) \times (M + d) \) dense linear system:

\[
\begin{bmatrix}
u_1 \\
u_2 \\
\cdots \\
\cdots \\
u_M \\
0 \\
0
\end{bmatrix}
= \begin{bmatrix}
a_1 \\
a_2 \\
\cdots \\
\cdots \\
a_M \\
a_{M+1} \\
a_{M+2} \\
a_{M+3}
\end{bmatrix}
\]
where
\begin{align*}
A_{i,j} &= h_i(x_i, y_j) & \text{for } i \leq (M + d), j \leq M, i \neq j \\
A_{i,j} &= \beta^{-1} h_i(x_i, y_j) & \text{for } i = j \leq M \\
A_{i,j} &= h_j(x_i, y_j) & \text{for } i \leq M, M < j \leq M + d \\
A_{i,j} &= 0 & \text{for } i > M, j > M
\end{align*}

The advantages of this algorithm are: the solution of the linear system is well-understood, the algorithm results in functional forms for the surface allowing symbolic calculations, there is no problem with slower convergence away from information points or near the boundary, the algorithm can efficiently allow updating the information, no iteration is needed since the computation depends only on the number of information points.

### 2.3 Image Distortion Correction

Because of optical imperfections in cameras, every image taken by cameras has distortion. Figure 3 shows an original postscript file of black grids on white background, figure 2.3 shows the original image (distorted) taken by a CCD camera.

In this section, we describes briefly camera calibration and the algorithm to do distortion correction.
2.3.1 Camera Calibration

The camera calibration problem is to relate the locations of pixels in the image array to points in the scene. Since each pixel is imaged through perspective projection, it corresponds to a ray of points in the scene. In another word, camera calibration problem is to determine the equation of this ray from image coordinate system to the real world coordinate system. The camera calibration includes determining two sets of orientations: the extrinsic orientation and intrinsic orientation.

2.3.1.1 Extrinsic Orientation

The problem of exterior orientation is to determine the relationship between the image plane coordinates \((x', y')\) and the 3D world coordinates \((x, y, z)\).

Let the focal length of the camera be \(f\). Let the position of a point is \((x_u, y_u)\) in ideal image plane, \((x_d, y_d)\) in actual image plane, \((x, y, z)\) in 3D world coordinates and \((x_c, y_c, z_c)\) in 3D camera coordinates.

The rigid body transformation from world coordinates to camera coordinates is:

\[
\begin{align*}
x_c &= r_{xx}x + r_{xy}y + r_{xz}z + p_x \\
y_c &= r_{yx}x + r_{yy}y + r_{yz}z + p_y \\
z_c &= r_{zx}x + r_{zy}y + r_{zz}z + p_z.
\end{align*}
\]  

The projection of a point from camera coordinates to the ideal image coordinates is:

\[
\begin{align*}
x_u &= f \frac{x_c}{z_c} \\
y_u &= f \frac{y_c}{z_c},
\end{align*}
\]  

The relation between the position in ideal image plane and actual image plane is:

\[
\begin{align*}
x_u &= x_d + D_x \\
y_u &= y_d + D_y
\end{align*}
\]  

where \(D_x\) and \(D_y\) are the radial lens distortion.

2.3.1.2 Intrinsic Orientation
The problem of intrinsic orientation is to determine the internal geometry of the camera. The geometry is represented by a set of camera parameters:

- \( f \): effective focal length, or image plane to projective center distance;
- \( k_1, k_2 \): lens distortion coefficients;
- \( s_x \): uncertainty scale factor for \( x \), due to TV camera scanning and acquisition timing error;
- \((C_x, C_y)\): computer image coordinate for the origin in the image plane.

2.3.1.3 Calibrating the Camera

In 1986, Roger Y. Tsai\[2\] presented "An Efficient and Accurate Camera Calibration Technique for 3D Machine Vision". He introduced four steps of transformation from 3D world coordinate to computer image coordinate, which is the algorithm we used for camera calibration in this project.

The four steps are as followings:

1. Rigid body transformation from the object world coordinate system to the camera 3D coordinate system, as shown in equation [2.15].

2. Transformation from 3D camera coordinate to ideal image coordinate using perspective projection with pinhole camera geometry, as shown in equation [2.16]. In this step, effective focal length needs to be calibrated.

3. Radial lens distortion as shown in equation [2.17].

\[
D_x = X_d(k_1r^2 + k_2r^4)
\]
\[
D_y = Y_d(k_1r^2 + k_2r^4)
\]
\[
r = (X_d^2 + Y_d^2)^{1/2}
\]

In this step, \( k_1 \) and \( k_2 \) need to be calibrated.

4. Real image coordinate \((X_d, Y_d)\) to computer image coordinate \((X_l, Y_l)\) transformation

\[
X_l = s_xd_x^{-1}X_d + C_x
\]
\[
Y_l = d_y^{-1}Y_d + C_y
\]

where

\((X_l, Y_l)\): row and column # of image pixel in computer frame memory
\((C_x, C_y)\): computer image coordinate for origin in image plane

12
\[ d_x' = d_xN_{cx}/N_{fx} \]
\[ d_y' = d_yN_{cy}/N_{fy} \]
\[ d_x: \text{distance between adjacent sensor elements in X direction} \]
\[ d_y: \text{distance between adjacent sensor elements in Y direction} \]
\[ N_{cx}: \text{number of sensor elements in X direction} \]
\[ N_{cy}: \text{number of sensor elements in Y direction}. \]

In this step, uncertainty image scale factor \( s_x \) and the image origin \((C_x', C_y')\) need to be calibrated.

### 2.3.2 Distortion Correction

After the camera is calibrated, we know all the intrinsic parameters of the camera. Then we apply equation [2.19] to get \((X_d, Y_d)\), apply equation [2.18] to get \((D_x, D_y)\) and apply equation [2.17] to get the ideal image coordinate \((X_u, Y_u)\). In this way the distortion of an image can be corrected.

Figure 2.4 shows the corrected image of the image shown in figure 2.3. Actually, Tsai's algorithm gives an average accuracy of 1/4000 over the field of view, or 1/8000 over the depth.

![Figure 2.4](image.png)

**2.4 Image Intensity Linearization**

During the process of taking images, illumination is not constant and not linear. Thus we need to do brightness linearization.

#### 2.4.1 The Linear Relation Between Luminous Reflectance Factor and Image Brightness
The luminous reflectance factor (called albedo) and the brightness L under an illumination has the linear relation as in figure 2.5.

\[
\begin{array}{c}
\text{Brightness} \\
\hline
\text{Albedo}
\end{array}
\]

Figure 2.5

Thus if we have a chart with fixed albedo and relative brightness relation, we can put the chart on the scene and linearize the image intensities. The Macbeth ColorChecker chart is such a chart that can help us do the job.

The Macbeth ColorChecker is a checkerboard array of 24 scientifically prepared colored squares in a wide range of colors. Many of these squares represent natural objects of special interest, such as human skin, foliage and blue sky. These squares are not only the same color as their counterparts, but also reflect light the same way in all parts of the visible spectrum. And because of this unique feature, the squares will match the colors of natural objects under any illumination and with any color reproduction process.

Since we process black and white images, we only focus on the left most column. There are six squares on the column, they are white, light gray (neutral 8), light-medium gray (neutral 6.5), medium gray (neutral 5), dark gray (neutral 3.5) and black from top to bottom, with albedo 0.9, 0.591, 0.362, 0.198, 0.09 and 0.031 respectively.

2.4.2 The Algorithm of Intensity Linearization

The algorithm of intensity linearization is described in 3.3.1.
Chapter Three

Programs Used in the Project

In this chapter we introduce some major programs used in this project. They are programs of polarization parameter fitting, getting centroids of the circle chart, and linearizing the image intensity.

3.1 Fitting Polarization Parameters

The input of the polarization fitting are images of a same scene taken at different polarization filter angles. The output are 6 images for each color channel, $I_{\text{min}}$, $I_{\text{max}}$, $I_{\text{avg}}$ (average of $I_{\text{max}}$ and $I_{\text{min}}$), $\rho$ (degree of polarization), $\alpha$ (phase angle), and root-mean-square-error RMSE of fitting.

The program of fitting polarization parameters used in this project is implemented by my advisor Dr. Terrance Boult[1], [3].

3.1.1 Algorithm of Fitting

According to equation [2.2], we have $I_i$ for any given filter angle $\theta_i$:

$$I_i = I_c + I_s \cos 2(\theta_i - \alpha) \quad [3.1]$$

The cosine term can be written in the following vector notation:

$$f_i = (1, \cos 2\theta_i, \sin 2\theta_i) \quad [3.2]$$

$$v = (I_c, I_s \cos 2\alpha, I_s \sin 2\alpha) \quad [3.3]$$

to obtain:

$$I_i = f_i \cdot v \quad [3.4]$$

A group of equations [3.4] form a matrix equation:

$$I = f \cdot v \quad [3.5]$$

The fitting program solves $I_c$, $I_s \cos 2\alpha$ and $I_s \sin 2\alpha$ for every pixel with matrix equation [3.5], the number of equations is the number of discrete filter angles.

3.1.2 the Program Flow

A brief program flow is described as the following:
(1) Load images and parameters

Polarization fitting requires at least 3 orientations to get a robust result, we use 8 images in the project. The format of images can be ut, jpl, bw and pgm.

Parameters which have to be included in a file are image type, image size, input image file names, filter angles, etc.

(2) Fitting

First, a matrix equation [3.5] is formed by a group of equations [3.4]. Second, an average of pixel intensities from all images at one point is computed. If the average is too low (lower than 4) which means the pixel is too dark, we skip the computation and gives an output as:

\[ I_{\text{max}} = \text{pixel value}; \]
\[ I_{\text{min}} = \text{pixel value}; \]
\[ \theta = 0; \]

degree of polarization \( pp = 0.0625; \)

RMSE of image = -1.0.

Then \( I_c, I_s \cos 2\alpha, I_s \sin 2\alpha \) and \( I_v \) are calculated.

(3) Error Calculation

Knowing \( I_c, I_v \) and \( \alpha \), a theoretical intensity of an image point can be calculated, however, there is always a difference from the theoretical value to the measured value. The root mean square value of this difference is the error for the image point. In order to make it easier to observe in the output image, the number is multiplied by 10.

(4) Output images

Output images are \( I_{\text{max}}, I_{\text{min}}, I_{\text{avg}}, \) degree of polarization, phase angle \( \alpha \) and RMSE.

The image of \( I_{\text{max}} \) is got by adding \( I_c \) to \( I_v \). If \( I_{\text{max}} \) is greater than 255, it is modified to 255.

The image of \( I_{\text{min}} \) is got by subtracting \( I_v \) from \( I_c \). If \( I_{\text{min}} \) is less than .00001, it is modified to .00001.
The image of $I_{av}$ is got by averaging $I_{max}$ and $I_{min}$.

We get image $2\alpha$ by doing $(\sin^{-1}(I_{sv}\sin2\alpha / I_{sv}\cos2\alpha) + 90)$. The region of $\sin^{-1}(I_{sv}\sin2\alpha / I_{sv}\cos2\alpha)$ is from -90 to 90, in order to make it positive, 90 is added to make the region range from 0 to 180.

The image of degree of polarization is got by dividing $I_c$ to $I_{sv}$, in order to be easier for observation, it is multiplied by 100.

3.1.3 Usage

The usage of the polarization parameter fitting program is:

```
program-name parameter-file-name.
```

The format for a parameter file is:

```
number of images
image type
output base file name
start of image in x axis, end of image in x axis, start of image in y axis, end of image in y axis
name of input[0]  filter angle[0]  calibration file[0]

......

name of input[n]  filter angle[n]  calibration file[n]
```

where n is number of images.

3.2 Getting Centroids of Circle Chart

The circle chart is made by taping a 8.5" by 11" xerox paper containing 81 evenly spaced black circles on a paper box. The centroids of circles are used for calibration and warping. We use circles instead of squares as the calibration pattern because centroids of circles are easy to get and less influenced by errors.

Figure 3.1 shows the circle chart. The two circles with a white hole on each are used for orientation. After taking images with the chart on the scene, the centroids of circles are to be
calculated and numbered starting from the circle with hole on the outer side as #1 and the other circle with hole as #2.

3.2.1 Algorithm of Getting Centroids

3.2.1.1 Calculating of centroids

The centroid \((X_c, Y_c)\) of every circle on the chart is calculated as:

\[
X_c = \frac{\sum X_i \omega_i}{\sum \omega_i} \quad \text{[3.6]}
\]

\[
Y_c = \frac{\sum Y_i \omega_i}{\sum \omega_i} \quad \text{[3.7]}
\]

where \((X_i, Y_i)\) is the position of a pixel of the circle, \(\omega_i\) is the weight of that pixel.

Weight \(\omega_i\) is calculated as:

\[
\omega_i = 1 \quad \text{if } I_i \leq I_{in}
\]

\[
\omega_i = 0 \quad \text{if } I_i > I_{out}
\]

\[
\omega_i = \frac{I_{out} - I_i}{I_{out} - I_{in}} \quad \text{[3.8]}
\]

where \(I_{out}\) and \(I_{in}\) are thresholds, any intensity higher than \(I_{out}\) is weighted 0 and any intensity higher than \(I_{in}\) is weighted 1. The darker the intensity is, the higher weight it has.

Figure 3.2 shows \(I_{in}\), \(I_{out}\) and \(I_i\).
3.2.1.2 Removing noise

The program first binarizes the image with threshold \( I_{\text{out}} \). Then it marks every unconnected foreground object with a number. At this stage, the objects are not only circles, but also a lot of noise.

The program with -c option takes average size of all objects, if any object is 3 times larger or 1/3 smaller than the average, the program will remove it from the objects and consider it as background or noise.

The program with -p option does both size checking and shape checking. For size checking, it relies on user-defined maximum and minimum size of the objects. User has to estimate the maximum and minimum size in pixel and input it as a parameter to the program. Any object larger than the maximum size or smaller than the minimum size will be removed. For shape checking, the program checks the shape of the objects. It assumes every object is round and gets its radius. Then it checks if the distance from any point of the object to the centroid is too much larger or smaller than radius, if so, delete it.

3.2.1.3 Ordering centroids

After centroids are calculated and noise is removed, the program orders the centroids with the outer black circle with a hole as #1 and the other black circle with a hole as #2. Because camera can be put at any position, the circles may be distorted, which means the distance between centroid to centroid on the image will never be same as in the original chart. The makes ordering not very easy.

The program completes the job of ordering centroids in the following steps:

(1) It searches the two orientation circles and numbers them.

(2) It calculates the distance between the centroids of the two orientation circles.

(3) It predicts the next centroid as the distance of two previous circles. The program finds the real centroid which is closest to the predicted one from all the centroids, numbers this centroid, takes the distance of this centroid to the one before it and uses it to do next prediction. The program repeats this procedure until it gets to the end of a row.

(4) It comes back to the first centroid of the row and predicts the first centroid for the next row. And another row of centroids are ordered as described in step (3). The program also repeats this procedure until it finished the last row.

3.2.2 A Tutorial to the Program of Getting Centroids
The program is implemented by Yanhong Zhou, it works for greyscale images in pgm format.

3.2.2.1 Functions and usage

The program has the following four functions:

(1) Get the histogram of an image.
(2) Get a binary image using a user specified threshold.
(3) For good quality images, get centroids without specification of size limits of the circles.
(4) For bad quality images, get centroids with specification of size limits of the circles.

The flags and arguments of the program are used as follow:

(1) program -h input output
   program: the name of the executable program
   -h: option of getting histogram
   input: input image
   output: output image of histogram

(2) program -b input threshold output
   program: the name of the executable program
   -b: option of getting histogram
   input: input image
   threshold: threshold specified by user for getting binary image
   output: output binary image

(3) program -c input top bottom output1 output2
   program: the name of the executable program
   -c: option of getting centroids without specification of size limit of circles
   input: input image
   top: top-threshold specified by user (please refer the algorithm)
   bottom: bottom-threshold specified by user (please refer the algorithm)
   output1: output image of the marked centroids
   output2: output text of the positions of the centroids

(4) program -p input top bottom max-size min-size output1 output2
   program: the name of the executable program
   -p: option of getting centroids with specification of size limits of circles
   input: input image
   top: top-threshold specified by user (please refer the algorithm)
   bottom: bottom-threshold specified by user (please refer Algorithm)
   max-size: the possible maximum size of circles specified by user
   min-size: the possible minimum size of circles specified by user
   output1: output image of the marked centroids
3.2.2.2 Steps of running the program

To run the program, please take the following steps:

Step One: Make a circle chart.

The program requires that the chart has to be a piece of material with dark, round and evenly spaced circles on a light background. To determine the orientation, there has to be a circle with a hole on the upper left corner of the paper and another circle with a hole next on the right side of it.

Step Two: Take an image using the camera that needs to be calibrated.

The image has to be mainly the chart, not something irrelevant. The illumination when taking the image has to be good enough to make the circles out of the background. The holes on the circles have to be clear enough and they can not be too small.

The image can be rotated and the camera can be put at any angle. The program can handle those situations.

Figure 3.3 and 3.4 show two good images, they are well illuminated and the orientation holes are clear also.

Figure 3.3
Figure 3.4

Figure 3.5 and 3.6 show two bad images. There is too much irrelevant background in the first image. In the second image, the orientation holes are too small to make out.

Step Three: Check on histogram.

Before getting centroids, threshold for getting the binary image has to be known. To choose an appropriate threshold $I_{out}$ run the program with the parameter of -h.
There are scales on the histogram graph. Each scale represents 50 grey levels based on the assumption of all the images are 256-grey-level. Hopefully there are two peaks on the histogram if the quality of the image is good. Then pick up the threshold right before the second peak (it doesn't have to be really accurate though).

Figure 3.7 and 3.8 show the histogram of the images in figure 3.3 and 3.4. Thresholds we choose are 80 for the first image and 150 for the second one.

![Figure 3.7](image1)

![Figure 3.8](image2)

**Figure 3.7**

**Figure 3.8**

Step Four: Check with the binary image.

Run the program again with the parameter of -b using the threshold got from the histogram. Check with the output to see if the right threshold is chosen. If circles haven't been made out of the background, the threshold is probably too high; if the circles are too much incomplete compared to the original image, the threshold is probably too low. Adjust the threshold and run the program again, until a satisfying result is got. This threshold is very important to later calculation.

Figure 3.9, 3.10 and 3.11 show three binary images of figure 3.3. In figure 3.9, threshold is 150 and it is just good. In figure 3.10, threshold is 90 and background is not fully separated from circles. In figure 3.11, threshold is 250 and there is too much information lost on foreground.

![Figure 3.9](image3)
Step Five: Get centroids of circles without specifying size limits.

Run the program with the parameter of -c. This option works only when the images have good quality and the camera is not very much tilted. The program takes the average size of all the circles, and assume any object three times larger than or one third as small as the average size to be noise and get rid of them.
The program also orders the centroids after getting them. The circle with a hole at the corner is always numbered 0, the other one with a hole is always numbered 1. Circle number 0 is oriented on the upper left corner of the image. The other circles are ordered 2, 3, 4, 5 ... from left to right and from top to bottom according to the orientation.

Figure 3.12 and 3.13 show the centroids of figure 3.3 and 3.4 respectively. The centroids are marked on the original images.

Step Six: Get centroids of circles with specification of size limits.
Run the program with the parameter of -p. This option calculates the centroids of circles for relatively more noisy and distorted images. We have to estimate the maximum and minimum size of all the circles. If the program complains there is too much noise, we have to modify the size limits and try again. It may take trials to get the right estimation of sizes.

Figure 3.14 and 3.15 show the centroids of circles in two more noisy and distorted images. We choose the maximum size as 1000 pixels and minimum size as 50 pixels. These thresholds of size effectively eliminate the noise.
3.3 Image Intensity Linearization

In the experiments, a T3 bulb which provides us quite constant illumination is used. However, the lighting still changes from time to time, therefore we need to unify the intensities taken by a same camera at different times.

We talked about the background knowledge about intensity linearization in chapter two. A MacBeth ColorChecker chart is used to calibrate the intensity. Some programs is made by me to implement the linearization. In this section, we describe the algorithm and the usage of the program.

3.3.1 Algorithm of Linearizing Image Intensities

First, we need to generate a look-up table of input intensities to output intensities. We put the MacBeth ColorChecker chart on the scene using the same light we use in the experiments. But we found a problem after a few trials, that we can't make the six squares on the chart cover a broad enough range of intensities from low to high. Also, six intensities are not good enough to build a smooth curve of output intensity vs. input intensity.

Here we apply two polarizing filters to help us. According to linear optic, we have the intensity through two polarizers as:

\[ L = L_0 \sin^2 \alpha \]

where \( L \) is the intensity the camera gets through two polarizers, \( L_0 \) is the original intensity, \( \alpha \) is the relative orientation of the two polarizers.

Because \( \sin^2 \alpha \) can change from 0 to 1, if we can get the intensity on the white square of the chart high enough, we can get the whole range of intensities from 0 to 255. So we take images of the chart from 0 to 90 at angles of \( \alpha_1(90), \alpha_2, \alpha_3, \alpha_4 \ldots \alpha_n \). We don't take any image at 0°, because that angle the polarizers blocks all the light from the cameras. The intensities of the white square at those images are \( W_1, W_2, W_3, W_4 \ldots W_n \)(from bright to dark). We map \( W_1 \)(intensity of the white square at 0, usually between 220 to 240) to an intensity \( W \), and calculate Max as:

\[ \text{Max} = \frac{W_1}{0.9} \]

where 0.9 is the albedo of the white square. Max must not be larger than 255.

Then we have the following match:

<table>
<thead>
<tr>
<th>white(W1)</th>
<th>match to Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>light gray at ( \alpha_1 )</td>
<td>match to Max \cdot 0.9</td>
</tr>
<tr>
<td>light-medium gray at ( \alpha_1 )</td>
<td>match to Max \cdot 0.591</td>
</tr>
<tr>
<td>Color</td>
<td>Match to</td>
</tr>
<tr>
<td>---------------------</td>
<td>-------------------------------</td>
</tr>
<tr>
<td>medium gray at $\alpha_1$</td>
<td>match to</td>
</tr>
<tr>
<td>dark gray at $\alpha_1$</td>
<td>match to</td>
</tr>
<tr>
<td>black at $\alpha_1$</td>
<td>match to</td>
</tr>
<tr>
<td>white(W2)</td>
<td>match to</td>
</tr>
<tr>
<td>light gray at $\alpha_2$</td>
<td>match to</td>
</tr>
<tr>
<td>light-medium gray at $\alpha_2$</td>
<td>match to</td>
</tr>
<tr>
<td>medium gray at $\alpha_2$</td>
<td>match to</td>
</tr>
<tr>
<td>dark gray at $\alpha_2$</td>
<td>match to</td>
</tr>
<tr>
<td>black at $\alpha_2$</td>
<td>match to</td>
</tr>
<tr>
<td>white(W3)</td>
<td>match to</td>
</tr>
<tr>
<td>light gray at $\alpha_3$</td>
<td>match to</td>
</tr>
<tr>
<td>light-medium gray at $\alpha_3$</td>
<td>match to</td>
</tr>
<tr>
<td>medium gray at $\alpha_3$</td>
<td>match to</td>
</tr>
<tr>
<td>dark gray at $\alpha_3$</td>
<td>match to</td>
</tr>
<tr>
<td>black at $\alpha_3$</td>
<td>match to</td>
</tr>
<tr>
<td>white(W4)</td>
<td>match to</td>
</tr>
<tr>
<td>light gray at $\alpha_4$</td>
<td>match to</td>
</tr>
<tr>
<td>light-medium gray at $\alpha_4$</td>
<td>match to</td>
</tr>
<tr>
<td>medium gray at $\alpha_4$</td>
<td>match to</td>
</tr>
<tr>
<td>dark gray at $\alpha_4$</td>
<td>match to</td>
</tr>
<tr>
<td>black at $\alpha_4$</td>
<td>match to</td>
</tr>
<tr>
<td>white(Wn)</td>
<td>match to</td>
</tr>
<tr>
<td>light gray at $\alpha_n$</td>
<td>match to</td>
</tr>
<tr>
<td>light-medium gray at $\alpha_n$</td>
<td>match to</td>
</tr>
<tr>
<td>medium gray at $\alpha_n$</td>
<td>match to</td>
</tr>
<tr>
<td>dark gray at $\alpha_n$</td>
<td>match to</td>
</tr>
<tr>
<td>black at $\alpha_n$</td>
<td>match to</td>
</tr>
</tbody>
</table>

These matchings build up the skeleton of the look-up table, for the remaining part, we simply use straight lines to connect every two points.

Here we still have two problems, one is what if we find one input intensity corresponds to two or more output intensities, the other is that we sometimes find the curve is not singularly increasing. Actually these problems happen a lot when we take images at many angles. To solve the problems, what we do in the program are: (1) if there are more than one outputs matching to one input, we take the average of the outputs; (2) we start to construct the look-up table from the very low intensity and go up, if we find there is output lower than the last output, we simply delete this point from the table. From my experience, six to eight angles are enough to build...
a effective look-up table.

So far we get the look-up table from $n$ images and we are ready to calibrate the intensities under the same illumination. In order to finish the job as good as it can be, we have one last consideration about it. When we do the experiment, we have to put the chart normal to the camera so it doesn't polarize. As we need to take a lot of images, the intensity of the white square must be same under the same illumination and without polarization. However the lighting changes although we can't sense it with eyes, therefore the color of the white square changes a little bit from image to image. Let's pick the intensity of the white square from an image randomly as a reference and call it as $N$. Then for the intensity every pixel of other images, we need to do an adjustment as the following:

$$L = L_o \cdot \left(\frac{N'}{N}\right)$$

where $L_o$ is the original intensity of a pixel, $L$ is the adjusted intensity, $N'$ is the intensity of the white square of the image.

3.3.2 Usage of the Programs

For image intensity linearization, we need a input-output look-up table first, then we put an input image and get an output image. There three programs of which we will describe the usage at the following:

(1) Getting the average intensity of one square

Each square on the chart takes about 200 image pixels, whose intensities vary a little bit, thus we have to get the average of the intensities. We tell the program the location of the center of each square, and the program will take the average of a 13 by 13 square. The usage of the program is:

```
program input-image input-centers output-file
```

where input-centers is a text file containing the x and y locations of centers of squares in a sequence of $x_1, y_1, x_2, y_2, ..., x_6, y_6$; output data are six average intensity values in the sequence of the input.

(2) Generating the look-up table

We use the above program to get the average of squares for all the images, then we are ready to generate the look-up table.

The usage of the program is:
program NUM input-file output-table

where NUM is the number we want the intensity of the white square at 0° to match to; input-file are the angles at which we take images and 6n numbers which are the average intensities of all the squares for all the images, in a sequence of the match table in page 26 and 27; output-table is the look-up table which contains 255 number indexed from 0 to 255 consequently.

(3) Linearizing intensities

It's easy to linearize an image once we get the look-up table. The usage of the program is:

program input-image lut N N' output-image

where lut is the look-up table, N is the reference average intensity of the white square on the reference image, \(N'\) is the average intensity of the white square of any image taken by the same camera and under the same illumination.
Chapter Four

Experiments

4.1 Introduction

In Chapter two we talked about the algorithm of separating reflection components. By taking images at different polarization angles, we can compute the polarization parameters $I_{am}$, $I_{im}$, and $\alpha$ by linear least square fitting. To get a robust result, we take images at eight different filter angles.

We bundle four cameras together, with each one, we take eight images at different polarizations. We name the cameras cam1, cam2, cam3 and cam4. The polarization angles we use are 0, 30, 90, 140, 170, 220, 260 and 310. We let cam1 to be our reference camera, then warp the 30 image and the 220 image taken by cam2, the 90 image and the 260 image taken by cam3, the 140 image and the 310 image all to cam1. Then we put this group of warped images into the fitting program and compare it with putting a group of images taken only by the reference camera. We hope the results from the two fittings are not much different, which means warping can be applied to the polarization parameter fitting. This is of great significance in the fields of industry, for instance, rotating the filter after taking every image is very impractical in a assembly line.

We need two chart boards for the experiments. One is the Macbeth ColorChecker chart for doing image intensity linearization, the other is the chart for doing image warping and camera calibration, we call it simply circle chart. The circle chart is a piece of foam board with evenly spaced black circles on a white background. The centroids of the circles are reference points for both image warping and camera calibration.

I did a lot of experiments in this project, however, I am only going to include three of those in the thesis. In experiment one, we use a smaller calibration board and the cameras are closer to the objects. In experiment two, we use a bigger calibration board and the cameras are farther to the objects. In experiment three and four, we do intensity linearization to the original images and study the affects they make on warping and fitting.

In this chapter we describe setting up experiment, checking equipment, doing experiments and analyzing results.

4.2 Experiment Setup

The experiment uses four Sony XC-999 CCD color video cameras and four PS-99SU power devices to produce video signal in NTSC format. The video signal is digitized using a Sun VideoPix card to obtain a 640 by 480 RGB image. A Sony 6mm VCL-06S12XM lens is
mounted on each camera to image the scene.

The XC-999 is a ultra-small CCD color video camera module which uses a CCD(Charge Coupled Device), a solid stage image sensor. The camera is very small and light that can be installed almost anywhere. The camera ensures a high resolution image with 768 by 494 picture elements. A HYPER HAD sensor provides the camera with high sensitivity and less smear, and it is possible to shoot under low light condition.

The PS-99SU power device is designed to power and interface Sony's XC-999 CCD camera to standard video system(which use either BNC connectors for NTSC/PAL signals, and Y/C connectors for S-VHS or Y/C signals). In our experiment, we use BNC connectors for NTSC signals.

A linear polarizing filter, mounted in a precision rotation ring, is placed in front of the camera lens. Images are taken at various orientations of the polarizer. The polarizing filter is made of glass with diameter of about 4". It adds some green color to the images.

A switch box is used to control which camera is in use. The output of the four camera power devices go into the input of the switch box, and the output goes to the digitizer and a Sony color monitor. We use the monitor to check up the scene before the real experiment, however, monitor should be off while taking images because there maybe some electromagnetic interference from the monitor to the cables which are connected to the digitizer. Also some video cables are used in the experiment.

The above setup has several minor problems. First, the cameras are not calibrated, there is distortion on the images, however in these experiments, we believe this effect is not serious. Second, the cameras are not corrected for chromatic aberration. We will also do experiments later to learn the difference between fitting with unlinearized images and linearized images. Third, we are using CCD technology which is prone to blooming affects near strong highlight regions. All of these problems manifest themselves as errors in polarization fitting in small neighborhoods of scene boundaries.

A circle chart will be put on the scene for the image warping. Also, some dichromatic objects are put on the scene too. The objects are four plastic eggs, two plastic cubes and one china cup.

Another important factor in the experiment is lighting. We don't use fluorescent lamp because it blinks and causes too much illumination variations. What we use is a 500W T3 bulb. The angle between the lamp, the objects and the cameras should be approximately 120· as shown in figure 2.1. And we don't shine light from directly on the objects because the digitizer has only 8 bits and therefore can handle only 256 gray levels. Instead, we use a reflection board. We shine light on the reflection board and make the board be the light source to illuminate the objects.

### 4.3 Checking Equipments
After all equipments for the experiment are setup, we need to check if they work properly. We want to eliminate systematic errors which come from bad equipments or poor experiment technique as much as possible.

The reasons that can cause the equipment failure can be electricity problem and heat problem. Electricity problem means a device is electrically wrong, or two cables are touching at the switch box; heat problem means devices may behave abnormally if the lab is too warm. We need to make sure all the cameras, switches, cables work well before we start the real experiment.

We check up the equipments by taking dark images with a cap on each lens. If every device is fine, we expect to see a dark image with some small (usually smaller than 20) random noise. If we see some pattern on the image, it mostly indicates there is a electricity problem with one or more devices. In this case, we take off one device and put on a new one at a time, and take a dark image again. After several trials we should be able to find the device(s) that cause the error. If the random noise is too large, there is big chance of heat problem.

I didn't check up the equipments before I took images at first, after running fitting program, I found a patterned error in the image of RMSE, where there was much more significant left-and-right errors than up-and-down errors. Then I checked up the devices one by one and finally found out two cables was not working well. I changed them and got a satisfying result afterwards. There was also a heat problem which made the average RMSE too large. After the lab was cooled down, the result was better.

There are other factors that can cause systematic error, they are unstable rods that hold the cameras and the polarizor, cameras or polarizor movement during the image taking, unperpendicular polarizer to the cameras, cables with too big length, etc. However, these factors can be overcome easily.

### 4.4 Experiment One

In this experiment, we take images using four cameras which are bundled together at angles of 0, 30, 90, 140, 170, 220, 260 and 310. We then warp 30 and 220 images taken by cam2 to cam1, warp 90 and 260 images taken by cam3 to cam1, and warp 140 and 310 images taken by cam4 to cam1 also. Then we put 0 and 170 images taken by cam1 and 6 warped images to the fitting program and compare the result with putting images taken only by cam1 at 8 angles.

In this section, we describe taking images, warping images, fitting of polarization parameters and the error analysis of the results.

#### 4.4.1 Taking images

We take images with every camera at the angles of 0, 30, 90, 140, 170, 220, 260 and 310, there are 32 shots altogether. To reduce noise, 16 frames are averaged for one image.
The image grabber we use is a program called vpx_remote implemented by Dr. Terrance Boult[1], [3]. It grabs either color or black and white image and save it in various optional formats. It takes a dark image before starting to take a real image, then subtract the dark image from the real image later on to cancel the bias of the camera. We save the images in pgm format which separates the color image into channels of red, green and blue. We use only images of red channel.

The scene consists of a circle chart, four plastic eggs, two plastic cubes and one china cup. All the objects are dichromatic so the reflection of light come from those surfaces can get polarized by the polarizor.

Figure 4.2 to Figure 4.9 show the images taken by cam1 at angles of 0, 30, 90, 140, 170, 220, 260 and 310 respectively. Since the time period of the cosine function in equation [2.2] is 180, we notice that in the specular parts of the objects, polarization of 0 image is so much different with the polarization of 90 image. Actually the polarizations of two images with 90 difference of filter angle are not necessarily different, they can be same if they happen to the on the half height of the cosine curve.
The image grabber we use is a program called vpx_remllte implemented by Dr. Terrance Boult[1], [3]. It grabs either color or black and white image and save it in various optional formats. It takes a dark image before starting to take a real image, then subtract the dark image from the real image later on to cancel the bias of the camera. We save the images in pgm format which separates the color image into channels of red, green and blue. We use only images of red channel.

The scene consists of a circle chart, four plastic eggs, two plastic cubes and one china cup. All the objects are dichromatic so the reflection of light come from those surfaces can get polarized by the polarizor.

Figure 4.2 to Figure 4.9 show the images taken by cam1 at angles of 0, 30, 90, 140, 170, 220, 260 and 310 respectively. Since the time period of the cosine function in equation [2.2] is 180, we notice that in the specular parts of the objects, polarization of 0 image is so much different with the polarization of 90 image. Actually the polarizations of two images with 90 difference of filter angle are not necessarily different, they can be same if they happen to the on the half height of the cosine curve.
Figure 4.10 to figure 4.15 show the images taken by cam2 at the angles of 30 and 220, images taken by cam3 at the angles of 90 and 260, images taken by cam4 at the angles of 140 and 310. We can notice the spatial shift from camera to camera because of the different view points.

Figure 4.10

Figure 4.11

Figure 4.12
Figure 4.10 to figure 4.15 show the images taken by cam2 at the angles of 30 and 220, images taken by cam3 at the angles of 90 and 260, images taken by cam4 at the angles of 140 and 310. We can notice the spatial shift from camera to camera because of the different view points.
Figure 4.13

Figure 4.14

Figure 4.15
4.4.2. Image Warping

Now we have all the images we need, we do the following warping:

1. warp the image at 30° by cam2 to the image at 30° by cam1;
2. warp the image at 90° by cam3 to the image at 90° by cam1;
3. warp the image at 140° by cam4 to the image at 140° by cam1;
4. warp the image at 220° by cam2 to the image at 220° by cam1;
5. warp the image at 260° by cam3 to the image at 260° by cam1;
6. warp the image at 310° by cam4 to the image at 310° by cam1.

A software "improc" is used to do the warping job. The software made by Ming-chao Chiang and Dr. Terrance Boult provides an integrated environment for image processing including many basic functions such as thresholding, histogram, warping, morphing, and so on. To warp one image onto another, the program needs the centroids of the circles on the calibration chart from both the source image and the target image.

Figure 4.16 to figure 4.21 show the result images of the warping in the sequence as the warpings that are listed above. It can be noticed that there are dark bands on some images, that is because there are some regions where there is no data.
4.4.2. Image Warping

Now we have all the images we need, we do the following warping:

(1) warp the image at 30° by cam2 to the image at 30° by cam1;
(2) warp the image at 90° by cam3 to the image at 90° by cam1;
(3) warp the image at 140° by cam4 to the image at 140° by cam1;
(4) warp the image at 220° by cam2 to the image at 220° by cam1;
(5) warp the image at 260° by cam3 to the image at 260° by cam1;
(6) warp the image at 310° by cam4 to the image at 310° by cam1.

A software "improc" is used to do the warping job. The software made by Ming-chao Chiang and Dr. Terrance Boult provides an integrated environment for image processing including many basic functions such as thresholding, histogram, warping, morphing and so on. To warp one image onto another one, the program needs the centroids of the circles on the calibration chart from both the source image and the target image.

Figure 4.16 to figure 4.21 show the result images of the warping in the sequence as the warpings that is listed above. It can be noticed that there are dark bands on some images, that is because there are some regions where there is no data.

![Figure 4.16](image1.png)

![Figure 4.17](image2.png)
After warping, we check the warping errors. We take the difference of the source image and the target image, and calculate RMS error, signed maximum, signed minimum, unsigned mean and signed mean of the difference image. All the information of the error check is included in table 4.1.

<table>
<thead>
<tr>
<th>Warping Cases</th>
<th>RMS</th>
<th>Max</th>
<th>Min</th>
<th>Unsigned Mean</th>
<th>Signed Mean</th>
</tr>
</thead>
<tbody>
<tr>
<td>cam2 to cam1 at 30°</td>
<td>8.61</td>
<td>111</td>
<td>-113</td>
<td>6.97</td>
<td>-6.63</td>
</tr>
<tr>
<td>cam2 to cam1 at 220°</td>
<td>5.98</td>
<td>107</td>
<td>-105</td>
<td>4.01</td>
<td>-3.38</td>
</tr>
<tr>
<td>cam3 to cam1 at 90°</td>
<td>5.63</td>
<td>61</td>
<td>-66</td>
<td>3.32</td>
<td>-1.06</td>
</tr>
<tr>
<td>cam3 to cam1 at 260°</td>
<td>5.96</td>
<td>84</td>
<td>-68</td>
<td>3.73</td>
<td>2.04</td>
</tr>
<tr>
<td>cam4 to cam1 at 140°</td>
<td>5.15</td>
<td>35</td>
<td>-96</td>
<td>3.95</td>
<td>-3.65</td>
</tr>
<tr>
<td>cam4 to cam1 at 310°</td>
<td>3.44</td>
<td>43</td>
<td>-95</td>
<td>2.22</td>
<td>-1.03</td>
</tr>
</tbody>
</table>

Table 4.1

Compared to the average intensity of 70 ~ 80 of those images, the errors are not very bad. However, why do we have such large maximums and minimums? If we look at the images, we
After warping, we check the warping errors. We take the difference of the source image and the target image, and calculate RMS error, signed maximum, signed minimum, unsigned mean and signed mean of the difference image. All the information of the error check is included in Table 4.1.

<table>
<thead>
<tr>
<th>Warping Cases</th>
<th>RMS</th>
<th>Max</th>
<th>Min</th>
<th>Unsigned Mean</th>
<th>Signed Mean</th>
</tr>
</thead>
<tbody>
<tr>
<td>cam2 to cam1 at 30°</td>
<td>8.61</td>
<td>111</td>
<td>-113</td>
<td>6.97</td>
<td>-6.63</td>
</tr>
<tr>
<td>cam2 to cam1 at 220°</td>
<td>5.98</td>
<td>107</td>
<td>-10 5</td>
<td>4.01</td>
<td>-3.38</td>
</tr>
<tr>
<td>cam3 to cam1 at 90°</td>
<td>5.63</td>
<td>61</td>
<td>-66</td>
<td>3.32</td>
<td>-1.06</td>
</tr>
<tr>
<td>cam3 to cam1 at 260°</td>
<td>5.96</td>
<td>84</td>
<td>-68</td>
<td>3.73</td>
<td>2.04</td>
</tr>
<tr>
<td>cam4 to cam1 at 140°</td>
<td>5.15</td>
<td>35</td>
<td>-96</td>
<td>3.95</td>
<td>-3.65</td>
</tr>
<tr>
<td>cam4 to cam1 at 310°</td>
<td>3.44</td>
<td>43</td>
<td>-95</td>
<td>2.22</td>
<td>-1.03</td>
</tr>
</tbody>
</table>

Table 4.1

Compared to the average intensity of 70 ~ 80 of those images, the errors are not very bad. However, why do we have such large maximums and minimums? If we look at the images, we
can see there are some highly reflective areas on the plastic eggs and the cup. Because different cameras take images from different views, the highly reflective area of an object changes with camera. The reflection area of an object on the source image won't overlap with the reflection area on the target image. That's why we get big maximums and minimums.

The quality of warping also depends on how far one image point has to warp to the corresponding image point on another image. Usually the farther the distance, the worse quality of the warping. Figure 4.22 shows how we arrange the camera. Cam1 is the reference camera, we warp all the images taken by the other three to it. The distance from cam2 to cam1 is about 1", same as the distance from cam4 to cam1. The distance from cam3 to cam1 is about 1.4". Actually, the cameras can't get closer than this.

![Figure 4.22](image)

We also tried to warped all the images to the center of the four cameras, thus reduced the distance of warping to 0.7". We compared this warping errors to the errors of warping to cam1, the results shows no obvious improvement, which means our warping distance is good enough. Also, there is actually no such a camera in the center of the four cameras, we won't be able to take images at that position and do the parameter fitting.

4.4.3 Polarization Parameter Fitting

Now things are ready for fitting of polarization parameters. Two groups of fittings are to be done, the 8 input images of each one are as the following:

Group one:
1. image at 0° by cam1
2. image at 30° by cam1
3. image at 90° by cam1
4. image at 140° by cam1
5. image at 170° by cam1
Group two:

(1) image at 0° by cam1
(2) image at 30° by cam2 and warped to cam1
(3) image at 90° by cam3 and warped to cam1
(4) image at 140° by cam4 and warped to cam1
(5) image at 170° by cam1
(6) image at 220° by cam2 and warped to cam1
(7) image at 260° by cam3 and warped to cam1
(8) image at 310° by cam4 and warped to cam1

Six output images are got from each fitting, they are images of maximum intensities Imax, minimum intensities Imin, average intensities Iavg, phase angle α, degree of polarization pp and RMSE e. Figure 4.23 to 4.28 show the fitting results of group one, figure 4.29 to 4.34 show the fitting results of group two.

In 3.4.2, we talked about some factors that can cause problems of taking images. Those problems including movement of the polarizing filter and lighting variations manifest themselves as errors in polarization fitting, especially in scene boundaries. This is why in the images of RMSE, we can see the boundaries of almost all the objects.
(6) image at 220° by cam1
(7) image at 260° by cam1
(8) image at 310° by cam1

Group two:
(1) image at 0° by cam1
(2) image at 30° by cam2 and warped to cam1
(3) image at 90° by cam3 and warped to cam1
(4) image at 140° by cam4 and warped to cam1
(5) image at 170° by cam1
(6) image at 220° by cam2 and warped to cam1
(7) image at 260° by cam3 and warped to cam1
(8) image at 310° by cam4 and warped to cam1

Six output images are got from each fitting, they are images of maximum intensities Imax, minimum intensities Imin, average intensities Iavg, phase angle α, degree of polarization pp and RMSE e. Figure 4.23 to 4.28 show the fitting results of group one, figure 4.29 to 4.34 show the fitting results of group two.

In 3.4.2, we talked about some factors that can cause problems of taking images. Those problems including movement of the polarizing filter and lighting variations manifest themselves as errors in polarization fitting, especially in scene boundaries. This is why in the images of RMSE, we can see the boundaries of almost all the objects.
4.4.4. Error Analysis

In this section, results of the two fittings in different regions are compared to show how much warping affects fitting. What is concerned about are the results of degree of polarization $pp$, the difference of the maximum intensity and the minimum intensity ($I_{\text{max}} - I_{\text{min}}$) and the phase angle $\alpha$.

4.4.4.1. Error Analysis in Regions of $I_{\text{max}} - I_{\text{min}}$, Degree of Polarization $pp$ and Phase angle $\alpha$.

First, the difference images $\Delta I_{\text{max}}, \Delta I_{\text{min}}, \Delta I_{\text{avg}}, \Delta pp, \Delta t$ and $\Delta e$ are generated from the results of the two fittings, then the errors of $pp$, ($I_{\text{max}} - I_{\text{min}}$) and $\alpha$ are calculated in the following 10 regions:

Figure 4.33

Figure 4.34
4.4.4. Error Analysis

In this section, results of the two fittings in different regions are compared to show how much warping affects fitting. What is concerned about are the results of degree of polarization $pp$, the difference of the maximum intensity and the minimum intensity ($I_{\text{max}} - I_{\text{min}}$) and the phase angle $\alpha$.

4.4.4.1. Error Analysis in Regions of $I_{\text{max}} - I_{\text{min}}$. Degree of Polarization $pp$ and Phase angle $\alpha$.

First, the difference images $\Delta I_{\text{max}}$, $\Delta I_{\text{min}}$, $\Delta I_{\text{avg}}$, $\Delta pp$, $\Delta t$ and $\Delta \alpha$ are generated from the results of the two fittings. Then the errors of $pp$, $(I_{\text{max}} - I_{\text{min}})$ and $\alpha$ are calculated in the following 10 regions:
(1) Region 1: region of error where RMSE ≥ 20 in either warped or unwarped image of RMSE

(2) Region 2: region of high pp and high (Imax - Imin) of the unwarped images, where pp ≥ 35 and (Imax - Imin) ≥ 40

(3) Region 3: region of high pp and medium (Imax - Imin) of the unwarped images, where pp ≥ 35 and 15 ≤ (Imax - Imin) < 40

(4) Region 4: region of high pp and low (Imax - Imin) of the unwarped images, where pp > 35 and (Imax - Imin) < 15

(5) Region 5: region of medium pp and high (Imax - Imin) of the unwarped images, where 10 ≤ pp < 35 and (Imax - Imin) ≥ 40

(6) Region 6: region of medium pp and medium (Imax - Imin) of the unwarped images, where 10 ≤ pp < 35 and 15 ≤ (Imax - Imin) < 40

(7) Region 7: region of medium pp and low (Imax - Imin) of the unwarped images, where 10 ≤ pp < 35 and (Imax - Imin) < 15

(8) Region 8: region of low pp and high (Imax - Imin) of the unwarped images, where pp < 10 and (Imax - Imin) ≥ 40

(9) Region 9: region of low pp and medium (Imax - Imin) of the unwarped images, where pp < 10 and 15 ≤ (Imax - Imin) < 40

(10) Region 10: region of low pp and low (Imax - Imin) of the unwarped images, where pp < 10 and (Imax - Imin) < 15

From the fitting program, two times of phase angle 2α is calculated, with the range from 0° to 180°. The difference image of phase angles Δα is calculated as 0.5 × Δ2α, while Δ2α is calculated as the following:

\[ Δ2α = \min( \text{abs}(2α1 - 2α2), \text{abs}(2α1 - 2α2 + 180), \text{abs}(2α1 - 2α2 - 180) ) \]  

[4.1]

For example, the difference of 2° and 179° is 3° instead of 177°.

Table 4.2, 4.3 and 4.4 show the errors of pp, (Imax - Imin) and α. In the table, there are 8 items of compares including mean square error, unsigned maximum value, unsigned minimum value, signed average, unsigned average, average intensity of the unwarped image avg_1, average intensity of the warped image avg_2, and the number of image points in the region. ND means there is no data in that region.
Errors of pp:

<table>
<thead>
<tr>
<th>Region</th>
<th>RMS</th>
<th>Max</th>
<th>Min</th>
<th>Signed Average</th>
<th>Unsigned Average</th>
<th>Avg_1</th>
<th>Avg_2</th>
<th>Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>15.7480</td>
<td>79</td>
<td>0</td>
<td>-4</td>
<td>9</td>
<td>17</td>
<td>22</td>
<td>5613</td>
</tr>
<tr>
<td>2</td>
<td>10.7703</td>
<td>34</td>
<td>0</td>
<td>6</td>
<td>8</td>
<td>41</td>
<td>34</td>
<td>1820</td>
</tr>
<tr>
<td>3</td>
<td>4.6904</td>
<td>19</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>21</td>
<td>21</td>
<td>116</td>
</tr>
<tr>
<td>4</td>
<td>ND</td>
<td>ND</td>
<td>ND</td>
<td>ND</td>
<td>ND</td>
<td>ND</td>
<td>ND</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>12.5300</td>
<td>32</td>
<td>0</td>
<td>3</td>
<td>10</td>
<td>35</td>
<td>31</td>
<td>254</td>
</tr>
<tr>
<td>6</td>
<td>4.6904</td>
<td>42</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>18</td>
<td>18</td>
<td>29009</td>
</tr>
<tr>
<td>7</td>
<td>2.2361</td>
<td>8</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>8</td>
<td>9</td>
<td>139</td>
</tr>
<tr>
<td>8</td>
<td>ND</td>
<td>ND</td>
<td>ND</td>
<td>ND</td>
<td>ND</td>
<td>ND</td>
<td>ND</td>
<td>0</td>
</tr>
<tr>
<td>9</td>
<td>4.2426</td>
<td>45</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>12</td>
<td>11</td>
<td>17326</td>
</tr>
<tr>
<td>10</td>
<td>3.3166</td>
<td>48</td>
<td>0</td>
<td>-1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>191099</td>
</tr>
</tbody>
</table>

Table 4.2

<table>
<thead>
<tr>
<th>Region</th>
<th>RMS</th>
<th>Max</th>
<th>Min</th>
<th>Signed Average</th>
<th>Unsigned Average</th>
<th>Avg_1</th>
<th>Avg_2</th>
<th>Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>20.1246</td>
<td>104</td>
<td>0</td>
<td>-3</td>
<td>13</td>
<td>25</td>
<td>28</td>
<td>5613</td>
</tr>
<tr>
<td>2</td>
<td>16.8226</td>
<td>0</td>
<td>0</td>
<td>11</td>
<td>13</td>
<td>50</td>
<td>39</td>
<td>1820</td>
</tr>
<tr>
<td>3</td>
<td>8.6603</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>6</td>
<td>41</td>
<td>39</td>
<td>116</td>
</tr>
<tr>
<td>4</td>
<td>ND</td>
<td>ND</td>
<td>ND</td>
<td>ND</td>
<td>ND</td>
<td>ND</td>
<td>ND</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>15.8114</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>13</td>
<td>37</td>
<td>33</td>
<td>254</td>
</tr>
<tr>
<td>6</td>
<td>5.6569</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>3</td>
<td>20</td>
<td>19</td>
<td>29009</td>
</tr>
<tr>
<td>7</td>
<td>4.0000</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>16</td>
<td>15</td>
<td>139</td>
</tr>
<tr>
<td>8</td>
<td>ND</td>
<td>ND</td>
<td>ND</td>
<td>ND</td>
<td>ND</td>
<td>ND</td>
<td>ND</td>
<td>0</td>
</tr>
<tr>
<td>9</td>
<td>4.2426</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>11</td>
<td>10</td>
<td>17326</td>
</tr>
<tr>
<td>10</td>
<td>2.8284</td>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>1</td>
<td>2</td>
<td>4</td>
<td>191099</td>
</tr>
</tbody>
</table>

Table 4.3

49
<table>
<thead>
<tr>
<th>Region</th>
<th>RMS</th>
<th>Max</th>
<th>Min</th>
<th>Signed Average</th>
<th>Unsigned Average</th>
<th>Avg_1</th>
<th>Avg_2</th>
<th>Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Region 1</td>
<td>21.4942</td>
<td>45</td>
<td>0</td>
<td>18</td>
<td>18</td>
<td>38</td>
<td>38</td>
<td>5613</td>
</tr>
<tr>
<td>Region 2</td>
<td>8.0000</td>
<td>45</td>
<td>0</td>
<td>4</td>
<td>4</td>
<td>35</td>
<td>35</td>
<td>1820</td>
</tr>
<tr>
<td>Region 3</td>
<td>9.9499</td>
<td>27</td>
<td>0</td>
<td>8</td>
<td>8</td>
<td>38</td>
<td>34</td>
<td>116</td>
</tr>
<tr>
<td>Region 4</td>
<td>ND</td>
<td>ND</td>
<td>ND</td>
<td>ND</td>
<td>ND</td>
<td>ND</td>
<td>ND</td>
<td>0</td>
</tr>
<tr>
<td>Region 5</td>
<td>15.3623</td>
<td>45</td>
<td>0</td>
<td>9</td>
<td>9</td>
<td>34</td>
<td>38</td>
<td>254</td>
</tr>
<tr>
<td>Region 6</td>
<td>11.0905</td>
<td>45</td>
<td>0</td>
<td>8</td>
<td>8</td>
<td>36</td>
<td>30</td>
<td>29009</td>
</tr>
<tr>
<td>Region 7</td>
<td>6.7082</td>
<td>17</td>
<td>0</td>
<td>5</td>
<td>5</td>
<td>36</td>
<td>33</td>
<td>139</td>
</tr>
<tr>
<td>Region 8</td>
<td>ND</td>
<td>ND</td>
<td>ND</td>
<td>ND</td>
<td>ND</td>
<td>ND</td>
<td>ND</td>
<td>0</td>
</tr>
<tr>
<td>Region 9</td>
<td>14.7648</td>
<td>45</td>
<td>0</td>
<td>12</td>
<td>12</td>
<td>38</td>
<td>29</td>
<td>17326</td>
</tr>
<tr>
<td>Region 10</td>
<td>22.5832</td>
<td>45</td>
<td>0</td>
<td>18</td>
<td>18</td>
<td>36</td>
<td>26</td>
<td>191099</td>
</tr>
</tbody>
</table>

Table 4.4
From the above tables, we have the following analysis:

(1) The errors of region1 in all three tables are all very large. In this region, the RMSE image points of either the warped image or the unwarped image are greater than 20. Since the errors in this region mainly come from fitting, it shouldn't be considered as the errors of warping. Figure 4.35 shows the thresholded (T=20) image of image RMSE, the bright part of the image is actually the error region.

Because cameras at different positions different images of the same scene, when we warp an image taken by one camera to an image taken by another camera, there must be bands on the side that contains no data on the warped image. These regions is no use for our analysis, we are going to cut it in the next two experiments.

(2) We don't have any data in region4 and region8 because there is no such region that has high pp and low (I_{max} - I_{min}) or has low pp and high (I_{max} - I_{min}). There is also only very small number of image points in region3 where there is a high pp and medium (I_{max} - I_{max}), in region7 where there is a medium pp and low (I_{max} - I_{min}) and in region5 where there is medium pp and high (I_{max} - I_{min}). This result depends on the thresholds according to which the regions are classified.

(3) About 60% of image points in the images of pp, (I_{max} - I_{min}) are in region10 where there is low pp and low (I_{max} - I_{min}). The image points of the scene in this region is mostly not polarized, although the errors are not so small compared to the the average intensity, they are mostly random errors. The errors in this region shouldn't be counted very much because our concern is the surfaces that polarize.
From the above tables, we have the following analysis:

(1) The errors of region 1 in all three tables are all very large. In this region, the RMSE image points of either the warped image or the unwarped image are greater than 20. Since the errors in this region mainly come from fitting, it shouldn't be considered as the errors of warping. Figure 4.35 shows the thresholded (T=20) image of image RMSE, the bright part of the image is actually the error region.

Because cameras at different positions different images of the same scene. When we warp an image taken by one camera to an image taken by another camera, there must be bands on the side that contains no data on the warped image. These regions is no use for our analysis, we are going to cut it in the next two experiments.

![Figure 4.35](image_url)

(2) We don't have any data in region 4 and region 8 because there is no such region that has high pp and low (I_{max} - I_{min}) or has low pp and high (I_{max} - I_{min}). There is also only very small number of image points in region 3 where there is a high pp and medium (I_{max} - I_{max}), in region 7 where there is a medium pp and low (I_{max} - I_{min}) and in region 5 where there is medium pp and high (I_{max} - I_{max}). This result depends on the thresholds according to which the regions are classified.

(3) About 60% of image points in the images of pp. (I_{max} - I_{min}) are in region 10 where there is low pp and low (I_{max} - I_{min}). The image points of the scene in this region is mostly not polarized. although the errors are not so small compared to the the average intensity, they are mostly random errors. The errors in this region shouldn't be counted very much because our concern is the surfaces that polarize.
(4) The errors in region 2, region 6 and region 9 are very important. Compared to the average intensities, the errors in these regions are almost same and quite satisfactory. The signed average in region 6 and region 9 is 0, which shows the errors are spread evenly positive and negative. We also have very close values of average of the warped image and the unwarped image for the three regions. Because the regions are classified only by the unwarped images, the same averages tell us the regions overlap well, which shows the warping job is done pretty successfully.

(5) The errors of α are shown in table 4.4. In region 1 and region 10, errors are very large, the reason for that is same for errors of pp and (I_{max} - I_{min}). But the errors of region 2, region 6 and region 9 are not same, we need to do more research to figure out why.

4.4.4.2 Error analysis in regions of interest

Since specular surface of the objects are more interested and there are different shapes of shiny surfaces in the experiment, in this section, we compare the errors of those specular surfaces. We are also interested in errors of objects with different distances to the cameras. In this experiment, objects are close to the cameras, we expect there is be a focus problem with objects on different positions.

From the result image pp, we find eggs, cup, cubes, and some circles on the chart are shiny and quite polarized. Among those objects, eggs are very much curve shaped, cup is less curve shaped, cubes and the circles on the chart are not curve shaped. However, the cube is much more polarized than the other objects. Table 3.5, 3.6, 3.7, 3.8 show the errors of an egg, the cup, the four circles and a cube. Figure 4.36 shows our interests on the image pp.

![Figure 4.36](image-url)
(4) The errors in region2, region6 and region9 are very important. Compared to the average intensities, the errors in these regions are almost same and quite satisfactory. The signed average in region 6 and region9 is 0, which shows the errors are spreaded evenly positive and negative. We also have very close values of average of the warped image and the unwarped image for the three regions. Because the regions are classified only by the unwarped images, the same averages tell us the regions overlap well, which shows the warping job is done pretty successfully.

(5) The errors of $\alpha$ are shown in Table 4.4. In region1 and region10, errors are very large, the reason for that is same for errors of $pp$ and $(l_{max} - l_{min})$. But the errors of region2, region6 and region9 are not same, we need to do more research to figure out why.

4.4.4.2 Error analysis in regions of interest

Since specular surface of the objects are more interested and there are different shapes of shiny surfaces in the experiment, in this section, we compare the errors of those specular surfaces. We are also interested in errors of objects with different distances to the cameras. In this experiment, objects are close to the cameras, we expect there is be a focus problem with objects on different positions.

From the result image $pp$, we find eggs, cup, cubes, and some circles on the chart are shiny and quite polarized. Among those objects, eggs are very much curve shaped, cup is less curve shaped, cubes and the circles on the chart are not curve shaped. However, the cube is much more polarized than the other objects. Table 3.5, 3.6, 3.7, 3.8 show the errors of an egg, the cup, the four circles and a cube. Figure 4.36 shows our interests on the image $pp$. 

![Figure 4.36](image-url)
(1) Errors for an egg

<table>
<thead>
<tr>
<th>RMS</th>
<th>Max</th>
<th>Min</th>
<th>Signed Average</th>
<th>Unsigned Average</th>
<th>Avg_1</th>
<th>Avg_2</th>
<th>Number</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Error of pp</td>
<td>9.2195</td>
<td>47</td>
<td>0</td>
<td>-2</td>
<td>5</td>
<td>7</td>
<td>560</td>
</tr>
<tr>
<td>Error of Imax - Imin</td>
<td>17.0294</td>
<td>89</td>
<td>0</td>
<td>-1</td>
<td>8</td>
<td>12</td>
<td>560</td>
</tr>
<tr>
<td>Error of t</td>
<td>21.3073</td>
<td>44</td>
<td>0</td>
<td>17</td>
<td>17</td>
<td>92</td>
<td>560</td>
</tr>
</tbody>
</table>

Table 4.5

(2) Errors for the cup

<table>
<thead>
<tr>
<th>RMS</th>
<th>Max</th>
<th>Min</th>
<th>Signed Average</th>
<th>Unsigned Average</th>
<th>Avg_1</th>
<th>Avg_2</th>
<th>Number</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Error of pp</td>
<td>6.4807</td>
<td>43</td>
<td>0</td>
<td>-2</td>
<td>4</td>
<td>14</td>
<td>3196</td>
</tr>
<tr>
<td>Error of Imax - Imin</td>
<td>6.7082</td>
<td>39</td>
<td>0</td>
<td>-2</td>
<td>4</td>
<td>14</td>
<td>3196</td>
</tr>
<tr>
<td>Error of t</td>
<td>18.4120</td>
<td>45</td>
<td>0</td>
<td>14</td>
<td>14</td>
<td>65</td>
<td>3196</td>
</tr>
</tbody>
</table>

Table 4.6

(3) Errors for four circles on the chart

<table>
<thead>
<tr>
<th>RMS</th>
<th>Max</th>
<th>Min</th>
<th>Signed Average</th>
<th>Unsigned Average</th>
<th>Avg_1</th>
<th>Avg_2</th>
<th>Number</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Error of pp</td>
<td>3.0000</td>
<td>11</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>20</td>
<td>1815</td>
</tr>
<tr>
<td>Error of Imax - Imin</td>
<td>4.4721</td>
<td>20</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>25</td>
<td>1815</td>
</tr>
<tr>
<td>Error of t</td>
<td>9.3808</td>
<td>30</td>
<td>0</td>
<td>7</td>
<td>7</td>
<td>73</td>
<td>1815</td>
</tr>
</tbody>
</table>

Table 4.7

(4) Error for the cube

53
From the above tables, we can clearly see that for the errors of pp and (Imax - Imin), the more curve shaped the object is, the more errors occur on it. The reflective region of an object changes if the image is taken from different view point, so the more curve shaped the reflective region is, the more significant the errors are. That is why the egg has very big errors, the circles on the chart have much less errors. The cube doesn't have any curve shape, yet it has greater errors than the circles and the cup, the reason might be that the surface of the cube is polished by sand paper thus it is rougher than the surface of the cup and the circles, the polarization of image points on that surface is therefore more diverse. For errors of α, there is no such observation as for pp and (Imax - Imin), it shows that there errors of α are not affected by the shape of the object.

Now we pick up some objects with different distances from the cameras and study the errors happen on them. The objects are three eggs at different positions, which are marked in figure 4.37.

Table 4.9, 4.10 and 4.11 report the errors on the eggs of pp, (Imax - Imin) and α.

Table 4.8
From the above tables, we can clearly see that for the errors of pp and (Imax - Imin), the more curve shaped the object is, the more errors occur on it. The reflective region of an object changes if the image is taken from different view point, so the more curve shaped the reflective region is, the more significant the errors are. That is why the egg has very big errors, the circles on the chart have much less errors. The cube doesn't have any curve shape, yet it has greater errors than the circles and the cup. the reason might be that the surface of the cube is polished by sand paper thus it is rougher than the surface of the cup and the circles. the polarization of image points on that surface is therefore more diverse. For errors of $\alpha$, there is no such observation as for pp and (Imax - Imin), it shows that there errors of $\alpha$ are not affected by the shape of the object.

Now we pick up some objects with different distances from the cameras and study the errors happen on them. The objects are three eggs at different positions, which are marked in figure 4.37.

Table 4.8

<table>
<thead>
<tr>
<th>Error of $\alpha$</th>
<th>RMS</th>
<th>Max</th>
<th>Min</th>
<th>Signed Average</th>
<th>Unsigned Average</th>
<th>Avg_1</th>
<th>Avg_2</th>
<th>Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Error of pp</td>
<td>10.2470</td>
<td>50</td>
<td>0</td>
<td>0</td>
<td>6</td>
<td>20</td>
<td>21</td>
<td>3400</td>
</tr>
<tr>
<td>Error of $\text{Imax - Imin}$</td>
<td>13.4164</td>
<td>74</td>
<td>0</td>
<td>0</td>
<td>8</td>
<td>24</td>
<td>24</td>
<td>3400</td>
</tr>
<tr>
<td>Error of $\alpha$</td>
<td>17.3781</td>
<td>45</td>
<td>0</td>
<td>12</td>
<td>12</td>
<td>73</td>
<td>60</td>
<td>34008</td>
</tr>
</tbody>
</table>

Table 4.9, 4.10 and 4.11 report the errors on the eggs of pp, (Imax - Imin) and $\alpha$. 

Figure 4.37
(1) Error for egg #1 (the middle egg)

<table>
<thead>
<tr>
<th>Errors</th>
<th>RMS</th>
<th>Max</th>
<th>Min</th>
<th>Signed Average</th>
<th>Unsigned Average</th>
<th>Avg_1</th>
<th>Avg_2</th>
<th>Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>pp</td>
<td>6.9282</td>
<td>32</td>
<td>0</td>
<td>-2</td>
<td>0</td>
<td>11</td>
<td></td>
<td>364</td>
</tr>
<tr>
<td>Imax - Imin</td>
<td>9.0554</td>
<td>55</td>
<td>0</td>
<td>-1</td>
<td>5</td>
<td>12</td>
<td></td>
<td>364</td>
</tr>
<tr>
<td>t</td>
<td>23.4307</td>
<td>44</td>
<td>0</td>
<td>19</td>
<td>19</td>
<td>76</td>
<td></td>
<td>364</td>
</tr>
</tbody>
</table>

Table 4.9

(2) Error for egg #2 (the far egg)

<table>
<thead>
<tr>
<th>Errors</th>
<th>RMS</th>
<th>Max</th>
<th>Min</th>
<th>Signed Average</th>
<th>Unsigned Average</th>
<th>Avg_1</th>
<th>Avg_2</th>
<th>Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>pp</td>
<td>11.2250</td>
<td>47</td>
<td>0</td>
<td>-2</td>
<td>6</td>
<td>12</td>
<td></td>
<td>364</td>
</tr>
<tr>
<td>Imax - Imin</td>
<td>20.9284</td>
<td>89</td>
<td>0</td>
<td>-2</td>
<td>12</td>
<td>18</td>
<td></td>
<td>364</td>
</tr>
<tr>
<td>t</td>
<td>19.6469</td>
<td>44</td>
<td>0</td>
<td>16</td>
<td>16</td>
<td>88</td>
<td></td>
<td>364</td>
</tr>
</tbody>
</table>

Table 4.10

(3) Error for egg #3 (the close egg)

<table>
<thead>
<tr>
<th>Errors</th>
<th>RMS</th>
<th>Max</th>
<th>Min</th>
<th>Signed Average</th>
<th>Unsigned Average</th>
<th>Avg_1</th>
<th>Avg_2</th>
<th>Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>pp</td>
<td>19.0000</td>
<td>71</td>
<td>0</td>
<td>-5</td>
<td>12</td>
<td>18</td>
<td></td>
<td>364</td>
</tr>
<tr>
<td>Imax - Imin</td>
<td>26.8514</td>
<td>104</td>
<td>0</td>
<td>-3</td>
<td>16</td>
<td>19</td>
<td></td>
<td>364</td>
</tr>
<tr>
<td>t</td>
<td>22.0000</td>
<td>45</td>
<td>0</td>
<td>18</td>
<td>18</td>
<td>86</td>
<td></td>
<td>364</td>
</tr>
</tbody>
</table>

Table 4.11

From egg #1 to egg #3, the errors of pp and (Imax - Imin) get bigger significantly. Actually the cameras are focused at about the position of egg #1. That is why the errors are not so bad over
there but become rapidly larger as they are off focus. In the next experiment, we are going to put the cameras further and see if we can cancel the affect made by focus. However, the errors of $\alpha$ have nothing to do with focus.

4.4.4.3 Error analysis in regions of pp gradient

We have looked at the errors in regions of pp, now we want to study the errors of the gradient of pp. We take the gradient of pp through out the image (image of degree of polarization as a result of fitting), and look at the image of degree of polarization in figure 4.38.

We can see in the figure there is high gradient of pp on edges of the objects, where there is a big change of degree of polarization.

![Image](image_url)

**Figure 4.38**

There are a large number of image points with low gradient of pp, among those, some of image points have high pp. To look at the errors in greater detail, we calculate the errors in the following five regions:

1. Region 1: gradient of pp is less than 10, pp is greater than 35
2. Region 2: gradient of pp is less than 10, pp is between 10 and 35
3. Region 3: gradient of pp is less than 10, pp is less than 10
4. Region 4: gradient of pp is between 10 and 20
5. Region 5: gradient of pp is greater than 20

The gradient of an image point is calculated as:

$$\text{gradient} = \sqrt{(a-b)^2 + (c-d)^2}$$  \[4.1\]
there but become rapidly larger as the they are off focus. In the next experiment, we are going to put the cameras further and see if we can cancel the affect made by focus. However, the errors of $\alpha$ have nothing to do with focus.

4.4.4.3 Error analysis in regions of pp gradient

We have looked at the errors in regions of pp, now we want to study the errors of the gradient of pp. We take the gradient of pp through out the image(image of degree of polarization as a result of fitting), and look at the image of degree of polarization in figure 4.38.

We can see in the figure there is high gradient of pp on edges of the objects, where there is a big change of degree of polarization.

![Figure 4.38](image)

There are a large number of image points with low gradient of pp, among those, some of image points have high pp. To look at the errors in greater detail, we calculate the errors in the following five regions:

1. Region 1: gradient of pp is less than 10, pp is greater than 35
2. Region 2: gradient of pp is less than 10, pp is between 10 and 35
3. Region 3: gradient of pp is less than 10, pp is less than 10
4. Region 4: gradient of pp is between 10 and 20
5. Region 5: gradient of pp is greater than 20

The gradient of an image point is calculated as:

$$\text{gradient} = \sqrt{(a-b)^2 + (c-d)^2}$$  \[4.1\]
where a, b are the left and right neighbors of the image point, c, d are its up and down neighbors.

Table 4.12 shows the errors in the five regions.

### error for pp

<table>
<thead>
<tr>
<th>Region</th>
<th>RMS</th>
<th>Max</th>
<th>Min</th>
<th>Signed Average</th>
<th>Unsigned Average</th>
<th>Avg_1</th>
<th>Avg_2</th>
<th>Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Region 1</td>
<td>1.12694</td>
<td>36</td>
<td>0</td>
<td>6</td>
<td>8</td>
<td>41</td>
<td>34</td>
<td>2307</td>
</tr>
<tr>
<td>Region 2</td>
<td>.44721</td>
<td>47</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>16</td>
<td>15</td>
<td>45771</td>
</tr>
<tr>
<td>Region 3</td>
<td>.36056</td>
<td>71</td>
<td>0</td>
<td>-1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>186843</td>
</tr>
<tr>
<td>Region 4</td>
<td>.95917</td>
<td>68</td>
<td>0</td>
<td>0</td>
<td>6</td>
<td>17</td>
<td>17</td>
<td>4964</td>
</tr>
<tr>
<td>Region 5</td>
<td>1.62788</td>
<td>68</td>
<td>0</td>
<td>-2</td>
<td>12</td>
<td>21</td>
<td>23</td>
<td>805</td>
</tr>
</tbody>
</table>

### error for Imax - Imin:

<table>
<thead>
<tr>
<th>Region</th>
<th>RMS</th>
<th>Max</th>
<th>Min</th>
<th>Signed Average</th>
<th>Unsigned Average</th>
<th>Avg_1</th>
<th>Avg_2</th>
<th>Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Region 1</td>
<td>1.87883</td>
<td>74</td>
<td>0</td>
<td>11</td>
<td>14</td>
<td>52</td>
<td>40</td>
<td>2307</td>
</tr>
<tr>
<td>Region 2</td>
<td>.54772</td>
<td>78</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>17</td>
<td>16</td>
<td>45771</td>
</tr>
<tr>
<td>Region 3</td>
<td>.31623</td>
<td>101</td>
<td>0</td>
<td>-1</td>
<td>1</td>
<td>2</td>
<td>4</td>
<td>186843</td>
</tr>
<tr>
<td>Region 4</td>
<td>1.29228</td>
<td>102</td>
<td>0</td>
<td>0</td>
<td>7</td>
<td>20</td>
<td>19</td>
<td>4964</td>
</tr>
<tr>
<td>Region 5</td>
<td>2.38747</td>
<td>104</td>
<td>0</td>
<td>-3</td>
<td>17</td>
<td>23</td>
<td>27</td>
<td>805</td>
</tr>
</tbody>
</table>

### error of t:

<table>
<thead>
<tr>
<th>Region</th>
<th>RMS</th>
<th>Max</th>
<th>Min</th>
<th>Signed Average</th>
<th>Unsigned Average</th>
<th>Avg_1</th>
<th>Avg_2</th>
<th>Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Region 1</td>
<td>1.04403</td>
<td>45</td>
<td>0</td>
<td>6</td>
<td>6</td>
<td>36</td>
<td>36</td>
<td>2307</td>
</tr>
<tr>
<td>Region 2</td>
<td>1.29228</td>
<td>45</td>
<td>0</td>
<td>10</td>
<td>10</td>
<td>37</td>
<td>37</td>
<td>45771</td>
</tr>
<tr>
<td>Region 3</td>
<td>2.26053</td>
<td>45</td>
<td>0</td>
<td>18</td>
<td>18</td>
<td>36</td>
<td>36</td>
<td>186843</td>
</tr>
<tr>
<td>Region 4</td>
<td>1.64621</td>
<td>45</td>
<td>0</td>
<td>13</td>
<td>13</td>
<td>38</td>
<td>38</td>
<td>4964</td>
</tr>
<tr>
<td>Region 5</td>
<td>1.85742</td>
<td>45</td>
<td>0</td>
<td>15</td>
<td>15</td>
<td>40</td>
<td>40</td>
<td>805</td>
</tr>
</tbody>
</table>

Table 4.12
We get from the table of errors for pp and (I_{max} - I_{min}), all the errors compared to the average in that region are durable. Still we notice that region where there is a high pp gradient has greater errors than region, region2 and region4. Region contains mainly edges of the objects, we expect there are more errors in the region because of warping. In region3, there are large number of image points that are mostly not polarized, but due to the random noise and radiometric problem, the errors in that region are large compared to the small average. We expect the errors in this are smaller after intensity linearization. Also we don't find much clue with the errors of $\alpha$.

4.4.4.4 Errors of phase angle $\alpha$

So far we haven't found any clue about the errors of $\alpha$, we haven't found in what area $\alpha$ has large errors. In this section, we take a look at the errors of $\alpha$.

First, we calculate the image of difference between the image of $\alpha$ from unwarped images and the image of $\alpha$ from warped images. Figure 4.39 shows a normalized image $\Delta \alpha$.

From the image, we find there are very small errors on the calibration chart except on some edges of the circles, also there are very small errors on the shiny surfaces of the eggs, the cup and the cubes. These surfaces are very smooth and thus the image points in the regions tend to have same phase angles (please refer figure 2.1 for phase angle). On the contrary, for the rough surfaces or other places where phase angles are very diverse, we have large errors of $\alpha$. This phenomenon is also consistent with the data we get in 4.4.4.1.
We get from the table of errors for pp and (Imax - Imin), all the errors compared to the average in that region are durable. Still we notice that region where there is a high pp gradient has greater errors than region, region2 and region4. Region contains mainly edges of the objects, we expect there are more errors in the region because of warping. In region3, there are large number of image points that are mostly not polarized, but due to the random noise and radiometric problem, the errors in that region are large compared to the small average. We expect the errors in this are smaller after intensity linearization. Also we don't find much clue with the errors of α.

4.4.4.4 Errors of phase angle α

So far we haven't found any clue about the errors of α, we haven't found in what area α has large errors. In this section, we take a look at the errors of α.

First, we calculate the image of difference between the image of α from unwarped images and the image of α from warped images. Figure 4.39 shows a normalized image Δα.

![Figure 4.39](image.png)

From the image, we find there are very small errors on the calibration chart except on some edges of the circles, also there are very small errors on the shiny surfaces of the eggs, the cup and the cubes. These surfaces are very smooth and thus the image points in the regions tend to have same phase angles (please refer figure 2.1 for phase angle). On the contrary, for the rough surfaces or other places where phase angles are very diverse, we have large errors of α. This phenomenon is also consistent with the data we get in 4.4.4.1.
4.5 Experiment Two

In this experiment, we do the same things as in experiment one except two things: one, we use a larger calibration chart; two, the cameras are about three times farther to the objects. We also take images using four cameras which are bundled together at angles of 0, 30, 90, 140, 170, 220, 260 and 310. We then warp 30 and 220 images taken by cam2 to cam1, warp 90 and 260 images taken by cam3 to cam1, and warp 140 and 310 images taken by cam4 to cam1 also. Then we put 0 and 170 original images taken by cam1 and 6 warped images to the fitting program and compare the results with putting images taken only by cam1 at 8 angles.

4.5.1 Experiment Setup and Image Taking

The setup of the experiment are mostly the same with experiment one. We put a nine times bigger calibration chart with 54 black circles in the scene. We take images with every camera at the angles of 0, 30, 90, 140, 170, 220, 260 and 310. Also to reduce noise, 16 frames are averaged for one image. Here we show the image at 0 by cam1, the image at 30 by cam2, the image at 90 by cam3 and the image at 140 by cam4 in figure 4.40; and we show the image at 0, 30, 90 and 140 all by cam1 in figure 4.41. Please compare the two groups of images.

![Image of calibration chart and camera angles](image_url)

Figure 4.40
4.5 Experiment Two

In this experiment, we do the same things as in experiment one except two things: one, we use a larger calibration chart; two, the cameras are about three times farther to the objects. We also take images using four cameras which are bundled together at angles of 0, 30, 90, 140, 170, 220, 260 and 310. We then warp 30 and 220 images taken by cam2 to cam1, warp 90 and 260 images taken by cam3 to cam1, and warp 140 and 310 images taken by cam4 to cam1 also. Then we put 0 and 170 original images taken by cam1 and 6 warped images to the fitting program and compare the results with putting images taken only by cam1 at 8 angles.

4.5.1 Experiment Setup and Image Taking

The setup of the experiment are mostly the same with experiment one. We put a nine times bigger calibration chart with 54 black circles in the scene. We take images with every camera at the angles of 0, 30, 90, 140, 170, 220, 260 and 310. Also to reduce noise, 16 frames are averaged for one image. Here we show the image at 0 by cam1, the image at 30 by cam2, the image at 90 by cam3 and the image at 140 by cam4 in figure 4.40; and we show the image at 0, 30, 90 and 140 all by cam1 in figure 4.41. Please compare the two groups of images.
4.5.2 Image Warping

We do the same warping as in experiment one. Figure 4.42 shows the original image at 0 by cam1, the warped image at 30 from cam2 to cam1, the warped image at 90 from cam3 to cam1, and the warped image at 140 from cam4 to cam1.
4.5.2 Image Warping

We do the same warping as in experiment one. Figure 4.42 shows the original image at 0 by cam1, the warped image at 30 from cam2 to cam1, the warped image at 90 from cam3 to cam1, and the warped image at 140 from cam4 to cam1.
After warping, we check the warping errors same way as in experiment one. We take the difference of the source image and the target image, and calculate RMS error, signed maximum, signed minimum, unsigned mean and signed mean of the difference image. Please check table 4.12 for errors of warping.

<table>
<thead>
<tr>
<th>Warping Cases</th>
<th>RMS</th>
<th>Max</th>
<th>Min</th>
<th>Unsigned Mean</th>
<th>Signed Mean</th>
</tr>
</thead>
<tbody>
<tr>
<td>cam2 to cam1 at 30</td>
<td>4.98</td>
<td>69</td>
<td>-104</td>
<td>3.15</td>
<td>-1.52</td>
</tr>
<tr>
<td>cam2 to cam1 at 220</td>
<td>5.59</td>
<td>98</td>
<td>-122</td>
<td>3.63</td>
<td>2.40</td>
</tr>
<tr>
<td>cam3 to cam1 at 90</td>
<td>7.26</td>
<td>86</td>
<td>-120</td>
<td>5.30</td>
<td>4.50</td>
</tr>
<tr>
<td>cam3 to cam1 at 260</td>
<td>6.74</td>
<td>87</td>
<td>-142</td>
<td>4.00</td>
<td>3.00</td>
</tr>
<tr>
<td>cam4 to cam1 at 140</td>
<td>5.26</td>
<td>52</td>
<td>-59</td>
<td>3.71</td>
<td>2.72</td>
</tr>
<tr>
<td>cam4 to cam1 at 310</td>
<td>5.86</td>
<td>60</td>
<td>-47</td>
<td>4.17</td>
<td>3.21</td>
</tr>
</tbody>
</table>

Table 4.13
After warping, we check the warping errors same way as in experiment one. We take the difference of the source image and the target image, and calculate RMS error, signed maximum, signed minimum, unsigned mean and signed mean of the difference image. Please check table 4.12 for errors of warping.

<table>
<thead>
<tr>
<th>Warping Cases</th>
<th>RMS</th>
<th>Max</th>
<th>Min</th>
<th>Unsigned Mean</th>
<th>Signed Mean</th>
</tr>
</thead>
<tbody>
<tr>
<td>cam2 to cam1 at 30</td>
<td>4.98</td>
<td>69</td>
<td>-104</td>
<td>3.15</td>
<td>-1.52</td>
</tr>
<tr>
<td>cam2 to cam1 at 220</td>
<td>5.59</td>
<td>98</td>
<td>-122</td>
<td>3.63</td>
<td>2.40</td>
</tr>
<tr>
<td>cam3 to cam1 at 90</td>
<td>7.26</td>
<td>86</td>
<td>-120</td>
<td>5.30</td>
<td>4.50</td>
</tr>
<tr>
<td>cam3 to cam1 at 260</td>
<td>6.74</td>
<td>87</td>
<td>-142</td>
<td>4.00</td>
<td>3.00</td>
</tr>
<tr>
<td>cam4 to cam1 at 140</td>
<td>5.26</td>
<td>52</td>
<td>-59</td>
<td>3.71</td>
<td>2.72</td>
</tr>
<tr>
<td>cam4 to cam1 at 310</td>
<td>5.86</td>
<td>60</td>
<td>-47</td>
<td>4.17</td>
<td>3.21</td>
</tr>
</tbody>
</table>

Table 4.13
Errors are durable compared to the average intensity of about 70 to 80 of the images. Also we have some large maximum and minimum values, the explanation to it is same as we explain the same phenomenon in experiment one.

4.5.3 Fitting of polarization parameters

In order to get the polarization parameters, we have to do two groups of fittings, the 8 input images of each one are as the following:

Group one:
1. image at 0 by cam1
2. image at 30 by cam1
3. image at 90 by cam1
4. image at 140 by cam1
5. image at 170 by cam1
6. image at 220 by cam1
7. image at 260 by cam1
8. image at 310 by cam1

Group two:
1. image at 0 by cam1
2. image at 30 by cam2 and warped to cam1
3. image at 90 by cam3 and warped to cam1
4. image at 140 by cam4 and warped to cam1
5. image at 170 by cam1
6. image at 220 by cam2 and warped to cam1
7. image at 260 by cam3 and warped to cam1
8. image at 310 by cam4 and warped to cam1

We get 6 output images from each fitting, they are images of maximum intensities $I_{\text{max}}$, minimum intensities $I_{\text{min}}$, average intensities $I_{\text{avg}}$, phase angle $\alpha$, degree of polarization $p_p$ and RMSE $e$. The result image of each fitting is shown in figure 4.43 and 4.44. Among them, the images of RMSE (every pixel in the image has been multiplied by 100) are normalized, otherwise we would see nothing.
Figure 4.43
Figure 4.44
Figure 4.44
4.5.4 Error Analysis

In this section, we also compare the results from the two fittings in different regions and see how much warping affects fitting. We basically do the same analysis as we do in experiment one.

4.5.4.1. Error analysis in regions of \( I_{\text{max}} - I_{\text{min}} \), degree of polarization \( \Delta p \) and phase angle \( \alpha \).

First, the difference images \( \Delta I_{\text{max}}, \Delta I_{\text{min}}, \Delta I_{\text{avg}}, \Delta p, \Delta t \) and \( \Delta \epsilon \) are generated from the results of the two fittings, then the errors of \( \Delta p, (I_{\text{max}} - I_{\text{min}}) \) and \( \alpha \) are calculated in the same 10 regions with experiment one.

Table 4.13, 4.14 and 4.15 show the errors of \( \Delta p, (I_{\text{max}} - I_{\text{min}}) \) and \( \alpha \).

Errors for \( \Delta p \):

<table>
<thead>
<tr>
<th>Region</th>
<th>RMS</th>
<th>Max</th>
<th>Min</th>
<th>Signed Average</th>
<th>Unsigned Average</th>
<th>Avg_1</th>
<th>Avg_2</th>
<th>Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>11.5758</td>
<td>64</td>
<td>0</td>
<td>-5</td>
<td>8</td>
<td>15</td>
<td>21</td>
<td>9531</td>
</tr>
<tr>
<td>2</td>
<td>8.7178</td>
<td>31</td>
<td>0</td>
<td>4</td>
<td>6</td>
<td>39</td>
<td>34</td>
<td>310</td>
</tr>
<tr>
<td>3</td>
<td>2.4495</td>
<td>32</td>
<td>0</td>
<td>-1</td>
<td>2</td>
<td>18</td>
<td>20</td>
<td>6641</td>
</tr>
<tr>
<td>4</td>
<td>ND</td>
<td>ND</td>
<td>ND</td>
<td>ND</td>
<td>ND</td>
<td>ND</td>
<td>ND</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>8.1240</td>
<td>27</td>
<td>0</td>
<td>2</td>
<td>5</td>
<td>36</td>
<td>34</td>
<td>122</td>
</tr>
<tr>
<td>6</td>
<td>4.7958</td>
<td>35</td>
<td>0</td>
<td>-1</td>
<td>3</td>
<td>17</td>
<td>18</td>
<td>27268</td>
</tr>
<tr>
<td>7</td>
<td>2.8284</td>
<td>23</td>
<td>0</td>
<td>-2</td>
<td>2</td>
<td>7</td>
<td>10</td>
<td>21008</td>
</tr>
<tr>
<td>8</td>
<td>ND</td>
<td>ND</td>
<td>ND</td>
<td>ND</td>
<td>ND</td>
<td>ND</td>
<td>ND</td>
<td>0</td>
</tr>
<tr>
<td>9</td>
<td>4.2426</td>
<td>33</td>
<td>0</td>
<td>-2</td>
<td>3</td>
<td>13</td>
<td>15</td>
<td>27824</td>
</tr>
<tr>
<td>10</td>
<td>4.8990</td>
<td>40</td>
<td>0</td>
<td>-3</td>
<td>3</td>
<td>4</td>
<td>8</td>
<td>166496</td>
</tr>
</tbody>
</table>

Table 4.14
Errors for \((I_{\text{max}} - I_{\text{min}})\):

<table>
<thead>
<tr>
<th>Region</th>
<th>RMS</th>
<th>Max</th>
<th>Min</th>
<th>Signed Average</th>
<th>Unsigned Average</th>
<th>Avg_1</th>
<th>Avg_2</th>
<th>Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>18.1934</td>
<td>132</td>
<td>0</td>
<td>-8</td>
<td>13</td>
<td>24</td>
<td>32</td>
<td>9531</td>
</tr>
<tr>
<td>2</td>
<td>13.0000</td>
<td>49</td>
<td>0</td>
<td>6</td>
<td>9</td>
<td>46</td>
<td>40</td>
<td>310</td>
</tr>
<tr>
<td>3</td>
<td>8.1240</td>
<td>53</td>
<td>0</td>
<td>-7</td>
<td>7</td>
<td>48</td>
<td>55</td>
<td>6641</td>
</tr>
<tr>
<td>4</td>
<td>ND</td>
<td>ND</td>
<td>ND</td>
<td>ND</td>
<td>ND</td>
<td>ND</td>
<td>ND</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>10.0995</td>
<td>30</td>
<td>0</td>
<td>0</td>
<td>7</td>
<td>35</td>
<td>36</td>
<td>122</td>
</tr>
<tr>
<td>6</td>
<td>7.2111</td>
<td>53</td>
<td>0</td>
<td>-3</td>
<td>5</td>
<td>20</td>
<td>24</td>
<td>27268</td>
</tr>
<tr>
<td>7</td>
<td>7.1414</td>
<td>47</td>
<td>0</td>
<td>-6</td>
<td>6</td>
<td>16</td>
<td>23</td>
<td>21008</td>
</tr>
<tr>
<td>8</td>
<td>ND</td>
<td>ND</td>
<td>ND</td>
<td>ND</td>
<td>ND</td>
<td>ND</td>
<td>ND</td>
<td>0</td>
</tr>
<tr>
<td>9</td>
<td>4.3589</td>
<td>46</td>
<td>0</td>
<td>-2</td>
<td>3</td>
<td>10</td>
<td>13</td>
<td>27824</td>
</tr>
<tr>
<td>10</td>
<td>6.0828</td>
<td>56</td>
<td>0</td>
<td>-4</td>
<td>4</td>
<td>4</td>
<td>9</td>
<td>166496</td>
</tr>
</tbody>
</table>

Figure 4.15

Errors for \(\alpha\):

<table>
<thead>
<tr>
<th>Region</th>
<th>RMS</th>
<th>Max</th>
<th>Min</th>
<th>Signed Average</th>
<th>Unsigned Average</th>
<th>Avg_1</th>
<th>Avg_2</th>
<th>Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>18.4932</td>
<td>45</td>
<td>0</td>
<td>14</td>
<td>14</td>
<td>24</td>
<td>27</td>
<td>9531</td>
</tr>
<tr>
<td>2</td>
<td>10.6771</td>
<td>37</td>
<td>0</td>
<td>7</td>
<td>7</td>
<td>27</td>
<td>24</td>
<td>310</td>
</tr>
<tr>
<td>3</td>
<td>6.4031</td>
<td>44</td>
<td>0</td>
<td>6</td>
<td>6</td>
<td>25</td>
<td>19</td>
<td>6641</td>
</tr>
<tr>
<td>4</td>
<td>ND</td>
<td>ND</td>
<td>ND</td>
<td>ND</td>
<td>ND</td>
<td>ND</td>
<td>ND</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>11.5326</td>
<td>43</td>
<td>0</td>
<td>8</td>
<td>8</td>
<td>29</td>
<td>23</td>
<td>122</td>
</tr>
<tr>
<td>6</td>
<td>10.2956</td>
<td>45</td>
<td>0</td>
<td>8</td>
<td>8</td>
<td>23</td>
<td>17</td>
<td>27268</td>
</tr>
<tr>
<td>7</td>
<td>12.7279</td>
<td>40</td>
<td>0</td>
<td>12</td>
<td>12</td>
<td>23</td>
<td>11</td>
<td>21008</td>
</tr>
<tr>
<td>8</td>
<td>ND</td>
<td>ND</td>
<td>ND</td>
<td>ND</td>
<td>ND</td>
<td>ND</td>
<td>ND</td>
<td>0</td>
</tr>
<tr>
<td>9</td>
<td>10.7238</td>
<td>45</td>
<td>0</td>
<td>8</td>
<td>8</td>
<td>19</td>
<td>13</td>
<td>27824</td>
</tr>
<tr>
<td>10</td>
<td>14.4568</td>
<td>45</td>
<td>0</td>
<td>11</td>
<td>11</td>
<td>18</td>
<td>12</td>
<td>166496</td>
</tr>
</tbody>
</table>

Table 4.16
From the above tables and compare it to the errors of 10 regions in experiment one, we have the following analysis:

(1) In the error region, situation is same in here as in experiment one.

(2) We still don't find any region with high pp and low (I_{max}-I_{min}) or with low pp and high (I_{max}-I_{min}). We also have similar situation in region10.

(3) In region3, region6, region7 and region9 where the most polarization goes on, we have a little smaller errors than in region2, region6 and region9 for experiment one. The reason could be that we use a much bigger calibration chart this time, we can find the centroids more precisely and thus do a better warping job.

(4) There are still large errors in region1 and region10. In regions of polarization, there are bigger errors of $\alpha$ than in experiment one. The explanation is that the objects are relatively small this time, there is bigger area of smooth flat chart surfaces that would make same $\alpha$ for every scene point in a region.
4.5.4.2 Errors for regions of interest

We are still interested in errors on the cube, the cup, the eggs and the circles on the chart. The interested objects are shown in figure 4.45.

![Figure 4.45](image)

The following tables show the errors of the objects respectively.

(1) Errors on the cube

<table>
<thead>
<tr>
<th></th>
<th>RMS</th>
<th>Max</th>
<th>Min</th>
<th>Signed Average</th>
<th>Unsigned Average</th>
<th>Avg_1</th>
<th>Avg_2</th>
<th>Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Error of pp</td>
<td>8.7178</td>
<td>50</td>
<td>0</td>
<td>-3</td>
<td>5</td>
<td>9</td>
<td>13</td>
<td>2726</td>
</tr>
<tr>
<td>Error of Imax - Imin</td>
<td>10.7238</td>
<td>76</td>
<td>0</td>
<td>-3</td>
<td>6</td>
<td>9</td>
<td>13</td>
<td>2726</td>
</tr>
<tr>
<td>Error of t</td>
<td>13.6015</td>
<td>45</td>
<td>0</td>
<td>9</td>
<td>9</td>
<td>31</td>
<td>25</td>
<td>2726</td>
</tr>
</tbody>
</table>

Table 4.17
4.5.4.2 Errors for regions of interest

We are still interested in errors on the cube, the cup, the eggs and the circles on the chart. The interested objects are shown in figure 4.45.

![Figure 4.45](image_url)

The following tables show the errors of the objects respectively.

(1) Errors on the cube

<table>
<thead>
<tr>
<th></th>
<th>RMS</th>
<th>Max</th>
<th>Min</th>
<th>Signed Average</th>
<th>Unsigned Average</th>
<th>Avg_1</th>
<th>Avg_2</th>
<th>Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Error of pp</td>
<td>8.7178</td>
<td>50</td>
<td>0</td>
<td>-3</td>
<td>5</td>
<td>9</td>
<td>13</td>
<td>2726</td>
</tr>
<tr>
<td>Error of I_{max} - I_{min}</td>
<td>10.7238</td>
<td>76</td>
<td>0</td>
<td>-3</td>
<td>6</td>
<td>9</td>
<td>13</td>
<td>2726</td>
</tr>
<tr>
<td>Error of t</td>
<td>13.6015</td>
<td>45</td>
<td>0</td>
<td>9</td>
<td>9</td>
<td>31</td>
<td>25</td>
<td>2726</td>
</tr>
</tbody>
</table>

Table 4.17
(2) Errors on the cup

<table>
<thead>
<tr>
<th></th>
<th>RMS</th>
<th>Max</th>
<th>Min</th>
<th>Signed Average</th>
<th>Unsigned Average</th>
<th>Avg_1</th>
<th>Avg_2</th>
<th>Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Error of pp</td>
<td>8.4261</td>
<td>62</td>
<td>0</td>
<td>-4</td>
<td>5</td>
<td>6</td>
<td>11</td>
<td>4260</td>
</tr>
<tr>
<td>Error of Imax - Imin</td>
<td>12.4499</td>
<td>132</td>
<td>0</td>
<td>-4</td>
<td>5</td>
<td>8</td>
<td>13</td>
<td>4260</td>
</tr>
<tr>
<td>Error of t</td>
<td>20.1990</td>
<td>45</td>
<td>0</td>
<td>16</td>
<td>16</td>
<td>49</td>
<td>21</td>
<td>4260</td>
</tr>
</tbody>
</table>

Table 4.18

(3) Errors on the circles

<table>
<thead>
<tr>
<th></th>
<th>RMS</th>
<th>Max</th>
<th>Min</th>
<th>Signed Average</th>
<th>Unsigned Average</th>
<th>Avg_1</th>
<th>Avg_2</th>
<th>Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Error of pp</td>
<td>4.7958</td>
<td>29</td>
<td>0</td>
<td>-3</td>
<td>3</td>
<td>14</td>
<td>17</td>
<td>6834</td>
</tr>
<tr>
<td>Error of Imax - Imin</td>
<td>9.2195</td>
<td>45</td>
<td>0</td>
<td>-6</td>
<td>7</td>
<td>18</td>
<td>24</td>
<td>6834</td>
</tr>
<tr>
<td>Error of t</td>
<td>11.1355</td>
<td>45</td>
<td>0</td>
<td>9</td>
<td>9</td>
<td>39</td>
<td>23</td>
<td>6834</td>
</tr>
</tbody>
</table>

Table 4.19

(4) Errors on egg #1 (the middle egg)

<table>
<thead>
<tr>
<th></th>
<th>RMS</th>
<th>Max</th>
<th>Min</th>
<th>Signed Average</th>
<th>Unsigned Average</th>
<th>Avg_1</th>
<th>Avg_2</th>
<th>Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Error of pp</td>
<td>10.0499</td>
<td>41</td>
<td>0</td>
<td>-5</td>
<td>6</td>
<td>5</td>
<td>11</td>
<td>609</td>
</tr>
<tr>
<td>Error of Imax - Imin</td>
<td>12.4900</td>
<td>67</td>
<td>0</td>
<td>-5</td>
<td>7</td>
<td>5</td>
<td>11</td>
<td>609</td>
</tr>
<tr>
<td>Error of t</td>
<td>17.0294</td>
<td>44</td>
<td>0</td>
<td>12</td>
<td>12</td>
<td>36</td>
<td>28</td>
<td>609</td>
</tr>
</tbody>
</table>

Table 4.20
(5) Error on egg #2 (the far egg)

<table>
<thead>
<tr>
<th></th>
<th>RMS</th>
<th>Max</th>
<th>Min</th>
<th>Signed Average</th>
<th>Unsigned Average</th>
<th>Avg_1</th>
<th>Avg_2</th>
<th>Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Error of pp</td>
<td>9.3808</td>
<td>46</td>
<td>0</td>
<td>-5</td>
<td>6</td>
<td>5</td>
<td>11</td>
<td>704</td>
</tr>
<tr>
<td>Error of Imax - Imin</td>
<td>10.9545</td>
<td>78</td>
<td>0</td>
<td>-5</td>
<td>6</td>
<td>5</td>
<td>10</td>
<td>704</td>
</tr>
<tr>
<td>Error of t</td>
<td>18.0555</td>
<td>45</td>
<td>0</td>
<td>13</td>
<td>13</td>
<td>39</td>
<td>26</td>
<td>704</td>
</tr>
</tbody>
</table>

Table 4.21

(6) Errors on egg #3 (the close egg)

<table>
<thead>
<tr>
<th></th>
<th>RMS</th>
<th>Max</th>
<th>Min</th>
<th>Signed Average</th>
<th>Unsigned Average</th>
<th>Avg_1</th>
<th>Avg_2</th>
<th>Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Error of pp</td>
<td>7.8740</td>
<td>34</td>
<td>0</td>
<td>-5</td>
<td>5</td>
<td>4</td>
<td>9</td>
<td>814</td>
</tr>
<tr>
<td>Error of Imax - Imin</td>
<td>11.3137</td>
<td>65</td>
<td>0</td>
<td>-5</td>
<td>7</td>
<td>5</td>
<td>11</td>
<td>814</td>
</tr>
<tr>
<td>Error of t</td>
<td>19.8997</td>
<td>45</td>
<td>0</td>
<td>15</td>
<td>15</td>
<td>37</td>
<td>30</td>
<td>814</td>
</tr>
</tbody>
</table>

Table 4.22

All the errors listed above are very large compared to the errors on the same objects in experiment one. As the cameras get further from the objects, the objects become smaller as well. Let’s consider an extreme situation, if the object is as small as a few pixels in an image, there is a big chance that the warped image of the object wouldn’t overlap with the original image at all. So small objects get more errors through warping. We can also compare the errors of the above objects among themselves. The circles on the chart are much larger than the other objects, so the errors on it are smaller. The eggs have the largest errors because they are very small.

Checking on the eggs, we find although they are arranged almost the same way as in experiment one, the errors on them don’t change with the distance from the eggs to the cameras. This is what we expect that the problem of focus can be got rid of if the cameras are farther enough to the objects. However since we use 6mm cameras, we couldn’t eliminate the focus errors and at the same time keep the objects big enough to get moderate errors. I am sure if we use some
cameras with larger zoom, we are able to get satisfactory results.

4.5.4.3 Error analysis in regions of pp gradient

In this section, we also take the gradient of pp through out the image and look at the errors of gradient pp and pp in five regions as the following:

(1) Region 1: gradient of pp is less than 10, pp is greater than 35
(2) Region 2: gradient of pp is less than 10, pp is between 10 and 35
(3) Region 3: gradient of pp is less than 10, pp is less than 10
(4) Region 4: gradient of pp is between 10 and 20
(4) Region 5: gradient of pp is greater than 20

The gradient of pp is calculated the same ways as in experiment one. The normalized image of pp gradient is shown in figure 4.46.

![Figure 4.46](image)

Table 4.22 shows the errors in the five regions.
cameras with larger zoom, we are able to get satisfactory results.

4.5.4.3 Error analysis in regions of pp gradient

In this section, we also take the gradient of pp through out the image and look at the errors of gradient pp and pp in five regions as the following:

1. Region 1: gradient of pp is less than 10, pp is greater than 35
2. Region 2: gradient of pp is less than 10, pp is between 10 and 35
3. Region 3: gradient of pp is less than 10, pp is less than 10
4. Region 4: gradient of pp is between 10 and 20
5. Region 5: gradient of pp is greater than 20

The gradient of pp is calculated the same ways as in experiment one. The normalized image of pp gradient is shown in figure 4.46.

Figure 4.46

Table 4.22 shows the errors in the five regions.
error of pp:

<table>
<thead>
<tr>
<th></th>
<th>RMS</th>
<th>Max</th>
<th>Min</th>
<th>Signed Average</th>
<th>Unsigned Average</th>
<th>Avg_1</th>
<th>Avg_2</th>
<th>Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Region 1</td>
<td>8.3666</td>
<td>31</td>
<td>0</td>
<td>3</td>
<td>6</td>
<td>39</td>
<td>36</td>
<td>488</td>
</tr>
<tr>
<td>Region 2</td>
<td>4.6904</td>
<td>46</td>
<td>0</td>
<td>-2</td>
<td>3</td>
<td>15</td>
<td>17</td>
<td>62455</td>
</tr>
<tr>
<td>Region 3</td>
<td>4.8990</td>
<td>50</td>
<td>0</td>
<td>-3</td>
<td>3</td>
<td>4</td>
<td>8</td>
<td>183822</td>
</tr>
<tr>
<td>Region 4</td>
<td>8.7750</td>
<td>64</td>
<td>0</td>
<td>-3</td>
<td>6</td>
<td>15</td>
<td>18</td>
<td>6418</td>
</tr>
<tr>
<td>Region 5</td>
<td>17.6918</td>
<td>62</td>
<td>0</td>
<td>-8</td>
<td>12</td>
<td>24</td>
<td>33</td>
<td>713</td>
</tr>
</tbody>
</table>

Table 4.23

error of Imax - Imin:

<table>
<thead>
<tr>
<th></th>
<th>RMS</th>
<th>Max</th>
<th>Min</th>
<th>Signed Average</th>
<th>Unsigned Average</th>
<th>Avg_1</th>
<th>Avg_2</th>
<th>Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Region 1</td>
<td>13.3791</td>
<td>60</td>
<td>0</td>
<td>2</td>
<td>9</td>
<td>49</td>
<td>46</td>
<td>488</td>
</tr>
<tr>
<td>Region 2</td>
<td>7.1414</td>
<td>78</td>
<td>0</td>
<td>-3</td>
<td>4</td>
<td>19</td>
<td>23</td>
<td>62455</td>
</tr>
<tr>
<td>Region 3</td>
<td>6.4807</td>
<td>76</td>
<td>0</td>
<td>-4</td>
<td>4</td>
<td>6</td>
<td>10</td>
<td>183822</td>
</tr>
<tr>
<td>Region 4</td>
<td>13.5277</td>
<td>132</td>
<td>0</td>
<td>-5</td>
<td>10</td>
<td>21</td>
<td>26</td>
<td>6418</td>
</tr>
<tr>
<td>Region 5</td>
<td>28.9482</td>
<td>118</td>
<td>0</td>
<td>-13</td>
<td>21</td>
<td>39</td>
<td>53</td>
<td>713</td>
</tr>
</tbody>
</table>

Table 4.24

error of α:

<table>
<thead>
<tr>
<th></th>
<th>RMS</th>
<th>Max</th>
<th>Min</th>
<th>Signed Average</th>
<th>Unsigned Average</th>
<th>Avg_1</th>
<th>Avg_2</th>
<th>Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Region 1</td>
<td>11.6619</td>
<td>37</td>
<td>0</td>
<td>8</td>
<td>8</td>
<td>28</td>
<td>28</td>
<td>488</td>
</tr>
<tr>
<td>Region 2</td>
<td>10.3923</td>
<td>45</td>
<td>0</td>
<td>8</td>
<td>8</td>
<td>21</td>
<td>21</td>
<td>62455</td>
</tr>
<tr>
<td>Region 3</td>
<td>14.2127</td>
<td>45</td>
<td>0</td>
<td>11</td>
<td>11</td>
<td>19</td>
<td>19</td>
<td>183822</td>
</tr>
<tr>
<td>Region 4</td>
<td>16.0000</td>
<td>45</td>
<td>0</td>
<td>12</td>
<td>12</td>
<td>25</td>
<td>25</td>
<td>6418</td>
</tr>
<tr>
<td>Region 5</td>
<td>22.0000</td>
<td>45</td>
<td>0</td>
<td>17</td>
<td>17</td>
<td>27</td>
<td>27</td>
<td>713</td>
</tr>
</tbody>
</table>

Table 4.25
We get from the table of errors for pp and (Imax - Imin), all the errors are durable compared to the average in that region, they are also about the same amount errors as in experiment one. Also we notice that region5 where there is a high pp gradient has greater errors than region1, region2 and region4 because region5 contains mainly edges of the objects. In region3, there are large number of image points which are mostly not polarized, but due to the random noise and radiometric problem, the errors in that region are large compared to the small average, since we don't put too much interest on the region, so we don't worry about it.

The errors for α in region5 are however larger than in the same region in experiment one. The reason is that the objects are smaller in the image, there are not so many smooth and flat areas where there are quite constant phase angles.

4.6 Experiment Three

In this experiment, we take images to generate the look-up table for intensity linearization, linearize the images we take in experiment two, do warping and polarization parameter fitting, and compare the results with experiment two.

4.6.1 Taking Images for Look-up Table

We don't need the circle chart neither any objects on the scene. What we need is just a MacBeth ColorChecker chart which is put not normal to the cameras so that it can be polarized. The chart has to put at about the same position where we plan to put our objects in the experiments for fitting because we have to eliminate the error from focus as much as possible. Light is put in a position which forms a 120 angle from light to chart and from chart to cameras. We take images with four cameras respectively using the same light at the angles of 90, 65, 57, 48, 30, 20.

We set up the equipments the same way except that we use two polarizors instead of one.

Since we need to know the relative orientations of two filters, we find the relative orientation at angle 0 first. Ideal, the polarizer will block all the light to the cameras at the angle 0. We put a light bulb on the scene as the object because it is easy to tell the darkest scene. Then we adjust the primary filter (the one facing us) to 0 and rotate the secondary filter, in the process we observe significant changes of intensities. We get the darkest scene easily and keep the secondary filter at the position.

From now on we only adjust the primary filter and take images with four cameras at the angles of 90, 65, 57, 48, 30, 20. We don't take image at 0 because there is nothing on the image. Why do we have those odd numbers as 57, 48, 39 and so on? Because we want to spread our skeleton matching pairs evenly, the sin^2α of these angles are 1, 0.81, 0.70, 0.40, 0.25 and 0.12.
Figure 4.47, 4.48 and 4.49 show the images taken by a camera at the angles of 90, 48 and 20, please look at the great changes of intensities.
Figure 4.47, 4.48 and 4.49 show the images taken by a camera at the angles of 90, 48 and 20. Please look at the great changes of intensities.

Figure 4.47

Figure 4.48

Figure 4.49
4.6.2 Generating Look-up Table

Here are two steps to generate a look-up table:

(1) Find the average intensity of all squares

We calculate the average intensity of every square for all images using the program we describe in chapter 3.

(2) Generate the Look-up Table

We put all the angles and all the average intensities of every square in the look-up table generating program, we will get the look-up table which contains simply 255 numbers. Every camera needs its own look-up table.

The following figure shows us the look up table in the form of output intensity vs. input intensity. The curve is not perfectly smooth but it is quite OK.

![Figure 4.50](image)

4.6.3 Linearize Images

75
Linearizing an image is fairly easy once we have the look-up table. We just put the input intensities of an image as the index and the output is derived right away. Figure 4.51 show the linearized image of figure 4.47.

![Figure 4.51](image)

We then linearize all the images in experiment two.

4.6.4 Image Warping and Polarization Parameter Fitting

After all the images are linearized, we do image warping and polarization parameter fitting the same way as in experiment two.

First, the warped image at 30° by cam2 to cam1 is shown in figure 4.52.

![Figure 4.52](image)
Linearizing an image is fairly easy once we have the look-up table. We just put the input intensities of an image as the index and the output is derived right away. Figure 4.51 show the linearized image of figure 4.47.

![Figure 4.51](image)

We then linearize all the images in experiment two.

4.6.4 Image Warping and Polarization Parameter Fitting

After all the images are linearized, we do image warping and polarization parameter fitting the same way as in experiment two.

First, the warped image at 30° by cam2 to cam1 is shown in figure 4.52.

![Figure 4.52](image)
Then we check up the errors in table 4.25.

<table>
<thead>
<tr>
<th>Warping Cases</th>
<th>RMS</th>
<th>Max</th>
<th>Min</th>
<th>Unsigned Mean</th>
<th>Signed Mean</th>
</tr>
</thead>
<tbody>
<tr>
<td>cam2 to cam1 at 30</td>
<td>5.55</td>
<td>125</td>
<td>-115</td>
<td>3.24</td>
<td>0.49</td>
</tr>
<tr>
<td>cam2 to cam1 at 220</td>
<td>10.49</td>
<td>192</td>
<td>-142</td>
<td>6.01</td>
<td>4.82</td>
</tr>
<tr>
<td>cam3 to cam1 at 90</td>
<td>6.77</td>
<td>163</td>
<td>-142</td>
<td>3.44</td>
<td>-1.94</td>
</tr>
<tr>
<td>cam4 to cam1 at 260</td>
<td>6.59</td>
<td>158</td>
<td>-196</td>
<td>6.59</td>
<td>1.53</td>
</tr>
<tr>
<td>cam4 to cam1 at 140</td>
<td>6.02</td>
<td>61</td>
<td>-53</td>
<td>6.02</td>
<td>1.946</td>
</tr>
</tbody>
</table>

Table 4.25

We find the errors are little bigger than the warping in experiment two. A lot of those errors come from the linearization which usually spreads the intensities to a broader range. Our big chart board is unfortunately quite smooth thus it looks more polarized by the linearization than in the original case. Different cameras see different shiny areas on the board, and the difference is carried to the output through warping, that's why we see larger errors. The warping itself is good, we can check up the following two images:

(1) The normalized image of difference between warped image at 30 by cam2 to cam1 without linearization in figure 4.53;

(2) The normalized image of difference between warped image at 30 by cam2 to cam1 with linearization in figure 4.54.
Then we check up the errors in table 4.25.

<table>
<thead>
<tr>
<th>Warping Cases</th>
<th>RMS</th>
<th>Max</th>
<th>Min</th>
<th>Unsigned Mean</th>
<th>Signed Mean</th>
</tr>
</thead>
<tbody>
<tr>
<td>cam2 to cam1 at 30</td>
<td>5.55</td>
<td>125</td>
<td>-115</td>
<td>3.24</td>
<td>0.49</td>
</tr>
<tr>
<td>cam2 to cam1 at 220</td>
<td>10.49</td>
<td>192</td>
<td>-142</td>
<td>6.01</td>
<td>4.82</td>
</tr>
<tr>
<td>cam3 to cam1 at 90</td>
<td>6.77</td>
<td>163</td>
<td>-142</td>
<td>3.44</td>
<td>-1.94</td>
</tr>
<tr>
<td>cam3 to cam1 at 260</td>
<td>6.59</td>
<td>158</td>
<td>-196</td>
<td>6.59</td>
<td>1.53</td>
</tr>
<tr>
<td>cam4 to cam1 at 140</td>
<td>6.02</td>
<td>61</td>
<td>-53</td>
<td>6.02</td>
<td>1.946</td>
</tr>
</tbody>
</table>

Table 4.25

We find the errors are little bigger than the warping in experiment two. A lot of those errors come from the linearization which usually spreads the intensities to a broader range. Our big chart board is unfortunately quite smooth thus it looks more polarized by the linearization than in the original case. Different cameras see different shiny areas on the board, and the difference is carried to the output through warping, that's why we see larger errors. The warping itself is good, we can check up the following two images:

(1) The normalized image of difference between warped image at 30 by cam2 to cam1 without linearization in figure 4.53:

(2) The normalized image of difference between warped image at 30 by cam2 to cam1 with linearization in figure 4.54.
Now we put all the linearized images to the fitting program, and get two sets of results, one from the unwarped images, one from the warped images.
Now we put all the linearized images to the fitting program, and get two sets of results, one from the unwarped images, one from the warped images.
4.6.5 Error Analysis

In this section, we compare the results of the two fittings in the same different regions as in experiment two.

4.6.5.1. Error analysis in regions of Imax - Imin, degree of polarization pp and phase angle α.

The difference images ΔImax, ΔImin, ΔIavg, Δpp, Δt and Δe are generated from the results of the two fittings, then the errors, listed in the following table, of pp, (Imax - Imin) and α are calculated in the 10 regions as the following:

(1) Region 1: region of error where RMSE ≥ 30 in either warped or unwarped image of RMSE

(2) Region 2: region of high pp and high (Imax - Imin) of the unwarped images, where pp ≥ 105 and (Imax - Imin) ≥ 35

(3) Region 3: region of high pp and medium (Imax - Imin) of the unwarped images, where pp ≥ 105 and 15 ≤ (Imax - Imin) < 35

(4) Region 4: region of high pp and low (Imax - Imin) of the unwarped images, where pp > 105 and (Imax - Imin) < 15

(5) Region 5: region of medium pp and high (Imax - Imin) of the unwarped images, where 50 ≤ pp < 105 and (Imax - Imin) ≥ 35

(6) Region 6: region of medium pp and medium (Imax - Imin) of the unwarped images, where 50 ≤ pp < 105 and 15 ≤ (Imax - Imin) < 35

(7) Region 7: region of medium pp and low (Imax - Imin) of the unwarped images, where 10 ≤ pp < 35 and (Imax - Imin) < 15

(8) Region 8: region of low pp and high (Imax - Imin) of the unwarped images, where pp < 10 and (Imax - Imin) ≥ 40

(9) Region 9: region of low pp and medium (Imax - Imin) of the unwarped images, where pp < 10 and 15 ≤ (Imax - Imin) < 40

(10) Region 10: region of low pp and low (Imax - Imin) of the unwarped images, where pp < 10 and (Imax - Imin) < 15
errors for pp:

<table>
<thead>
<tr>
<th>Region</th>
<th>RMS</th>
<th>Max</th>
<th>Min</th>
<th>Signed Average</th>
<th>Unsigned Average</th>
<th>Avg_1</th>
<th>Avg_2</th>
<th>Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Region 1</td>
<td>28.7750</td>
<td>255</td>
<td>0</td>
<td>-4</td>
<td>11</td>
<td>8</td>
<td>13</td>
<td>155942</td>
</tr>
<tr>
<td>Region 2</td>
<td>5.3852</td>
<td>95</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>41</td>
<td>42</td>
<td>624</td>
</tr>
<tr>
<td>Region 3</td>
<td>3.0000</td>
<td>22</td>
<td>0</td>
<td>-2</td>
<td>2</td>
<td>29</td>
<td>32</td>
<td>956</td>
</tr>
<tr>
<td>Region 4</td>
<td>ND</td>
<td>ND</td>
<td>ND</td>
<td>ND</td>
<td>ND</td>
<td>ND</td>
<td>ND</td>
<td>0</td>
</tr>
<tr>
<td>Region 5</td>
<td>21.4243</td>
<td>110</td>
<td>0</td>
<td>4</td>
<td>15</td>
<td>71</td>
<td>67</td>
<td>4263</td>
</tr>
<tr>
<td>Region 6</td>
<td>5.6569</td>
<td>67</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>22</td>
<td>22</td>
<td>16747</td>
</tr>
<tr>
<td>Region 7</td>
<td>2.6458</td>
<td>5</td>
<td>0</td>
<td>-2</td>
<td>2</td>
<td>8</td>
<td>11</td>
<td>11</td>
</tr>
<tr>
<td>Region 8</td>
<td>15.7797</td>
<td>172</td>
<td>0</td>
<td>3</td>
<td>11</td>
<td>80</td>
<td>76</td>
<td>36403</td>
</tr>
<tr>
<td>Region 9</td>
<td>18.8149</td>
<td>189</td>
<td>0</td>
<td>9</td>
<td>12</td>
<td>16</td>
<td>15</td>
<td>18495</td>
</tr>
<tr>
<td>Region 10</td>
<td>18.9473</td>
<td>185</td>
<td>0</td>
<td>-14</td>
<td>15</td>
<td>5</td>
<td>20</td>
<td>19125</td>
</tr>
</tbody>
</table>

Table 4.27

table 4.27

error for Imax - Imin:

<table>
<thead>
<tr>
<th>Region</th>
<th>RMS</th>
<th>Max</th>
<th>Min</th>
<th>Signed Average</th>
<th>Unsigned Average</th>
<th>Avg_1</th>
<th>Avg_2</th>
<th>Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Region 1</td>
<td>7.1414</td>
<td>167.000</td>
<td>0</td>
<td>-2</td>
<td>3</td>
<td>7</td>
<td>9</td>
<td>155942</td>
</tr>
<tr>
<td>Region 2</td>
<td>6.7052</td>
<td>36</td>
<td>0</td>
<td>-3</td>
<td>5</td>
<td>59</td>
<td>63</td>
<td>624</td>
</tr>
<tr>
<td>Region 3</td>
<td>7</td>
<td>28</td>
<td>0</td>
<td>-5</td>
<td>5</td>
<td>49</td>
<td>55</td>
<td>956</td>
</tr>
<tr>
<td>Region 4</td>
<td>ND</td>
<td>ND</td>
<td>ND</td>
<td>ND</td>
<td>ND</td>
<td>ND</td>
<td>ND</td>
<td>0</td>
</tr>
<tr>
<td>Region 5</td>
<td>8.4261</td>
<td>40</td>
<td>0</td>
<td>1</td>
<td>6</td>
<td>21</td>
<td>19</td>
<td>4263</td>
</tr>
<tr>
<td>Region 6</td>
<td>72.111</td>
<td>39</td>
<td>0</td>
<td>-2</td>
<td>5</td>
<td>24</td>
<td>26</td>
<td>16747</td>
</tr>
<tr>
<td>Region 7</td>
<td>5.7446</td>
<td>11</td>
<td>1</td>
<td>-4</td>
<td>4</td>
<td>16</td>
<td>21</td>
<td>11</td>
</tr>
<tr>
<td>Region 8</td>
<td>3</td>
<td>49</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>8</td>
<td>8</td>
<td>36403</td>
</tr>
<tr>
<td>Region 9</td>
<td>8.6023</td>
<td>52</td>
<td>0</td>
<td>-5</td>
<td>6</td>
<td>8</td>
<td>13</td>
<td>18495</td>
</tr>
<tr>
<td>Region 10</td>
<td>12.8452</td>
<td>42</td>
<td>0</td>
<td>-10</td>
<td>11</td>
<td>4</td>
<td>15</td>
<td>19125</td>
</tr>
</tbody>
</table>

Table 4.28
Table 4.29

<table>
<thead>
<tr>
<th>Region</th>
<th>RMS</th>
<th>Max</th>
<th>Min</th>
<th>Signed Average</th>
<th>Unsigned Average</th>
<th>Avg_1</th>
<th>Avg_2</th>
<th>Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Region 1</td>
<td>9.1104</td>
<td>45</td>
<td>0</td>
<td>4</td>
<td>4</td>
<td>6</td>
<td>7</td>
<td>155942</td>
</tr>
<tr>
<td>Region 2</td>
<td>3.3166</td>
<td>25</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>25</td>
<td>26</td>
<td>624</td>
</tr>
<tr>
<td>Region 3</td>
<td>1.7321</td>
<td>14</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>24</td>
<td>25</td>
<td>956</td>
</tr>
<tr>
<td>Region 4</td>
<td>ND</td>
<td>ND</td>
<td>ND</td>
<td>ND</td>
<td>ND</td>
<td>ND</td>
<td>ND</td>
<td>0</td>
</tr>
<tr>
<td>Region 5</td>
<td>15.5242</td>
<td>45</td>
<td>0</td>
<td>11</td>
<td>11</td>
<td>25</td>
<td>24</td>
<td>4263</td>
</tr>
<tr>
<td>Region 6</td>
<td>7.2801</td>
<td>44</td>
<td>0</td>
<td>5</td>
<td>5</td>
<td>20</td>
<td>18</td>
<td>16747</td>
</tr>
<tr>
<td>Region 7</td>
<td>9.1104</td>
<td>14</td>
<td>1</td>
<td>8</td>
<td>8</td>
<td>17</td>
<td>25</td>
<td>11</td>
</tr>
<tr>
<td>Region 8</td>
<td>10.4881</td>
<td>45</td>
<td>0</td>
<td>7</td>
<td>7</td>
<td>20</td>
<td>20</td>
<td>36403</td>
</tr>
<tr>
<td>Region 9</td>
<td>16.7332</td>
<td>45</td>
<td>0</td>
<td>12</td>
<td>12</td>
<td>19</td>
<td>19</td>
<td>18495</td>
</tr>
<tr>
<td>Region 10</td>
<td>13.3791</td>
<td>45</td>
<td>0</td>
<td>9</td>
<td>9</td>
<td>16</td>
<td>17</td>
<td>19125</td>
</tr>
</tbody>
</table>

Look at the errors, the first thing we notice is that there are large number of errors and values of the errors are also huge. As we talked about in the last section, the linearization spreads the intensity to a broader range, so the background which is originally 30 to 40 is matched to about 2 to 3. But if the intensity of a image point is less than 4, the fitting program just ignores it and sets the RMSE to be -1, which turns to be 255 on the image. We can eliminate these errors simply by putting a smaller threshold in the fitting program. In this case, we keep the threshold same because we want to compare the results with experiment two. Actually these error points are background, we don't really concern about them.

Overall the errors in this experiment are smaller than in experiment two, especially in the high pp regions where our most concerns are. This improvement is clearly because intensity linearization removes a lot noise. However in the low pp regions there seem to be larger errors, this is also caused by intensity linearization, because in this case some medium gray level image points are matched to very high gray levels. The problem of the worsened result in pp low regions can be overcome by adjusting the lighting of the experiment. The right way is to get the white square of the chart as bright as we can and at the same time make the black square as dark as it can be.

4.6.5.2 Errors for regions of interest
We are interested in the same cube, cup, eggs and circles on the chart, the tables of errors are listed below:

(1) Errors for the cube

<table>
<thead>
<tr>
<th></th>
<th>rms</th>
<th>max</th>
<th>min</th>
<th>signed average</th>
<th>unsigned average</th>
<th>avg_1</th>
<th>avg_2</th>
<th>number</th>
</tr>
</thead>
<tbody>
<tr>
<td>error of pp</td>
<td>42.2374</td>
<td>143</td>
<td>0</td>
<td>-2</td>
<td>24</td>
<td>69</td>
<td>71</td>
<td>910</td>
</tr>
<tr>
<td>error of I_{max - I_{min}}</td>
<td>13.1149</td>
<td>55</td>
<td>0</td>
<td>-1</td>
<td>8</td>
<td>17</td>
<td>18</td>
<td>910</td>
</tr>
<tr>
<td>error of t</td>
<td>13.7113</td>
<td>44</td>
<td>0</td>
<td>9</td>
<td>9</td>
<td>34</td>
<td>32</td>
<td>910</td>
</tr>
</tbody>
</table>

Table 4.30

(2) Errors for the cup

<table>
<thead>
<tr>
<th></th>
<th>rms</th>
<th>max</th>
<th>min</th>
<th>signed average</th>
<th>unsigned average</th>
<th>avg_1</th>
<th>avg_2</th>
<th>number</th>
</tr>
</thead>
<tbody>
<tr>
<td>error of pp</td>
<td>41.7612</td>
<td>183</td>
<td>0</td>
<td>-11</td>
<td>19</td>
<td>28</td>
<td>40</td>
<td>1848</td>
</tr>
<tr>
<td>error of I_{max - I_{min}}</td>
<td>21.0476</td>
<td>167</td>
<td>0</td>
<td>-3</td>
<td>8</td>
<td>15</td>
<td>19</td>
<td>1848</td>
</tr>
<tr>
<td>error of t</td>
<td>14.1067</td>
<td>45</td>
<td>0</td>
<td>7</td>
<td>7</td>
<td>13</td>
<td>16</td>
<td>1848</td>
</tr>
</tbody>
</table>

Table 4.31

(3) Errors for the egg #2

<table>
<thead>
<tr>
<th></th>
<th>rms</th>
<th>max</th>
<th>min</th>
<th>signed average</th>
<th>unsigned average</th>
<th>avg_1</th>
<th>avg_2</th>
<th>number</th>
</tr>
</thead>
<tbody>
<tr>
<td>error of pp</td>
<td>29.7825</td>
<td>103</td>
<td>0</td>
<td>-12</td>
<td>20</td>
<td>20</td>
<td>32</td>
<td>1353</td>
</tr>
<tr>
<td>error of I_{max - I_{min}}</td>
<td>8.6603</td>
<td>72</td>
<td>0</td>
<td>-3</td>
<td>4</td>
<td>3</td>
<td>6</td>
<td>1353</td>
</tr>
<tr>
<td>error of t</td>
<td>19.8242</td>
<td>45</td>
<td>0</td>
<td>14</td>
<td>14</td>
<td>35</td>
<td>31</td>
<td>1353</td>
</tr>
</tbody>
</table>

Table 4.32
It is proved in experiment two that all the eggs have about same errors, we only show errors of one egg only.

(4) Errors for the circles

<table>
<thead>
<tr>
<th></th>
<th>rms</th>
<th>max</th>
<th>min</th>
<th>signed</th>
<th>unsigned average</th>
<th>avg_1</th>
<th>avg_2</th>
<th>number</th>
</tr>
</thead>
<tbody>
<tr>
<td>error of pp</td>
<td>10.3441</td>
<td>130</td>
<td>0</td>
<td>-3</td>
<td>6</td>
<td>48</td>
<td>52</td>
<td>8136</td>
</tr>
<tr>
<td>error of I_max-I_min</td>
<td>10.3441</td>
<td>41</td>
<td>0</td>
<td>-6</td>
<td>7</td>
<td>22</td>
<td>29</td>
<td>8136</td>
</tr>
<tr>
<td>error of t</td>
<td>9.1652</td>
<td>44</td>
<td>0</td>
<td>6</td>
<td>6</td>
<td>42</td>
<td>47</td>
<td>8136</td>
</tr>
</tbody>
</table>

Table 4.33

The errors of the cube and the circles are smaller than experiment two because of linearization. Also we find the two intensity averages in this experiment are much closer, this is because linearization can actually remove noise by matching a image point to a new gray value. This noise elimination effect of the intensity linearization is also shown in last error analysis. The errors of the egg and the cup are not much improved because the high reflective areas on the objects are very small. We have already explained in experiment two that small objects have big errors and suggested to use cameras with bigger zoom.

4.6.5.3 Error analysis in regions of pp gradient

In this section, we take the gradient of pp through out the image and look at the errors of gradient pp and pp in same five regions as in experiment two.

The gradient of pp is calculated the same ways as in experiment one. The normalized image of pp gradient is shown in figure 4.55.

Figure 4.55
It is proved in experiment two that all the eggs have about same errors, we only show errors of one egg only.

(4) Errors for the circles

<table>
<thead>
<tr>
<th>Error</th>
<th>rms</th>
<th>max</th>
<th>min</th>
<th>signed average</th>
<th>unsigned average</th>
<th>avg_1</th>
<th>avg_2</th>
<th>number</th>
</tr>
</thead>
<tbody>
<tr>
<td>error of pp</td>
<td>10.3441</td>
<td>130</td>
<td>0</td>
<td>-3</td>
<td>6</td>
<td>48</td>
<td>52</td>
<td>8136</td>
</tr>
<tr>
<td>error of lmax-lmin</td>
<td>10.3441</td>
<td>41</td>
<td>0</td>
<td>-6</td>
<td>7</td>
<td>22</td>
<td>29</td>
<td>8136</td>
</tr>
<tr>
<td>error of t</td>
<td>9.1652</td>
<td>44</td>
<td>0</td>
<td>6</td>
<td>6</td>
<td>42</td>
<td>47</td>
<td>8136</td>
</tr>
</tbody>
</table>

Table 4.33

The errors of the cube and the circles are smaller than experiment two because of linearization. Also we find the two intensity averages in this experiment are much closer, this is because linearization can actually remove noise by matching a image point to a new gray value. This noise elimination effect of the intensity linearization is also shown in last error analysis. The errors of the egg and the cup are not much improved because the high reflective areas on the objects are very small. We have already explained in experiment two that small objects have big errors and suggested to use cameras with bigger zoom.

4.6.5.3 Error analysis in regions of pp gradient

In this section, we take the gradient of pp through out the image and look at the errors of gradient pp and pp in same five regions as in experiment two.

The gradient of pp is calculated the same ways as in experiment one. The normalized image of pp gradient is shown in figure 4.55.

Figure 4.55
Table 3.33, 3.34 and 3.35 show the errors in the five regions.

**error of pp:**

<table>
<thead>
<tr>
<th></th>
<th>rms</th>
<th>max</th>
<th>min</th>
<th>signed average</th>
<th>unsigned average</th>
<th>avg_1</th>
<th>avg_2</th>
<th>number</th>
</tr>
</thead>
<tbody>
<tr>
<td>region1</td>
<td>35.6791</td>
<td>133</td>
<td>0</td>
<td>7</td>
<td>19</td>
<td>109</td>
<td>101</td>
<td>745</td>
</tr>
<tr>
<td>region2</td>
<td>27.5136</td>
<td>172</td>
<td>0</td>
<td>11</td>
<td>14</td>
<td>89</td>
<td>781</td>
<td>9645</td>
</tr>
<tr>
<td>region3</td>
<td>18.2209</td>
<td>248</td>
<td>0</td>
<td>-6</td>
<td>7</td>
<td>735</td>
<td>131</td>
<td>79329</td>
</tr>
<tr>
<td>region4</td>
<td>27.9285</td>
<td>225</td>
<td>0</td>
<td>-4</td>
<td>17</td>
<td>35</td>
<td>39</td>
<td>12557</td>
</tr>
<tr>
<td>region5</td>
<td>40.5586</td>
<td>255</td>
<td>0</td>
<td>-3</td>
<td>26</td>
<td>43</td>
<td>47</td>
<td>41620</td>
</tr>
</tbody>
</table>

**Table 4.34**

**error of Imax - Imin:**

<table>
<thead>
<tr>
<th></th>
<th>rms</th>
<th>max</th>
<th>min</th>
<th>signed average</th>
<th>unsigned average</th>
<th>avg_1</th>
<th>avg_2</th>
<th>number</th>
</tr>
</thead>
<tbody>
<tr>
<td>region1</td>
<td>11.8322</td>
<td>93</td>
<td>0</td>
<td>-2</td>
<td>6</td>
<td>20</td>
<td>22</td>
<td>745</td>
</tr>
<tr>
<td>region2</td>
<td>4.6904</td>
<td>70</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>12</td>
<td>11</td>
<td>19645</td>
</tr>
<tr>
<td>region3</td>
<td>7.1414</td>
<td>83</td>
<td>0</td>
<td>-3</td>
<td>3</td>
<td>8</td>
<td>11</td>
<td>179329</td>
</tr>
<tr>
<td>region4</td>
<td>10.8167</td>
<td>107</td>
<td>0</td>
<td>-4</td>
<td>7</td>
<td>15</td>
<td>19</td>
<td>12557</td>
</tr>
<tr>
<td>region5</td>
<td>8.2462</td>
<td>167</td>
<td>0</td>
<td>-1</td>
<td>5</td>
<td>8</td>
<td>10</td>
<td>41620</td>
</tr>
</tbody>
</table>

**Table 4.35**

**error of α:**

<table>
<thead>
<tr>
<th></th>
<th>rms</th>
<th>max</th>
<th>min</th>
<th>signed average</th>
<th>unsigned average</th>
<th>avg_1</th>
<th>avg_2</th>
<th>number</th>
</tr>
</thead>
<tbody>
<tr>
<td>region1</td>
<td>12.5698</td>
<td>45</td>
<td>0</td>
<td>8</td>
<td>8</td>
<td>24</td>
<td>24</td>
<td>745</td>
</tr>
<tr>
<td>region2</td>
<td>9.3808</td>
<td>45</td>
<td>0</td>
<td>6</td>
<td>6</td>
<td>21</td>
<td>21</td>
<td>19645</td>
</tr>
<tr>
<td>region3</td>
<td>8.3666</td>
<td>45</td>
<td>0</td>
<td>43</td>
<td>4</td>
<td>8</td>
<td>8</td>
<td>179329</td>
</tr>
<tr>
<td>region4</td>
<td>15.3623</td>
<td>45</td>
<td>0</td>
<td>11</td>
<td>11</td>
<td>20</td>
<td>20</td>
<td>12557</td>
</tr>
<tr>
<td>region5</td>
<td>15.5563</td>
<td>45</td>
<td>0</td>
<td>11</td>
<td>11</td>
<td>16</td>
<td>16</td>
<td>41620</td>
</tr>
</tbody>
</table>

**Table 4.36**

84
Compared to the errors in experiment two, the first thing we notice from the table is there is a much larger number of image points in region 5 in this experiment. Figure 4.55 shows clearly that there is a big area of background that has big pp gradient. This phenomenon which is not there in experiment two is original caused by lighting, linearization magnifies the effect.

Overall, the errors in all the regions are OK and about the same large as in experiment two. The errors in region 3 are much larger compared to the errors in the other regions because this region contains mostly background and it is very much noisy although linearization has already decreased the noise.

4.7 Conclusion and Discussion

Several conclusions and discussions are drawn from the three experiments:

(1) Generally speaking, image warping technology is usable with polarization parameter fitting, however it depends on the precision requirement of a particular application. Comparing the errors in different regions, it is found that errors are large in regions of background but are much smaller in regions of highlights where the diffuse image is recovered from.

(2) If highlight happens on an object, the shape of an object affects the warping and fitting very much. The more curve shaped an object is, the worse the results are obtained. The surface roughness also affects result in a way that rougher surfaces get more errors. The background of the experiments is carpet which is very rough, that is why there are lot of errors in this region.

(3) Bad focus of cameras can affect the results as well. Objects should not be placed too close to the cameras. However at the same time, the objects should not be too small compared the size of the image. Thus selecting right type of camera is very important.

(4) Intensity linearization is helpful to image warping and polarization parameter fitting. It can remove some noise by matching original image intensities to new values according to a look-up table. Generating the look-up table is therefore very critical. However, the lighting, the angle of the MacBeth ColorChecker chart to the cameras and the filter angles can all affect the look-up table. The look-up table generated in experiment three is not perfect mostly because of the lighting.

(5) Equipments play a very important role in the experiments. They should all function well and must be checked before taking images. It is positive that the higher quality and the higher model the equipments are, the better results can be obtained. The cameras we use can sense 256 gray levels, so illumination has to be adjusted and therefore produces more errors. The light source, the reflection board are both not perfect and they greatly increase the errors.
Appendix A

Program Codes

In this chapter, the source codes of the programs used in this project are listed.

The programs are:
1. Program of getting centroids of circles on the circle chart;
2. Program of calculating errors in the ten regions;
3. Program of calculating errors in the regions of interest;
4. Program of calculating errors in the regions of polarization degree gradients;
5. Program of generating intensity linearization look-up table;
6. Program of image intensity linearization;
7. Program of converting image from ppm format to pgm format.

Program codes:

```c
/*********************************************************
* Program Name: main.c                                   *
* Function: Getting centroids of circles on the circle chart *
* Implemented by: Yanhong Zhou                             *
***********************************************************/

#include <stdio.h>
#include "def.h"

static void help(); /* program of help */
static void binary(); /* program of binary thresholding */
static void histogram(); /* program of getting histogram */
static void centroid(); /* program of getting centroids for good images */
static void centroid_plus(); /* program of getting centroids for poor images */

/* main program */
main(argc, argv)
int argc;
char **argv;
{

int ac;
char **av;
int i;
char *programe;

programe = argv[0];

if (argc < 2 || *argv[1] != '-')
  help(programe);
```
ac  = argc;
av  = (char**)malloc(ac * sizeof(char*));
av[0] = argv[0]; for (i = 2; i < argc; i++) av[i-1] = argv[i];

switch (argv[1][1]){
case 'h':
    if (ac != 4) help(progname);
histogram(ac, av);
    break;
case 'c':
    if (ac != 7) help(progname);
    centroid(ac, av);
    break;
case 'p':
    if (ac != 9) help(progname);
    centroid_plus(ac, av);
    break;
case 'b':
    if (ac != 5) help(progname);
    binary(ac, av);
    break;
default:
    help(progname);
}

free(av);

exit(positive);
}
static void help(progname)
char *progname;
{
    fprintf(stderr, "%The flags and arguments of program %s are used as following:\n\n", progname);
    fprintf(stderr, "%h input output\n\n");
    fprintf(stderr, "% Plot histogram of input image and save it in output.\n\n");
    fprintf(stderr, "%c input T top bottom output1 output2\n\n");
    fprintf(stderr, "% Get centroid of input image with threshold top, threshold bottom and save the\n centroid-marked image in output1 and the result data file in output2\n\n");
    fprintf(stderr, "% T: threshold for making a binary image;\n\n");
    fprintf(stderr, "% top: balabala;\n\n");
    fprintf(stderr, "% bottom: balabala.\n\n");
    fprintf(stderr, "%Examples:\n\n");
    fprintf(stderr, "program -h left.pgm hist.out\n\n");
    fprintf(stderr, "program -c left.pgm 150 200 30 centroid-img:out centroid-img.txt\n\n");
    exit(negative);
}

static void histogram(ac, av)
int    ac;
char   **av;
{
char *input = av[1];
char *output = av[2];
imageP img1;
imageP img2;

img1 = readimage(input);
img2 = hist(img1);
writeimage(output, img2, img2->width, img2->height, img2->grey);

freeimage(img1);
freeimage(img2);

}
char *output2 = av[7];
imageP img1;
imageP img2;
imageP temp1;
circleP temp2;

img1 = readimage(input);
temp1 = threshold(img1, top);
temp2 = centroid_cal_plus(temp1, img1, top, bottom, max_size, min_size);
temp2 = sort(temp2, temp1);

img2 = show(img1, temp2, 42);
writeimage(output1, img2, img2->width, img2->height, img2->grey);
writeimage(output2, temp2);
freeimage(img1);
freeimage(temp1);
freecircle(temp2);
}

static void binary(ac, av)
int ac;
char **av;
{
    char *input = av[1];
    int threshold = atoi(av[2]);
    char *output = av[3];
    imageP img1;
    imageP img2;

    img1 = readimage(input);
    img2 = bin(img1, threshold);
    writeimage(output, img2, img2->width, img2->height, img2->grey);

    freeimage(img1);
    freeimage(img2);
}

******************************************************************************
* Program Name: hist.c                                               *
* Fuction: Getting histogram of a image                            *
* Implemented by: Yanhong Zhou                                     *
******************************************************************************

#include "def.h"

imageP hist(img1)
imageP img1;
{
    imageP img;

    int i, j;
}
int w, h, g;
int ww, hh;
float *r, second_r;
int *sum, *hist;

w = standard_width;
h = standard_height;
g = img1->grey;

ww = img1->width;
hh = img1->height;

r = (float *)malloc(sizeof(float)*(g+1));
sum = (int *)malloc(sizeof(int)*(g+1));
hist = (int *)malloc(sizeof(int)*w*h);

img = (imageP)malloc(sizeof(imageS));

for (i=0; i<(g+1); ++i)
    sum[i] = 0;

for (i=0; i<ww*hh; ++i)
    sum[*(img1->image+i)]++;

for (i=0; i<(g+1); ++i)
    r[i] = ((float)(sum[i])/(ww*hh));

second_r = get_second(r, (g+1));

for (i=0; i<(g+1); ++i)
    r[i] = (h-40) * r[i] / second_r;

for (i=0; i<w*h; ++i)
    hist[i] = g;

for (i=0; i<w; ++i)
    hist[w*(h-40)+i] = 0;

i = 50;
while(i <= w)
{
    for (j= (h - 41); j<(h - 35); ++j) hist[j*w + i] = 0;
    i = i + 50;
}

for (i=0; i<(g+1); ++i)
    for (j= (h - 40); j > (h - 40 - (int)(r[i])); --j)
        hist[j*w+i+50] = 0;

img->image = hist;
img->width = w;
img->height = h;
```c
img->grey = g;
free(r);
free(sum);
/* should not free hist */
return(img);
}

/********************************************************************************/
* Program Name: binary.c
* Fuction: Getting histogram of a image
* Implemented by: Yanhong Zhou
******************************************************************************/
#include "def.h"
imageP bin(img1, threshold)
imageP img1;
int threshold;
{
    imageP img;
    int i, j;
    int w, h, g;
    w = img1->width;
    h = img1->height;
    g = img1->grey;
    img = (imageP )malloc(sizeof(imageS));
    img->image = (int *)malloc(sizeof(int) * w * h);
    for (i=0; i<w*h; ++i)
    {
        if (*(img1->image+i) >= threshold) *(img->image+i) = white;
        else *(img->image+i) = black;
    }
    img->width = w;
    img->height = h;
    img->grey = g;
    return(img);
}

/********************************************************************************/
* Program Name: centroid.c
* Fuction: Getting centroids of circle chart
* Implemented by: Yanhong Zhou
******************************************************************************/
#include "def.h"
```
/*==========================================================================*/
circleP centroid_cal(temp1, img1, top, bottom)
imageP temp1;
imageP img1;
int top, bottom;
{
pixelsP pixels1;
pixelsP pixels2;
circleP circles;
int w, h, g;
int num;
int i, j;
int *r;

w = temp1->width;
h = temp1->height;
g = temp1->grey;
pixels1 = (pixelsP )malloc(sizeof(pixelsS));
pixels1->grey = (int *)malloc(sizeof(int) * w * h);
pixels1->mark = (int *)malloc(sizeof(int) * w * h);
pixels2 = (pixelsP )malloc(sizeof(pixelsS));
pixels2->grey = (int *)malloc(sizeof(int) * w * h);
pixels2->mark = (int *)malloc(sizeof(int) * w * h);
initialization_for_noise(temp1, pixels1);
num = mark(pixels1, w, h);

r = (int *)malloc(sizeof(int)*num);
circles = rm_noise(pixels1, num, w, h, r);

initialization_for_centroid(img1, pixels1, pixels2);
circles = get_centroid(circles, pixels2, w, h, top, bottom);
circles = orientation(circles, pixels1, w, h, r);

return(circles);
}

="/*******************************************************************/
circleP centroid_cal_plus(temp1, img1, top, bottom, max_size, min_size)
imageP temp1;
imageP img1;
int top, bottom;
int max_size, min_size;
{
pixelsP pixels1;
pixelsP pixels2;
circleP circles;
int w, h, g;
int num;
int i, j;
int *r;

w = temp1->width;
h = temp1->height;
g = temp1->grey;

pixels1 = (pixelsP)malloc(sizeof(pixelsS));
pixels1->grey = (int *)malloc(sizeof(int) * w * h);
pixels1->mark = (int *)malloc(sizeof(int) * w * h);

pixels2 = (pixelsP)malloc(sizeof(pixelsS));
pixels2->grey = (int *)malloc(sizeof(int) * w * h);
pixels2->mark = (int *)malloc(sizeof(int) * w * h);

initialization_for_noise(temp1, pixels1);
num = mark(pixels1, w, h);

r = (int *)malloc(sizeof(int) * num);

circles = rm_noise_plus(pixels1, num, w, h, max_size, min_size, r);

initialization_for_centroid(img1, pixels1, pixels2);
circles = get_centroid(circles, pixels2, w, h, top, bottom);
circles = orientation(circles, pixels1, w, h, r);

return(circles);
free(r);
}

/*============================================================================
 ==
 ==============================================================================
 ==*/
int check_end_row(column_number, k)
int column_number;
int k;
{
    int end;

    if((k % column_number) == 0) end = 1;
    else end = 0;

    return(end);
}
```c
void count_black(pixels, x, y, w, h, m, b, left, right)
{
pixelsP pixels;
int x, y;
int w, h;
double m, b;
int *left, *right;
{
    int i, j;
    int index;
    int count_left, count_right;

    count_left = 0;
    count_right = 0;

    i = x;
    j = y;

    while(j >= 0 && j <= h && i >= 0 && i <= w)
    {
        index = j * w + i;
        if( *(pixels->mark+index) != background) count_left++;
        j--;
        j = (int)(i * m + b);
    }

    i = x;
    j = y;

    while(j >= 0 && j <= h && i >= 0 && i <= w)
    {
        index = j * w + i;
        if( *(pixels->mark+index) != background) count_right++;
        j++;
        j = (int)(i * m + b);
    }

    *left = count_left;
    *right = count_right;
}
```

94
int count_column(circles, img, distance_x, distance_y)
circleP circles;
imageP img;
int distance_x, distance_y;
{
    int number;
    int index;
    int ok;
    int prdLx, prdLy;
    float a1, a2, b1, b2, a, b;

    ok = 1;
    number = 2;
    a1 = 0;
    a2 = 0;
    b1 = 0;
    b2 = 0;

    while( ok == 1 )
    {
        prediction_for_same_row(circles, number, &prdLx, &prdLy,
          distance_x, distance_y, a1, a2, b1, b2, &a, &b);
        index = prdLy * (img->width) + prdLx;
        if(*((img->image+index) == black))
            {
                match_for_same_row(circles, prdLx, prdLy,
                   number, &distance_x, &distance_y, &a1, &a2, &b1, &b2, a, b);
                number++;
            }
        else ok = 0;
    }

    return(number);
}
*(circles->x+ONE) = temp_x;
*(circles->y+ONE) = temp_y;
}

/*===============================================================================
===*/
int first_try(circles, pixels, w)
circleP circles;
pixelsP pixels;
int w;
{
    int done;
    int index;
    int two;
    int i;
    int temp_x, temp_y;
    done = 0;
    two = 0;
    for (i=0; i<circles->count; ++i)
    {
        index = *(circles->y + i) * w + *(circles->x + i);
        if(*(pixels->grey+index) == white)
        {
            if(*(circles->number+i) != two)
            {
                temp_x = *(circles->x + i);
                temp_y = *(circles->y + i);
                *(circles->x + i) = *(circles->x + two);
                *(circles->y + i) = *(circles->y + two);
                *(circles->x + two) = temp_x;
                *(circles->y + two) = temp_y;
            }
            two++;
        }
    }
    if( two == 2) done = 1;
    else done = 0;
    return(done);
}

/*===============================================================================
===*/
/*---------------------------------*/
96
```c
/**
circleP get_centroid(circles, pixels2, w, h, top, bottom)
circleP circles;
pixelsP pixels2;
int w, h;
int top, bottom;
{
    int i, k;
    float sum_x, sum_y, sum_weight;
    float *weight;

    weight = (float *)malloc(sizeof(float)*w*h);
    k = 0;

    for(i=0; i<w*h; ++i)
        {
            if(*(pixels2->grey+i) >= top) weight[i] = 0.0;
            else if(*(pixels2->grey+i) <= bottom) weight[i] = 1.0;
            else weight[i] = ((float)(top - *(pixels2->grey+i))) / (float)((top - bottom));
        }

    while(k < circles->count){
        sum_x = 0;
        sum_y = 0;
        sum_weight = 0;
        for(i=0; i<w*h; ++i)
            if (*((pixels2->mark+i) == k))
                {
                    sum_x = sum_x + weight[i] * (i % w);
                    sum_y = sum_y + weight[i] * ((int)(i/w));
                    sum_weight = sum_weight + weight[i];
                }
        *(circles->x+k) = (int)(sum_x/sum_weight);
        *(circles->y+k) = (int)(sum_y/sum_weight);
        k++;
    }

    free(weight);
    return(circles);
}
*/

/*===============================================================================
================================================================================
===*/
void geCorientation(circles, pixels, w, h)
circleP circles;
pixelsP pixels;
int w, h;
{  
```
int x1, y1, x2, y2;
double m, b;
int left_one, right_one, left_two, right_two;
int min_black;

x1 = *(circles->x);
y1 = *(circles->y);
x2 = *(circles->x+1);
y2 = *(circles->y+1);

get_line(&m, &b, x1, y1, x2, y2);

count_black(pixels, x1, y1, w, h, m, b, &left_one, &right_one);
count_black(pixels, x2, y2, w, h, m, b, &left_two, &right_two);

min_black = left_one;

if(right_one < min_black) min_black = right_one;
if(left_two < min_black) min_black = left_two;
if(right_two < min_black) min_black = right_two;

if((left_two == min_black) || (right_two == min_black))
    exchange(circles);

/*===============================================================================
/*===============================================================================

void initialization_for_centroid(img1, pixels1, pixels2)
imageP img1;
pixelsP pixels1, pixels2;
{
    int w, h, g;
    int i;

    w = img1->width;
    h = img1->height;
    g = img1->grey;

    pixels2->grey = img1->image;
    pixels2->mark = pixels1->mark;
}

/*===============================================================================
/*===============================================================================

void initialization_for_noise(temp1, pixels)
imageP temp1;
pixelsP pixels;
{
  int w, h, g;
  int i;

  w = temp1->width;
  h = temp1->height;
  g = temp1->grey;

  pixels->grey = temp1->image;

  for (i=0; i<w*h; ++i)
    *(pixels->mark+i) = background;
}

/*============================================================================
============================================================================
===*/
int mark(pixels1 , w, h)
pixelsP pixels1;
int w, h;
{
  int i;
  int k;
  int count;

  k = 0;

  for (i=0; i<w*h; i++)
    if (*(pixels1->grey+i) == black && *(pixels1->mark+i) == background)
      {
        count = 0;
        markcircle(pixels1, i, k, w, h, count);
        k++;
      }

  return(k);
}

/*============================================================================
============================================================================
===*/
void markcircle(pixels1, i, k, w, h, count)
pixelsP pixels1;
int k, w, h, i, count;
{
  /*
   
   */
}
if(count < 50000)
{
    count++;
    *(pixels1->mark+i) = k;

    if ( (i+1) < w*h && (i % w) != (w-1) )
        if (*(pixels1->mark+i+1) == background
            && *(pixels1->grey+i+1) == black) markcircle(pixels1, i+1, k, w, h, count);

    if ( (i-1) >= 0 && (i % w) != 0 )
        if (*(pixels1->mark+i-1) == background
            && *(pixels1->grey+i-1) == black) markcircle(pixels1, i-1, k, w, h, count);

    if ( (i+w) < w*h)
        if (*(pixels1->mark+i+w) == background
            && *(pixels1->grey+i+w) == black) markcircle(pixels1, i+w, k, w, h, count);

    if ( (i-w) >= 0)
        if (*(pixels1->mark+i-w) == background
            && *(pixels1->grey+i-w) == black) markcircle(pixels1, i-w, k, w, h, count);
}

/*=============================================================
 *================================================================
 */

void match_for_next_row(circles, prdt_x, prdt_y, number, distance_x, distance_y,
                        a1, a2, b1, b2, dx, dy)

circleP circles;
int prdt_x, prdt_y;
int number;
int *distance_x, *distance_y;
float *a1, *a2, *b1, *b2;
int dx, dy;
{
    int *distance;
    int min, min_distance;
    int temp_x, temp_y;
    int i;

    distance = (int *)malloc(sizeof(int) * circles->count);

    for (i=0; i<circles->count; ++i)
        distance[i] = (*(circles->x+i) - prdt_x) *(*(circles->y+i) - prdt_y)
                    + (*(circles->y+i) - prdt_x) *(*(circles->y+i) - prdt_y);

    min = 0;
    min_distance = distance[0];
for (i=0; i<circles->count; ++i) {
    if (distance[i] < min_distance) {
        min_distance = distance[i];
        min = i;
    }
}

temp_x = *(circles->x + number);
temp_y = *(circles->y + number);

*(circles->x + number) = *(circles->x + min);
*(circles->y + number) = *(circles->y + min);

*(circles->x + min) = temp_x;
*(circles->y + min) = temp_y;

*distance_x = dx;
*distance_y = dy;

*a1 = 0;
*b1 = 0;
*a2 = 0;
*b2 = 0;

free(distance);
}

/*============================================================================================================*/
void match_for_same_row(circles, prdt_x, prdt_y, number, distance_x, distance_y, a1, a2, b1, b2, a, b)
circleP circles;
int prdt_x, prdt_y;
int number;
int *distance_x, *distance_y;
float *a1, *a2, *b1, *b2;
float a, b;
{
    int *distance;
    int min, min_distance;
    int temp_x, temp_y;
    int i;

    distance = (int *)malloc(sizeof(int) * circles->count);

    for (i=0; i<circles->count; ++i)
        distance[i] = (*(circles->x+i) - prdt_x) * (*(circles->x+i) - prdt_x) +
                      (*(circles->y+i) - prdt_y) * (*(circles->y+i) - prdt_y);
\begin{verbatim}
+ (*circles->y+i) * (*circles->y+i) - prd_y);

min = 0;
min_distance = distance[0];

for (i=0; i<circles->count; ++i)
{
  if (distance[i] < min_distance)
  {
    min_distance = distance[i];
    min = i;
  }
}

temp_x = *(circles->x + number);
temp_y = *(circles->y + number);

*(circles->x + number) = *(circles->x + min);
*(circles->y + number) = *(circles->y + min);

*(circles->x + min) = temp_x;
*(circles->y + min) = temp_y;

*a1 = a;
*b1 = b;
*a2 = (float)(*(circles->x+number) - *distance_x)/(float)(*circles->x+number-1))
  - 1.0;
*b2 = (float)(*(circles->y+number) - *distance_y)/(float)(*circles->y+number-1))
  - 1.0;
*distance_x = *(circles->x+number) - *(circles->x+number-1);
*distance_y = *(circles->y+number) - *(circles->y+number-1);

free(distance);
}

/*=============================================*/
/*
circleP orientation(circles, pixels, w, h, r)
circleP circles;
pixelsP pixels;
int w, h;
int *r;
{
  int done;

done = first_try(circles, pixels, w);

if (!done) done = second_try(circles, pixels, w, r);

102
\end{verbatim}
if(!done)
{
    printf("wrong image: probably too much noise\n");
    exit(negative);
}
else get_orientation(circles, pixels, w, h);
return(circles);

/*===============================================================================
================================================================================
===*/
int out_of_range(bx, by, ax, ay, w, h)
int bx, by, ax, ay;
int w, h;
{
    int yes;

    if(ax > w || ax < 0) yes = 1;
    if(bx > w || bx < 0) yes = 1;
    if(ay > h || ay < 0) yes = 1;
    if(by > h || by < 0) yes = 1;
    else yes = 0;

    return(yes);
}
/*===============================================================================
================================================================================
===*/
void prediction_for_next_row(circles, column_number, k, prdt_x, prdt_y,
    distance_x, distance_y)
circleP circles;
int column_number;
int k;
int *prdt_x, *prdt_y;
int distance_x, distance_y;
{

    *prdt_x = *(circles->x + k - column_number) - distance_y;
    *prdt_y = *(circles->y + k - column_number) + distance_x;
}
/*===============================================================================
==

103
void prediction_for_same_row(circles, number, prdt_x, prdt_y, distance_x, distance_y, a1, a2, b1, b2, a, b)

circleP circles;
int number;
int *prdt_x, *prdt_y;
int distance_x, distance_y;
float a1, a2, b1, b2;
float *a, *b;
{
  *a = (a1 + a2) / 2;
  *b = (b1 + b2) / 2;

  *prdt_x = (int)(*((circles->x + number - 1) * (1 + *a)) + distance_x;
  *prdt_y = (int)(*((circles->y + number - 1) * (1 + *b)) + distance_y;
}

===

void re_mark(pixels, w, h, count, i)
pixelsP pixels;
int w, h, count, i;
{
  int j;

  for(j=0; j<w*h; j++)
    if(*(pixels->mark+j) == i) *(pixels->mark+j) = count;
}

===

void rm_mark(pixels, w, h, i)
pixelsP pixels;
int w, h, i;
{
  int j;

  for(j=0; j<w*h; j++)
    if(*(pixels->mark+j) == i) *(pixels->mark+j) = background;
circleP rm_noise(pixels1, num, w, h, r)
pixelsP pixels1;
int num;
int w, h;
int *r;
{
circleP circles;
int i, k, count;
int sum_size, sum_x, sum_y, average_size;
int size;

circles = (circleP )malloc(sizeof(circleS));
circles->number = (int *)malloc(sizeof(int)*num);
circles->size = (int *)malloc(sizeof(int)*num);
circles->x = (int *)malloc(sizeof(int)*num);
circles->y = (int *)malloc(sizeof(int)*num);

k = 0;
sum_size = 0;
count = 0;

while(k < num)
{
    size = 0;
    for(i=0; i<w*h; ++i)
        if (*(pixels1->mark+i) == k)
            size++;
    *(circles->number+k) = k;
    *(circles->size+k) = size;
    r[k] = (int)(sqrt((double)(size/pi)));
    sum_size = sum_size + size;
    k++;
}

average_size = (int)(sum_size/num);

for(i=0; i<num; ++i)
{
    if ( *(circles->size+i) <= (int)(size_const * average_size) &&
        *(circles->size+i) >= (int)(average_size / size_const))
    {
        *(circles->size + count) = *(circles->size + i);
        *(circles->number + count) = count;
        re_mark(pixels1, w, h, count, i);
        r[count] = r[i];
        count++;
    }
    else rm_mark(pixels1, w, h, i);
}
```c
int circles->count = count;

return(circles);
}

/*===============================================================================
================================================================================
===*/
circleP rm_noise_plus(pixels1, num, w, h, max_size, min_size, r)
pixelsP pixels1;
int num;
int w, h;
int max_size, min_size;
int *r;
{
circleP circles;
int i, k, count;
int sum_x, sum_y;
int size;
int remove;

circles = (circleP )malloc(sizeof(circleS));
circles->number = (int *)malloc(sizeof(int)*num);
circles->size = (int *)malloc(sizeof(int)*num);
circles->x = (int *)malloc(sizeof(int)*num);
circles->y = (int *)malloc(sizeof(int)*num);

k = 0;
count = 0;

while(k < num)
{
    size = 0;
    sum_x = 0;
    sum_y = 0;
    for(i=0; i<w*h; ++i)
        if (*((pixels1->mark+i) == k)
            { 
                sum_x = sum_x + (i % w);
                sum_y = sum_y + (int)(i/w);
                size++;
            }
    *(circles->number+k) = k;
    *(circles->size+k) = size;
    *(circles->x+k) = (int)(sum_x/size);
    *(circles->y+k) = (int)(sum_y/size);
    r[k] = (int)(sqrt((double)(size/pi)));
    k++;
}
```
for(i=0; i<num; ++i)
{
    if( *(circles->size+i) <= max_size &&
        *(circles->size+i) >= min_size)
    {
        *(circles->size + count) = *(circles->size + i);
        *(circles->number + count) = count;
        *(circles->x + count) = *(circles->x + i);
        *(circles->y + count) = *(circles->y + i);
        r[count] = r[i];
        re_mark(pixels1, w, h, count, i);
        count++;
    }
    else rm_mark(pixels1, w, h, i);
}

num = count;
count = 0;

for(i=0; i<num; ++i)
{
    k = 0;
    remove = 0;
    while( k < w*h && remove == 0)
    {
        if( *(pixels1->mark+k) == i)
        {
            if( dis((k % w), (int)(k / w), *(circles->x+i), *(circles->y+i)) >
                (int)(shape_const * r[i]))
            {
                rm_mark(pixels1, w, h, i);
                remove = 1;
            }
        }
        k++;
    }
    if(remove == 0)
    {
        re_mark(pixels1, w, h, count, i);
        *(circles->size + count) = *(circles->size + i);
        r[count] = r[i];
        count++;
    }
}
circles->count = count;

return(circles);
}

/*===============================================================================
===================================================================================*/
int search(pixels, x, y, w)
    pixelsP pixels;
    int x, y;
    int w;
    {
        int index, i;
        int xx, yy;
        int yes;

        xx = x - radius;
        yy = y - radius;

        do
        {
            index = yy * w + xx;
            xx++;
            yy++;
        }
        while( *(pixels->grey+index) == black && xx < (x + radius));

        if (xx < (x + radius)) yes = 1;
        else yes = 0;

        return(yes);
    }
/*

/*===============================================================================
=====================~==========================================================
===*/
int second_try(circles, pixels, w, r)
circleP circles;
pixelsP pixels;
int w;
int *r;
{
    int done;
    int index;
    int xx, yy;
    int two;
    int i;
    int temp_x, temp_y;

    done = 0;
    two = 0;

    for (i=0; i<circles->count; ++i)
    {
        yy = *(circles->y+i) - (int)(radius_const * r[i]);
        do
        {
\[ xx = \ast (\text{circles} \rightarrow x+i) - (\text{int})(radius\_\text{const} \ast r[i]); \]

\begin{verbatim}
    do
        index = yy * w + xx;
        xx++;
    } while( *(pixels->grey+index) == black &&
        xx < \ast (\text{circles} \rightarrow x+i) + (\text{int})(radius\_\text{const} \ast r[i]));
    yy++;

    while( *(pixels->grey+index) == black &&
        yy < \ast (\text{circles} \rightarrow y+i) + (\text{int})(radius\_\text{const} \ast r[i]));

    if (xx < \ast (\text{circles} \rightarrow x+i) + (\text{int})(radius\_\text{const} \ast r[i]) ||
        yy < \ast (\text{circles} \rightarrow y+i) + (\text{int})(radius\_\text{const} \ast r[i]))
    { 
        temp_x = \ast (\text{circles} \rightarrow x + i);
        temp_y = \ast (\text{circles} \rightarrow y + i);
        \ast (\text{circles} \rightarrow x + i) = \ast (\text{circles} \rightarrow x + two);
        \ast (\text{circles} \rightarrow y + i) = \ast (\text{circles} \rightarrow y + two);
        \ast (\text{circles} \rightarrow x + two) = temp_x;
        \ast (\text{circles} \rightarrow y + two) = temp_y;
        two++;
    }
    
    if( two == 2) done = 1;
    else done = 0;

    return(done);
\end{verbatim}

/**=============================================*/

```c
imageP show(img1, temp2, test)
imageP img1;
circleP temp2;
int test;
{
    imageP img2;
    int index;
    int i;
    int w, h, g;

    w = img1->width;
    h = img1->height;
    g = img1->grey;

    img2 = (imageP)malloc(sizeof(imageS));
    ```
img2->image = (int *)malloc(sizeof(int) * w * h);

img2 = img1;

for (i=0; i<temp2->count; ++i)
{
  if(*(temp2->number+i) != test)
  {
    index = *(temp2->y+i) * w + *(temp2->x+i);
    *(img2->image + index) = white;
  }
}
return(img2);

/*===============================================================================
================================================================================
===*/
circleP sort(circles, img)
circleP circles;
imageP img;
{
  int end_of_row;
  int column_number;
  int distance_x, distance_y;
  int dx, dy;
  int predicted_x, predicted_y;
  float a1, a2, b1, b2, a, b;
  int i, k;

  distance_x = *(circles->x + 1) - *(circles->x);
  distance_y = *(circles->y + 1) - *(circles->y);

  a1 = 0;
  a2 = 0;
  b1 = 0;
  b2 = 0;

  column_number = count_column(circles, img, distance_x, distance_y);

  k = 2;

  while(k < circles->count)
  {
    end_of_row = check_end_row(column_number, k);
    if(!end_of_row)
    {
      if(k == 2)
      {
        dx = distance_x;
\[ dy = \text{distance}_y; \]

\[
\text{prediction_for_same_row}(\text{circles}, k, \&\text{predicted}_x, \&\text{predicted}_y, \\
\quad \text{distance}_x, \text{distance}_y, a1, a2, b1, b2, \&a, \&b); \\
\text{match_for_same_row}(\text{circles}, \text{predicted}_x, \text{predicted}_y, k, \\
\quad \&\text{distance}_x, \&\text{distance}_y, \&a1, \&a2, \&b1, \&b2, \&a1, \&b2, \&a, \&b); \\
\text{if}( (k \% \text{column}_\text{number}) == 1) \\
\quad \{ \\
\quad \quad dx = \text{distance}_x; \\
\quad \quad dy = \text{distance}_y; \\
\quad \} \\
\text{else} \\
\quad \{ \\
\quad \quad \text{prediction_for_next_row}(\text{circles}, \text{column}_\text{number}, k, \\
\quad \quad \quad \&\text{predicted}_x, \&\text{predicted}_y, dx, dy); \\
\quad \quad \text{match_for_next_row}(\text{circles}, \text{predicted}_x, \text{predicted}_y, k, \\
\quad \quad \quad \&\text{distance}_x, \&\text{distance}_y, \&a1, \&a2, \&b1, \&b2, \&dx, \&dy); \\
\quad \} \\
k++; \\
\}

\text{return}(\text{circles});
\}

/*========================================================================
   ==
   ==========================================================================
   ==*/

\text{imageP}\ \text{threshold}(\text{img1}, T)
\text{imageP} \text{img1};
\text{int} T;
\{
  \text{imageP} \text{img};
  \text{int} w, h, g;
  \text{int} i, j;
  w = \text{img1->width};
  h = \text{img1->height};
  g = \text{img1->grey};
  \text{img} = (\text{imageP})\text{malloc}(\text{sizeof(imageS)});
  \text{img->image} = (\text{int} *)\text{malloc}(\text{sizeof(int)*w*h});
  \text{for}(i=0; i<w*h; ++i)
  \quad \{ \\
  \quad \quad \text{if}( *(\text{img1->image}+i) >= T) *(\text{img->image}+i) = \text{white}; \\
  \quad \quad \text{else} *(\text{img->image}+i) = \text{black}; \\
  \quad \} \\
  \text{img->width} = w;
  \text{img->height} = h;
img->grey = g;

return(img);
}

/***************************************************************************/
/* Program Name:   def.h   */
/* Function:      header of definitions used by progrm    */
/* Getting centroids of circle chart                     */
/* Implemented by: Yanhong Zhou */
/***************************************************************************/
#include <sys/types.h>
#include <unistd.h>
#include <stdio.h>
#include <math.h>
#include <malloc.h>
#include <stdlib.h>

typedef struct {
    int *grey;
    int *mark;
} pixelsS, *pixelsP;

typedef struct {
    int width;  /* number of columns in the image */
    int height; /* number of rows in the image    */
    int grey;  /* number of gray levels in the image */
    int *image; /* pointer to the image data */
} imageS, *imageP;

typedef struct {
    int count;
    int *number;
    int *size;
    int *x;
    int *y;
} circleS, *circleP;

#define pi 3.1415926
#define black 0
#define white 255
#define radius_const 0.5
#define positive 1
#define negative 1
#define ONE 1
#define ZERO 0
#define standard_width 640
#define standard_height 480

112
#define background -99
#define size_const 3
#define shape_const 3

extern imageP bin();
extern circleP centroid_cal();
extern circleP centroid_cal_plus();
extern int check_end_row();
extern void count_black();
extern int count_column();
extern int dis();
extern void exchange();
extern int first_try();
extern void freecircle();
extern void freeimage();
extern circleP get_centroid();
extern void get_line();
extern void get_orientation();
extern float get_second();
extern imageP hist();
extern void initialization_for_centroid();
extern void initialization_for_noise();
extern void match_for_next_row();
extern void match_for_same_row();
extern int mark();
extern void markcircle();
extern circleP orientation();
extern int out_of_range();
extern void prediction_for_next_row();
extern void prediction_for_same_row();
extern imageP readimage();
extern void re_mark();
extern void rm_mark();
extern imageP rm_noise();
extern circleP rm_noise_plus();
extern int search();
extern int second_try();
extern imageP show();
extern circleP sort();
extern imageP threshold();
extern int who_is();
extern void writeimage();
extern void writetext();

******************************************************************************
* Program Name: def.c
* Function: some basic functions
* Implemented by: Yanhong Zhou
******************************************************************************

#include "def.h"
imageP readimage(filename)
char *filename;
{
  int w, h, g;
  int i;
  FILE *fd;
  imageP img;
  char junk[2];

  fd = fopen(filename, "r");

  img = (imageP)malloc(sizeof(imageS));

  fscanf(fd, "%ss", junk);
  fscanf(fd, "%d", &img->width);
  fscanf(fd, "%d", &img->height);
  fscanf(fd, "%d", &img->grey);

  w = img->width;
  h = img->height;
  g = img->grey;

  img->image = (int *)malloc(w*h*sizeof(int));
  for(i=0; i<w*h; ++i)
    fscanf(fd, "%d", (img->image+i));

  fclose(fd);

  return (img);
}

void writeimage(filename, img, w, h, g)
char *filename;
imageP img;
int w, h, g;
{
  int i;
  FILE *fd;

  fd = fopen(filename, "w");

  fprintf(fd, "%s
", "P2");
  fprintf(fd, "%d %d
", w, h);
  fprintf(fd, "%d
", g);

  for(i=0; i<w*h; ++i)
    { 
    fprintf(fd, "%d ", *(img->image+i));
    if( (i+1) % 17 == 0 ) fprintf(fd, "n");
    }
  fclose(fd);
}
void writetext(char *filename, circleP temp)
{
    int i;
    FILE *fd;

    fd = fopen(filename, "w");
    fprintf(fd, "%d\n", temp->count);
    for(i=0; i<temp->count; ++i)
    {
        fprintf(fd, "%d %d\n", *(temp->x+i), *(temp->y+i));
    }
    fclose(fd);
}

void freeimage(imageP img)
{
    free(img->image);
    free(img);
}

void freecircle(circleP circles)
{
    free(circles->number);
    free(circles->size);
    free(circles->x);
    free(circles->y);
    free(circles);
}

float get_second(float *r, int size)
{
    float maxr;
    float secr;
    int max_number;
    int i;

    maxr = r[0];
    for(i=1; i<size; ++i)
    {
        if (r[i] >= maxr)
        {
            maxr = r[i];
        }
    }
    return maxr;
}
max_number = i;
}

r[max_number] = 0;

secr = r[0];
for(i=1; i<size; ++i)
  if (r[i] >= secr)
    { 
      secr = r[i];
    }
return secr;
}

void get_line(m, b, x11, y11, x21, y21)
double *m, *b;
int x11, x21, y11, y21;
{
  *m = ((float)(y21-y11))/((float)(x21-x11));
  *b = ((float)(y21*x11 - y11*x21))/((float)(x11 - x21));
}

int dis(x1, y1, x2, y2)
int x1, x2, y1, y2;
{
  int d;
  d = (int)(sqrt((double)((x1 - x2) * (x1 - x2) + (y1 - y2) * (y1 - y2))));
  return(d);
}

/***************************************************************************/
/* Program Name: error_10.c                                          */
/* Fuction: Getting errors in the ten regions                         */
/* Implemented by: Yanhong Zhou                                      */
/***************************************************************************/

#include <stdio.h>
#include "polar_def.h"
static void help();
static void polar();
static void get_region_10();
static void get_error_10();

main(argc, argv)
int argc;
char **argv;
{
  int ac;
  char **av;
  int i;
  char *progname;
progname = argv[0];

if (argc != 5 )
  help(progname);

ac = argc;
av = (char **)malloc(ac * sizeof(char *));

for(i=0; i<argc; i++)
  av[i] = argv[i];

polar(ac, av);

exit(positive);
}

static void help()
{
  printf("usage:\n");
  printf("polar start_x, start_y, end_x, end_y\n");
}

static void polar(ac, av)
int ac;
char **av;
{
  int start_x = atoi(av[1]);
  int start_y = atoi(av[2]);
  int end_x   = atoi(av[3]);
  int end_y   = atoi(av[4]);

  imageP  img1_imax;
  imageP  img1_imin;
  imageP  img1_avg;
  imageP  img1_pp;
  imageP  img1_e;
  imageP  img1_t;

  imageP  img2_imax;
  imageP  img2_imin;
  imageP  img2_avg;
  imageP  img2_pp;
  imageP  img2_e;
  imageP  img2_t;

  imageP  delta_pp;
  imageP  delta_i;
  imageP  delta_t;

  imageP  delta_pp_5;
imageP *delta_i_5;
imageP *delta_t_5;

int *region_count;
int *region_avg;
/
int *region_count_5;
*/

int w, h, g;

img1_imax = readimage_interest(in_imax_1, start_x, start_y, end_x, end_y);
img1_imin = readimage_interest(in_imin_1, start_x, start_y, end_x, end_y);
img1_avg = readimage_interest(in_average_1, start_x, start_y, end_x, end_y);
img1_pp = readimage_interest(in_pp_1, start_x, start_y, end_x, end_y);
img1_e = readimage_interest(in_e_1, start_x, start_y, end_x, end_y);
img1_t = readimage_interest(in_t_1, start_x, start_y, end_x, end_y);

img2_imax = readimage_interest(in_imax_2, start_x, start_y, end_x, end_y);
img2_imin = readimage_interest(in_imin_2, start_x, start_y, end_x, end_y);
img2_avg = readimage_interest(in_average_2, start_x, start_y, end_x, end_y);
img2_pp = readimage_interest(in_pp_2, start_x, start_y, end_x, end_y);
img2_e = readimage_interest(in_e_2, start_x, start_y, end_x, end_y);
img2_t = readimage_interest(in_t_2, start_x, start_y, end_x, end_y);

w = img1_imax->width;

h = img1_imax->height;

g = img1_imax->grey;

delta_pp = (imageP)malloc(sizeof(imageS));
delta_pp->image = (int *)malloc(sizeof(int) * w * h);
delta_pp->region = (int *)malloc(sizeof(int) * w * h);
delta_pp->width = w;
delta_pp->height = h;
delta_pp->grey = g;

delta_i = (imageP)malloc(sizeof(imageS));
delta_i->image = (int *)malloc(sizeof(int) * w * h);
delta_i->region = (int *)malloc(sizeof(int) * w * h);
delta_i->width = w;
delta_i->height = h;
delta_i->grey = g;

delta_t = (imageP)malloc(sizeof(imageS));
delta_t->image = (int *)malloc(sizeof(int) * w * h);
delta_t->region = (int *)malloc(sizeof(int) * w * h);
delta_t->width = w;
delta_t->height = h;
delta_t->grey = g;

region_count = (int *)malloc(sizeof(int) * 10);
region_avg = (int *)malloc(sizeof(int) * 60);

gregion_10(img1_imax, img1_imin, img1_avg, img1_pp, img1_e, img1_t,
    img2_imax, img2_imin, img2_avg, img2_pp, img2_e, img2_t,
    delta_pp, delta_i, delta_t, region_count, region_avg);

gerror_10(img1_imax, img1_imin, img1_avg, img1_pp, img1_e, img1_t,
    img2_imax, img2_imin, img2_avg, img2_pp, img2_e, img2_t,
    delta_pp, delta_i, delta_t, region_count, region_avg);

writeimage("figure_3.54", delta_t, w, h, g);

freeimage(img1_imax);
freeimage(img1_imin);
freeimage(img1_avg);
freeimage(img1_pp);
freeimage(img1_e);
freeimage(img1_t);
freeimage(img2_imax);
freeimage(img2_imin);
freeimage(img2_avg);
freeimage(img2_pp);
freeimage(img2_e);
freeimage(img2_t);
freeimage(delta_pp);
freeimage(delta_i);
freeimage(delta_t);
free(region_count);
free(region_avg);
}

static void geLregion_10(imgCimax, img1_imin,
    imgCavg, img1_pp, imgCe, imgCt,
    img2_imax, img2_imin, img2_avg, img2_pp, img2_e, img2_t,
    delta_pp, delta_i, delta_t, region_count, region_avg)

imageP img1_imax;
imageP img1_imin;
imageP img1_avg;
imageP img1_pp;
imageP img1_e;
imageP img1_t;
imageP img2_imax;
imageP img2_imin;
imageP img2_avg;
imageP img2_pp;
imageP img2_e;
imageP img2_t;
imageP delta_pp;
imageP delta_i;
int *region_count;
int *region_avg;
```c
int i, j;
int w, h, g;
int *index;
int *region_sum;

w = img1_imax->width;
h = img1_imax->height;
g = img1_imax->grey;

index = (int *)malloc(sizeof(int) * w * h);
region_sum = (int *)malloc(sizeof(int) * 60);

for(i=region1; i<=region10; ++i)
    region_count[i] = 0;
for(i=0; i<60; ++i)
    region_sum[i] = 0;
for(i=0; i<w*h; ++i)
{
    if( *(img1_e->image + i) >= T_e || *(img2_e->image + i) >= T_e)
    {
        index[i] = region1;
        region_count[region1]++;
        region_sum[region1 * 6 + 0] = region_sum[region1 * 6 + 0] + *(img1_pp->image + i);
        region_sum[region1 * 6 + 1] = region_sum[region1 * 6 + 1] + *(img2_pp->image + i);
        region_sum[region1 * 6 + 2] = region_sum[region1 * 6 + 2] + *(img1_imax->image + i) - *(img1_imin->image + i);
        region_sum[region1 * 6 + 3] = region_sum[region1 * 6 + 3] + *(img2_imax->image + i) - *(img2_imin->image + i);
        region_sum[region1 * 6 + 4] = region_sum[region1 * 6 + 4] + *(img1_t->image + i);
        region_sum[region1 * 6 + 5] = region_sum[region1 * 6 + 5] + *(img2_t->image + i);
    }
    else if( abs(*(img1_imax->image + i) - *(img1_imin->image + i)) >= T_i_high &&
             (*(img1_pp->image + i) >= T_pp_high))
    {
        index[i] = region2;
        region_count[region2]++;
        region_sum[region2 * 6 + 0] = region_sum[region2 * 6 + 0] + *(img1_pp->image + i);
        region_sum[region2 * 6 + 1] = region_sum[region2 * 6 + 1] + *(img2_pp->image + i);
        region_sum[region2 * 6 + 2] = region_sum[region2 * 6 + 2] + *(img1_imax->image + i) - *(img1_imin->image + i);
        region_sum[region2 * 6 + 3] = region_sum[region2 * 6 + 3] + *(img2_imax->image + i) - *(img2_imin->image + i);
        region_sum[region2 * 6 + 4] = region_sum[region2 * 6 + 4] + *(img1_t->image + i);
        region_sum[region2 * 6 + 5] = region_sum[region2 * 6 + 5] + *(img2_t->image + i);
    }
    else if( abs(*(img1_imax->image + i) - *(img1_imin->image + i)) >= T_i_high &&
             (*(img1_pp->image + i) < T_pp_high) &&
/*img1_pp*/image + i) > T_pp_high)

} else if( abs(*img1_imin->image + i) >= abs(*img1_imax->image + i) - (img1_imax->image + i)) > T_i_high &&
/*img1_pp*/image + i) < T_pp_low))

} else if( abs(*img1_imax->image + i) - *img1_imin->image + i) < T_i_low &&
/*img1_imax*/image + i) > T_i_high) &&
/*img1_pp*/image + i) >= T_pp_low))

} else if( abs(*img1_imax->image + i) - *img1_imin->image + i) > T_i_high &&
/*img1_imax*/image + i) < T_i_low) &&
/*img1_pp*/image + i) >= T_pp_high))

} else if( abs(*img1_imax->image + i) - *img1_imin->image + i) < T_i_low &&
/*img1_imax*/image + i) > T_i_high) &&
/*img1_pp*/image + i) < T_pp_high))

} else if( abs(*img1_imax->image + i) - *img1_imin->image + i) > T_i_high &&
/*img1_imax*/image + i) < T_i_low) &
/*img1_pp*/image + i) >= T_pp_low))

} else if( abs(*img1_imax->image + i) - *img1_imin->image + i) < T_i_low &&
/*img1_imax*/image + i) > T_i_high) &
/*img1_pp*/image + i) < T_pp_high))

} else if( abs(*img1_imax->image + i) - *img1_imin->image + i) < T_i_low &&
/*img1_imax*/image + i) > T_i_high) &
/*img1_pp*/image + i) < T_pp_high))

}
else if( abs(*(img1_imax->image + i)) - *(img1_imin->image + i)) < T_i_high &&
    *(img1_imax->image + i) - *(img1_imin->image + i)) >= T_i_low &&
    *(img1_pp->image + i) < T_pp_low))
{
    index[i] = region7;
    region_sum[region7 * 6 + 0] = region_sum[region7 * 6 + 0] + *(img1_pp->image + i);
    region_sum[region7 * 6 + 1] = region_sum[region7 * 6 + 1] + *(img2_pp->image + i);
    region_sum[region7 * 6 + 2] = region_sum[region7 * 6 + 2] + *(img1_imax->image + i) -
    *(img1_imin->image + i);
    region_sum[region7 * 6 + 3] = region_sum[region7 * 6 + 3] + *(img2_imax->image + i) -
    *(img2_imin->image + i);
    region_sum[region7 * 6 + 4] = region_sum[region7 * 6 + 4] + *(img1_t->image + i);
    region_sum[region7 * 6 + 5] = region_sum[region7 * 6 + 5] + *(img2_t->image + i);
}
else if( abs(* (img1_imax->image + i)) - *(img1_imin->image + i)) < T_i_low &&
    *(img1_pp->image + i) >= T_pp_high))
{
    index[i] = region8;
    region_sum[region8 * 6 + 0] = region_sum[region8 * 6 + 0] + *(img1_pp->image + i);
    region_sum[region8 * 6 + 1] = region_sum[region8 * 6 + 1] + *(img2_pp->image + i);
    region_sum[region8 * 6 + 2] = region_sum[region8 * 6 + 2] + *(img1_imax->image + i) -
    *(img1_imin->image + i);
    region_sum[region8 * 6 + 3] = region_sum[region8 * 6 + 3] + *(img2_imax->image + i) -
    *(img2_imin->image + i);
    region_sum[region8 * 6 + 4] = region_sum[region8 * 6 + 4] + *(img1_t->image + i);
    region_sum[region8 * 6 + 5] = region_sum[region8 * 6 + 5] + *(img2_t->image + i);
}
else if( abs(* (img1_imax->image + i)) - *(img1_imin->image + i)) < T_i_low &&
    *(img1_pp->image + i) < T_pp_high) &&
    *(img1_pp->image + i) >= T_pp_low))
{
    index[i] = region9;
    region_sum[region9 * 6 + 0] = region_sum[region9 * 6 + 0] + *(img1_pp->image + i);
    region_sum[region9 * 6 + 1] = region_sum[region9 * 6 + 1] + *(img2_pp->image + i);
    region_sum[region9 * 6 + 2] = region_sum[region9 * 6 + 2] + *(img1_imax->image + i) -
    *(img1_imin->image + i);
    region_sum[region9 * 6 + 3] = region_sum[region9 * 6 + 3] + *(img2_imax->image + i) -
    *(img2_imin->image + i);
    region_sum[region9 * 6 + 4] = region_sum[region9 * 6 + 4] + *(img1_t->image + i);
    region_sum[region9 * 6 + 5] = region_sum[region9 * 6 + 5] + *(img2_t->image + i);
}
else if( abs(* (img1_imax->image + i)) - *(img1_imin->image + i)) < T_i_low &&
    *(img1_pp->image + i) < T_pp_low)
{ 
  index[i] = region10;
  region_count[region10]++;
  region_sum[region10 * 6 + 0] = region_sum[region10 * 6 + 0] + *(img1_pp->image + i);
  region_sum[region10 * 6 + 1] = region_sum[region10 * 6 + 1] + *(img2_pp->image + i);
  region_sum[region10 * 6 + 2] = region_sum[region10 * 6 + 2] + *(img1_imax->image + i) - *(img1_imin->image + i); 
  region_sum[region10 * 6 + 3] = region_sum[region10 * 6 + 3] + *(img2_imax->image + i) - *(img2_imin->image + i);
  region_sum[region10 * 6 + 4] = region_sum[region10 * 6 + 4] + *(img1_t->image + i);
  region_sum[region10 * 6 + 5] = region_sum[region10 * 6 + 5] + *(img2_t->image + i);
}

for(i=0; i<w*h; ++i)
{
  *(delta_pp->region + i) = index[i];
  *(delta_i->region + i) = index[i];
  *(delta_t->region + i) = index[i];
}

for(i = region1; i <= region10; ++i)
  for(j = 0; j < 6; ++j)
  {
    if(region_count[i] > 0)
    {
      if(j == 4 || j == 5)
        region_avg[i * 6 + j] = (int)(0.5*(region_sum[i * 6 + j] / region_count[i]));
      else
        region_avg[i * 6 + j] = (int)(region_sum[i * 6 + j] / region_count[i]);
    }
    else
      region_avg[i * 6 + j] = ND;
  }

static void get_error_10(img1_imax, img1_imin, img1_avg, img1_pp, img1_e, img1_t, 
  img2_imax, img2_imin, img2_avg, img2_pp, img2_e, img2_t, 
  delta_pp, delta_i, delta_t, region_count, region_avg)

imageP img1_imax;
imageP img1_imin;
imageP img1_avg;
imageP img1_pp;
imageP img1_e;
imageP img1_t;
imageP img2_imax;
imageP img2_imin;
imageP img2_avg;
imageP img2_pp;
imageP img2_e;
imageP img2_t;

123
imageP img2_t;
imageP delta_pp;
imageP delta_i;
imageP delta_t;
int *region_count;
int **region_avg;
{
    int i, j;
    int w, h, g;
    FILE *fp;

    float **error_pp;
    float **error_i;
    float **error_t;

    error_pp = (float **)malloc(sizeof(float *) * 5);
    error_i = (float **)malloc(sizeof(float *) * 5);
    error_t = (float **)malloc(sizeof(float *) * 5);

    w = img1_imax->width;
    h = img1_imax->height;
    g = img1_imax->grey;

    for(i=0; i<w*h; ++i)
    {
        *(delta_pp->image + i) = *(img1_pp->image + i) - *(img2_pp->image + i);
        *(delta_i->image + i) = (*(img1_imax->image + i) - *(img1_imin->image + i)) -
                               (*(img2_imax->image + i) - *(img2_imin->image + i));
        *(delta_t->image + i) = 0.5 * get_delta_t(*(img1_t->image + i),*(img2_t->image + i));
    }

    error_pp[0] = get_e(delta_pp, w, h, region_count, 10);
    error_pp[1] = get_max(delta_pp, w, h, region_count, 10);
    error_pp[2] = get_min(delta_pp, w, h, region_count, 10);
    error_pp[3] = get_signed_avg(delta_pp, w, h, region_count, 10);
    error_pp[4] = get_unsigned_avg(delta_pp, w, h, region_count, 10);

    error_i[0] = get_e(delta_i, w, h, region_count, 10);
    error_i[1] = get_max(delta_i, w, h, region_count, 10);
    error_i[2] = get_min(delta_i, w, h, region_count, 10);
    error_i[3] = get_signed_avg(delta_i, w, h, region_count, 10);
    error_i[4] = get_unsigned_avg(delta_i, w, h, region_count, 10);

    error_t[0] = get_e(delta_t, w, h, region_count, 10);
    error_t[1] = get_max(delta_t, w, h, region_count, 10);
    error_t[2] = get_min(delta_t, w, h, region_count, 10);
    error_t[3] = get_signed_avg(delta_t, w, h, region_count, 10);
    error_t[4] = get_unsigned_avg(delta_t, w, h, region_count, 10);

    fp = fopen(error_file_10, "w+");
fprintf(fp, "region1: error region --- e > %d\n", T_e);
fprintf(fp, "region2: delta_pp high, Imax-Imin high --- pp >= %d, Imax - Imin >= %d\n",
    T_pp_high, T_i_high);
fprintf(fp, "region3: delta_pp high, Imax-Imin medium --- pp >= %d, %d > Imax - Imin >= %d\n",
    T_pp_high, T_i_high, T_i_low); fprintf(fp, "region4: delta_pp high, Imax-Imin low --- pp >= %d, Imax - Imin < %d\n",
    T_pp_high, T_i_high, T_i_low);
fprintf(fp, "region5: delta_pp medium, Imax-Imin high --- %d > pp >= %d, %d > Imax - Imin >= %d\n",
    T_pp_high, T_i_low, T_i_high);
fprintf(fp, "region6: delta_pp medium, Imax-Imin medium --- %d > pp >= %d, %d > Imax - Imin >= %d\n",
    T_pp_high, T_i_high, T_i_low);
fprintf(fp, "region7: delta_pp medium, Imax-Imin low --- %d > pp >= %d, Imax - Imin < %d\n",
    T_pp_high, T_i_low, T_i_high);
fprintf(fp, "region8: delta_pp low, Imax-Imin high --- pp < %d, Imax - Imin >= %d\n",
    T_pp_low, T_i_high);
fprintf(fp, "region9: delta_pp low, Imax-Imin medium --- pp < %d, %d > Imax - Imin >= %d\n",
    T_pp_low, T_i_high, T_i_low);
fprintf(fp, "region10: delta_pp low, Imax-Imin low --- pp < %d, Imax - Imin < %d\n",
    T_pp_low, T_i_low);
fprintf(fp, "error for pp:\n");
fprintf(fp, "  rms max min signed_avg unsigned_avg avg_1 avg_2 number\n");
fprintf(fp, "\n");
for(j=region1; j<=region10; ++j)
{
    fprintf(fp, "region%d: \n", j+1);
    for(i=0; i<S; ++i)
    {
        if(error_pp[i][j] == NO)
            fprintf(fp, "NO\n");
        else fprintf(fp, "%9.4f", error_pp[i][j]);
    }
    if(region_count[j] == 0)
        fprintf(fp, "ND ND\n");
    else
    {
        fprintf(fp, "%10d", region_avg[j * 6 + 0]);
        fprintf(fp, "%10d", region_avg[j * 6 + 1]);
    }
    fprintf(fp, "%10d", region_count[j]);
    fprintf(fp, "\n");
}
fprintf(fp, "\n");
fprintf(fp, "error for Imax - Imin:\n");
fprintf(fp, "  rms max min signed_avg unsigned_avg avg_1 avg_2 number\n");
fprintf(fp, "\n");
for(j=region1; j<=region10; ++j)
{
    fprintf(fp, "region%d: \n", j+1);
    for(i=0; i<S; ++i)
    {
        printf(fp, "%d", i);
    }
    fprintf(fp, "\n");
}
fprintf(fp, "\n");
if(error_i[j][j] == NO)
    fprintf(fp, "ND");
else fprintf(fp, "%9.4f", error_i[j][j]);
}
if(region_count[j] == 0)
    fprintf(fp, "ND ND");
else {
    fprintf(fp, "%10d", region_avg[j * 6 + 2]);
    fprintf(fp, "%10d", region_avg[j * 6 + 3]);
}
fprintf(fp, "%10d", region_count[j]);
fprintf(fp, "n");
}
fprintf(fp, "n");
fprintf(fp, "error for t:\n");
fprintf(fp, "rms max min signed_avg unsigned_avg avg_1 avg_2 number\n");
for(j=region1; j<=region10; j++)
{
    fprintf(fp, "region%d:\n", j+1);
    for(i=0; i<5; i++)
    {
        if(error_i[j][i] == NO)
            fprintf(fp, "ND");
        else fprintf(fp, "%9.4f", error_i[j][i]);
    }
    if(region_count[j] == 0)
        fprintf(fp, "ND ND");
    else {
        fprintf(fp, "%10d", region_avg[j * 6 + 4]);
        fprintf(fp, "%10d", region_avg[j * 6 + 5]);
    }
    fprintf(fp, "%10d", region_count[j]);
    fprintf(fp, "n");
}
fclose(fp);
free(error_pp);
free(error_i);
free(error_t);

/***** Program Name: error.c ******/
/***** Fuction: Functions of getting different kinds of errors ******/
/***** Implemented by: Yanhong Zhou ******/
#include "polar_def.h"

int get_delta_t(x, y)
int x, y;
{
    int delta_t;
    int a, b, c;
    a = abs(x - y);
    b = abs(x - y + 180);
    c = abs(x - y - 180);
    delta_t = a;
    if(b < delta_t) delta_t = b;
    if(c < delta_t) delta_t = c;
    return(delta_t);
}

float get_image_avg(img)
imageP img;
{
    int w, h;
    int sum;
    int i;
    float avg;
    w = img->width;
    h = img->height;
    sum = 0;
    for(i=0; i<w*h; ++i)
        sum = sum + *(img->image + i);
    avg = (float)(sum)/(float)(w*h);
    return(avg);
}

float *get_e(img, w, h, region_count, count)
imageP img;
int w, h;
int *region_count;
int count;
{
    float *e;
int *sum;

int i, j;

e = (float *)malloc(sizeof(float)*count);
sum = (int *)malloc(sizeof(int)*count);

for(i=0; i<count; ++i)
    sum[i] = 0;

for(i=0; i<w*h; ++i)
{
    for(j=0; j<count; ++j)
    {
        if(*(img->region + i) == j)
        {
            sum[j] = sum[j] + abs(*(img->image + i)) * abs(*(img->image + i));
        }
    }
}

for(i=0; i<count; ++i)
{
    if (region_count[i] > 0)
        e[i] = sqrt(sum[i]/region_count[i]);
    else e[i] = NA;
}

free (sum);
return (e);
}

float get_e_interest(img, w, h)
imageP img;
int w, h;
{
    float e;
    int sum;
    int i;

    sum = 0;

    for(i=0; i<w*h; ++i)
    {
        sum = sum + abs(*(img->image + i)) * abs(*(img->image + i));
    }

    e = sqrt(sum/(w*h));

    return(e);
}

float *get_max(img, w, h, region_count, count)
imageP img;
int w, h;
int *region_count;
int count;
float *get_max(img, w, h, region_count, count)
imageP img;
int w, h, count;
float *max = (float *)malloc(sizeof(float) * count);

for (i=0; i<count; ++i)
{
    if (region_count[i] > 0)
    {
        j = 0;
        while(*(img->region + j) != i && j < w * h)
            j++;
        max[i] = abs(*(img->image + j));
    }
    else max[i] = NA;
}

for (i=0; i<w*h; ++i)
{
    for (j=0; j<count; ++j)
    {
        if (max[j] != NA)
            if(*(img->region + i) == j && abs(*(img->image + i)) > max[j])
            max[j] = abs(*(img->image + i));
    }
    return(max);
}

float get_max_interest(img, w, h)
imageP img;
int w, h;
{
    float max;
    int i, j;

    max = abs(*(img->image));

    for (i=0; i<w*h; ++i)
    {
        if (abs(*(img->image + i)) > max)
            max = abs(*(img->image + i));
    }

    return(max);
}

float *get_min(img, w, h, region_count, count)
imageP img;
int w, h;
int *region_count;
int count;
{
    float *min;
    int i, j;

    min = (float *)malloc(sizeof(float) * count);

    for (i=0; i<count; ++i)
    {
        if ( region_count[i] > 0 )
        {
            j = 0;
            while( *(img->region + j) != i && j < w * h)
            {
                j++;
                min[i] = abs(*(img->image + j));
            }
            else min[i] = NA;
        }
    }

    for(i=0; i<w*h; ++i)
    {
        for(j=0; j<count; ++j)
        {
            if(min[i] != NA)
            {
                if( *(img->region + i) == j && abs(*(img->image + i)) < min[j])
                {
                    min[j] = abs(*(img->image + i));
                }
            }
        }
    }

    return(min);
}

float get_min_interest(img, w, h)
imageP img;
int w, h;
{
    float min;
    int i, j;

    min = abs(*(img->image));

    for(i=0; i<w*h; ++i)
    {
        if(abs(*(img->image + i)) < min)
        {
            min = abs(*(img->image + i));
        }
    }

    return(min);
}
float *get_signed_avg(img, w, h, region_count, count)
    imageP img;
    int w, h;
    int *region_count;
    int count;
{
    float *signed_avg;
    int *sum;
    int i, j;

    signed_avg = (float *)malloc(sizeof(float) * count);
    sum = (int *)malloc(sizeof(int) * count);

    for(i=0; i<count; ++i)
        sum[i] = 0;

    for(i=0; i<w*h; ++i)
    {
        for(j=0; j<count; ++j)
            if(*((img->region + i) == j))
                sum[j] = sum[j] + *(img->image + i);
    }

    for(i=0; i<count; ++i)
    {
        if(region_count[i] > 0)
            signed_avg[i] = (float)(sum[i]/region_count[i]);
        else signed_avg[i] = NA;
    }

    free(sum);
    return(signed_avg);
}

float get_signed_avg_interest(img, w, h)
    imageP img;
    int w, h;
{
    float signed_avg;
    int sum;
    int i;

    sum = 0;

    for(i=0; i<w*h; ++i)
        sum = sum + *(img->image + i);

    signed_avg = (float)(sum/(w*h));

    return(signed_avg);
}
float *get_unsigned_avg(img, w, h, region_count, count)
imageP img;
int w, h;
int *region_count;
int count;
{
    float *unsigned_avg;
    int *sum;
    int i, j;
    unsigned_avg = (float *)malloc(sizeof(float) * count);
    sum = (int *)malloc(sizeof(int) * count);
    for(i=0; i<count; ++i)
        sum[i] = 0;
    for(i=0; i<w*h; ++i)
        {
            for(j=0; j<count; ++j)
                if(*(img->region + i) == j)
                    sum[j] = sum[j] + abs(*(img->image + i));
        }
    for(i=0; i<count; ++i)
        {
            if( region_count[i] > 0)
                unsigned_avg[i] = (float)(sum[i]/region_count[i]);
            else unsigned_avg[i] = NA;
        }
    free(sum);
    return(unsigned_avg);
}

float get_unsigned_avg_interest(img, w, h)
imageP img;
int w, h;
{
    float unsigned_avg;
    int sum;
    int i;
    sum = 0;
    for(i=0; i<w*h; ++i)
        sum = sum + abs(*(img->image + i));
    unsigned_avg = (float)(sum/(w*h));
float *get_e_5(img, w, h, region_count_5)
imageP img;
int w, h;
int *region_count_5;
{
    float *e;
    int *sum;

    int i;

    e = (float *)malloc(sizeof(float)*5);
    sum = (int *)malloc(sizeof(int)*5);

    for(i=region1; i<=region5; ++i)
        sum[i] = 0;

    for(i=0; i<w*h; ++i)
        {
            if(*(img->region+i) == region1)
                sum[region1] = sum[region1] + abs(*(img->image + i)) * abs(*(img->image + i));
            else if(*(img->region+i) == region2)
                sum[region2] = sum[region2] + abs(*(img->image + i)) * abs(*(img->image + i));
            else if(*(img->region+i) == region3)
                sum[region3] = sum[region3] + abs(*(img->image + i)) * abs(*(img->image + i));
            else if(*(img->region+i) == region4)
                sum[region4] = sum[region4] + abs(*(img->image + i)) * abs(*(img->image + i));
            else if(*(img->region+i) == region5)
                sum[region5] = sum[region5] + abs(*(img->image + i)) * abs(*(img->image + i));
        }

    for(i=region1; i<=region5; ++i)
        {
            if (region_count_5[i] > 0)
                e[i] = sqrt(sum[i]/region_count_5[i]);
            else e[i] = NA;
        }

    free(sum);
    return(e);
}

/*
float *get_max_5(img, w, h, region_count_5)
imageP img;
int w, h;
int *region_count_5;
{
    float *max;
int i, j;

max = (float *)malloc(sizeof(float) * 5);

for (i=region1; i<=region5; ++i)
{
    if (region_count_5[i] > 0)
    {
        j = 0;
        while ( *(img->region + j) != i && j < w * h)
            j++;
        max[i] = abs(*(img->image + j));
    }
    else max[i] = NA;
}

for(i=0; i<w*h; ++i)
{
    if(max[region1] != NA && *(img->region + i) == region1)
    {
        if(abs(*(img->image + i)) > max[region1])
            max[region1] = abs(*(img->image + i));
    } else if(max[region2] != NA && *(img->region + i) == region2)
    {
        if(abs(*(img->image + i)) > max[region2])
            max[region2] = abs(*(img->image + i));
    } else if(max[region3] != NA && *(img->region + i) == region3)
    {
        if(abs(*(img->image + i)) > max[region3])
            max[region3] = abs(*(img->image + i));
    } else if(max[region4] != NA && *(img->region + i) == region4)
    {
        if(abs(*(img->image + i)) > max[region4])
            max[region4] = abs(*(img->image + i));
    } else if(max[region5] != NA && *(img->region + i) == region5)
    {
        if(abs(*(img->image + i)) > max[region5])
            max[region5] = abs(*(img->image + i));
    }
}

return(max);
}

float *get_min_5(img, w, h, region_count_5)
imageP img;
int w, h;
int *region_count_5;
{
    float *min;
    int i, j;

    min = (float *)malloc(sizeof(float) * 5);

    for (i=region1; i<=region5; ++i)
    {
        if (region_count_5[i] > 0)
        {
            j = 0;
            while( *(img->region + j) != i && j < w * h)
                j++;
            min[i] = abs(*(img->image + j));
        }
        else min[i] = NA;
    }

    for(i=0; i<w*h; ++i)
    {
        if(min[region1] != NA && *(img->region + i) == region1)
        {
            if(abs(*(img->image + i)) < min[region1])
                min[region1] = abs(*(img->image + i));
        }
        else if(min[region2] != NA && *(img->region + i) == region2)
        {
            if(abs(*(img->image + i)) < min[region2])
                min[region2] = abs(*(img->image + i));
        }
        else if(min[region3] != NA && *(img->region + i) == region3)
        {
            if(abs(*(img->image + i)) < min[region3])
                min[region3] = abs(*(img->image + i));
        }
        else if(min[region4] != NA && *(img->region + i) == region4)
        {
            if(abs(*(img->image + i)) < min[region4])
                min[region4] = abs(*(img->image + i));
        }
        else if(min[region5] != NA && *(img->region + i) == region5)
        {
            if(abs(*(img->image + i)) < min[region5])
                min[region5] = abs(*(img->image + i));
        }
    }
    return(min);
}

float *get_signed_avg_5(img, w, h, region_count_5) imageP img;

135
int w, h;
int *region_count_5;
{
    float *signed_avg;
    int *sum;
    int i;

    signed_avg = (float *)malloc(sizeof(float) * 5);
    sum = (int *)malloc(sizeof(int) * 5);

    for(i=region1; i<=region5; ++i)
        sum[i] = 0;

    for(i=0; i<w*h; ++i)
    {
        if(*(img->region + i) == region1)
            sum[region1] = sum[region1] + *(img->image + i);
        else if(*(img->region + i) == region2)
            sum[region2] = sum[region2] + *(img->image + i);
        else if(*(img->region + i) == region3)
            sum[region3] = sum[region3] + *(img->image + i);
        else if(*(img->region + i) == region4)
            sum[region4] = sum[region4] + *(img->image + i);
        else if(*(img->region + i) == region5)
            sum[region5] = sum[region5] + *(img->image + i);
    }

    for(i=region1; i<=region5; ++i)
    {
        if( region_count_5[i] > 0)
            signed_avg[i] = (float)(sum[i]/region_count_5[i]);
        else signed_avg[i] = NA;
    }

    free(sum);
    return(signed_avg);
}

float *get_unsigned_avg_5(img, w, h, region_count_5)
imageP img;
int w, h;
int *region_count_5;
{
    float *unsigned_avg;
    int *sum;
    int i;

    unsigned_avg = (float *)malloc(sizeof(float) * 5);
    sum = (int *)malloc(sizeof(int) * 5);

    for(i=region1; i<=region5; ++i)
sum[i] = 0;

for(i=0; i<w*h; ++i)
{
    if(*(img->region + i) == region1)
        sum[region1] = sum[region1] + abs(*(img->image + i));
    else if(*(img->region + i) == region2)
        sum[region2] = sum[region2] + abs(*(img->image + i));
    else if(*(img->region + i) == region3)
        sum[region3] = sum[region3] + abs(*(img->image + i));
    else if(*(img->region + i) == region4)
    else if(*(img->region + i) == region5)
        sum[region5] = sum[region5] + abs(*(img->image + i));
}

for(i=region1; i<=region5; ++i)
{
    if( region_count[i] > 0)
        unsigned_avg[i] = (float)(sum[i]/region_count[i]);
    else unsigned_avg[i] = NA;
}

free(sum);
return(unsigned_avg);
}*/

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

/* Program Name: error_interest.c */
/* Function: Getting errors in the region of interest */
/* Implemented by: Yanhong Zhou */
/*******************************************************************************/
#include <stdio.h>
#include "polar_def.h"

static void help();
static void interest();
static void get_error_interest();

main(argc, argv)
int argc;
char **argv;
{
    int ac;
    char **av;
    int i;
    char *progname;

    progname = argv[0];

    for(i=0; i<w*h; ++i)
    {
        if(*(img->region + i) == region1)
            sum[region1] = sum[region1] + abs(*(img->image + i));
        else if(*(img->region + i) == region2)
            sum[region2] = sum[region2] + abs(*(img->image + i));
        else if(*(img->region + i) == region3)
            sum[region3] = sum[region3] + abs(*(img->image + i));
        else if(*(img->region + i) == region4)
        else if(*(img->region + i) == region5)
            sum[region5] = sum[region5] + abs(*(img->image + i));
    }

    for(i=region1; i<=region5; ++i)
    {
        if( region_count[i] > 0)
            unsigned_avg[i] = (float)(sum[i]/region_count[i]);
        else unsigned_avg[i] = NA;
    }

    free(sum);
    return(unsigned_avg);
}*/
if (argc != 5 )
    help(progname);

ac = argc;
av = (char **)malloc(ac * sizeof(char *));

for(i=0; i<argc; i++)
    av[i] = argv[i];

interest(ac, av);

exit(positive);
}

static void help()
{
    printf("usage:\n");
    printf("interest start_x, start_y, end_x, end_y\n");
}

static void interest(ac, av)
int ac;
char **av;
{
    int start_x = atoi(av[1]);
    int start_y = atoi(av[2]);
    int end_x = atoi(av[3]);
    int end_y = atoi(av[4]);

    imageP img1_imax;
    imageP img1_imin;
    imageP img1_avg;
    imageP img1_pp;
    imageP img1_e;
    imageP img1_t;

    imageP img2_imax;
    imageP img2_imin;
    imageP img2_avg;
    imageP img2_pp;
    imageP img2_e;
    imageP img2_t;

    imageP delta_pp;
    imageP delta_i;
    imageP delta_t;

    int w, h, g;
    int i;
int region_count;
int *region_avg;

region_avg = (int *)malloc(sizeof(int) * 6);

img1_imax = readimage_interest(in_imax_1, start_x, start_y, end_x, end_y);
img1_imin = readimage_interest(in_imin_1, start_x, start_y, end_x, end_y);
img1_avg = readimage_interest(in_average_1, start_x, start_y, end_x, end_y);
img1_pp = readimage_interest(in_pp_1, start_x, start_y, end_x, end_y);
img1_e = readimage_interest(in_e_1, start_x, start_y, end_x, end_y);
img1_t = readimage_interest(in_t_1, start_x, start_y, end_x, end_y);

img2_imax = readimage_interest(in_imax_2, start_x, start_y, end_x, end_y);
img2_imin = readimage_interest(in_imin_2, start_x, start_y, end_x, end_y);
img2_avg = readimage_interest(in_average_2, start_x, start_y, end_x, end_y);
img2_pp = readimage_interest(in_pp_2, start_x, start_y, end_x, end_y);
img2_e = readimage_interest(in_e_2, start_x, start_y, end_x, end_y);
img2_t = readimage_interest(in_t_2, start_x, start_y, end_x, end_y);

w = img1_imax->width;
h = img1_imax->height;
g = img1_imax->grey;

writeimage("fitting_interest", img1_pp, w, h, g);

delta_pp = (imageP)malloc(sizeof(imageS));
delta_pp->image = (int *)malloc(sizeof(int) * w * h);
delta_pp->region = (int *)malloc(sizeof(int) * w * h);
delta_pp->width = w;
delta_pp->height = h;
delta_pp->grey = g;

delta_i = (imageP)malloc(sizeof(imageS));
delta_i->image = (int *)malloc(sizeof(int) * w * h);
delta_i->region = (int *)malloc(sizeof(int) * w * h);
delta_i->width = w;
delta_i->height = h;
delta_i->grey = g;

delta_t = (imageP)malloc(sizeof(imageS));
delta_t->image = (int *)malloc(sizeof(int) * w * h);
delta_t->region = (int *)malloc(sizeof(int) * w * h);
delta_t->width = w;
delta_t->height = h;
delta_t->grey = g;

generate_error_interest(img1_imax, img1_imin, img1_avg, img1_pp, img1_e, img1_t,
                        img2_imax, img2_imin, img2_avg, img2_pp, img2_e, img2_t,
                        delta_pp, delta_i, delta_t, w, h, region_count, region_avg);

for(i=0; i<w*h; ++i)
{
  *(delta_pp->image + i) = abs(*(delta_pp->image + i));
  *(delta_i->image + i) = abs(*(delta_i->image + i));
  *(delta_t->image + i) = abs(*(delta_t->image + i));
}

writeimage("fitting-interest-pp", delta_pp, w, h, g);
writeimage("fitting-interest-i", delta_i, w, h, g);
writeimage("fitting-interest-t", delta_t, w, h, g);

freeimage(img1_imax);
freeimage(img1_imin);
freeimage(img1_avg);
freeimage(img1_pp);
freeimage(img1_e);
freeimage(img1_t);
freeimage(img2_imax);
freeimage(img2_imin);
freeimage(img2_avg);
freeimage(img2_pp);
freeimage(img2_e);
freeimage(img2_t);
freeimage(delta_pp);
freeimage(delta_i);
freeimage(delta_t);
free(image_avg);
}

static void get_error_interest(img1_imax, img1_imin, img1_avg, img1_pp, img1_e, img1_t,
                              img2_imax, img2_imin, img2_avg, img2_pp, img2_e, img2_t,
                              delta_pp, delta_i, delta_t, w, h, region_count, region_avg)

{ int i, j;
  FILE *fp;
}
float *error_pp;
float *error_i;
float *error_t;

error_pp = (float *)malloc(sizeof(float) * 5);
error_i = (float *)malloc(sizeof(float) * 5);
error_t = (float *)malloc(sizeof(float) * 5);

for(i = 0; i < 6; ++i)
    region_avg[i] = 0;

for(i=0; i<w*h; ++i)
{
    *(delta_pp->image + i) = *(img1_pp->image + i) - *(img2_pp->image + i);
    region_avg[0] += *(img1_pp->image + i);
    region_avg[1] += *(img2_pp->image + i);
    *(delta_i->image + i) = *(img1_imax->image + i) - *(img1_imin->image + i) -
                          *(img2_imax->image + i) - *(img2_imin->image + i);
    region_avg[2] += *(img1_imax->image + i) - *(img1_imin->image + i);
    region_avg[3] += *(img2_imax->image + i) - *(img2_imin->image + i);
    *(delta_t->image + i) = 0.5 * get_delta_t(*(img1_t->image + i),*(img2_t->image + i));
    region_avg[4] += *(img1_t->image + i);
    region_avg[5] += *(img2_t->image + i);
}

region_count = w*h;

for(i = 0; i <= 5; ++i)
    region_avg[i] = (int)(region_avg[i]/region_count);

error_pp[0] = get_e_interest(delta_pp, w, h);
error_pp[1] = get_max_interest(delta_pp, w, h);
error_pp[2] = get_min_interest(delta_pp, w, h);
error_pp[3] = get_signed_avg_interest(delta_pp, w, h);
error_pp[4] = get_unsigned_avg_interest(delta_pp, w, h);

error_i[0] = get_e_interest(delta_i, w, h);
error_i[1] = get_max_interest(delta_i, w, h);
error_i[2] = get_min_interest(delta_i, w, h);
error_i[3] = get_signed_avg_interest(delta_i, w, h);
error_i[4] = get_unsigned_avg_interest(delta_i, w, h);

error_t[0] = get_e_interest(delta_t, w, h);
error_t[1] = get_max_interest(delta_t, w, h);
error_t[2] = get_min_interest(delta_t, w, h);
error_t[3] = get_signed_avg_interest(delta_t, w, h);
error_t[4] = get_unsigned_avg_interest(delta_t, w, h);

fp = fopen(error_file_interest, "w");
printf(fp, "error of pp:\n");
for(i=0; i<5; ++i)
    fprintf(fp, "%.9f", error_pp[i]);
fprintf(fp, "%.10d", region_avg[0]);
fprintf(fp, "%.10d", region_avg[1]);
fprintf(fp, "%.10d", region_count);
fprintf(fp, "\n");
fprintf(fp, "\n");

for(i=0; i<5; ++i)
    fprintf(fp, "%.9f", error_i[i]);
fprintf(fp, "%.10d", region_avg[2]);
fprintf(fp, "%.10d", region_avg[3]);
fprintf(fp, "%.10d", region_avg[4]);
fprintf(fp, "%.10d", region_avg[5]);
fprintf(fp, "\n");
fprintf(fp, "\n");

fclose(fp);

free(error_pp);
free(error_i);
free(error_t);

feel_gradient.c

Getting errors in regions of gradient of polarization degree

Yanhong Zhou

#include <stdio.h>
#include "polar_def.h"

static void help();
static void gradient();
static void get_region();
static void get_error();

main(argc, argv)
int argc;
char **argv;
{ int ac; char **av; int i; char *proname;

proname = argv[0];

if (argc != 8 )
{
    help(proname);
    exit(positive);
}

ac = argc;
av = (char **)malloc(ac * sizeof(char *));

for(i=0; i<argc; i++)
    av[i] = argv[i];

gradient(ac, av);

free(av);
exit(positive);
}

static void help()
{
    printf("usage:
    printf("gradient filename(used for getting gradient) T1 T2
        start_x, start_y, end_x, end_y\n\n    ");
}

static void gradient(ac, av)
int ac;
char **av;
{
    char *gradient_file = av[1];
    int t_high = atoi(av[2]);
    int t_low = atoi(av[3]);
    int start_x = atoi(av[4]);
    int start_y = atoi(av[5]);
    int end_x = atoi(av[6]);
    int end_y = atoi(av[7]);

    imageP img1_imax;
    imageP img1_imin;
    imageP img1_avg;
    imageP img1_pp;
}
imageP img1_e;
imageP img1_t;

imageP img2_imax;
imageP img2_imin;
imageP img2_avg;
imageP img2_pp;
imageP img2_e;
imageP img2_t;

imageP img_gradient;
imageP img_gradient_out;

imageP delta_pp;
imageP delta_i;
imageP delta_t;

int *region_count;
int *region_avg;

int w, h, g;

img1_imax = readimage_interest(in_imax_1, start_x, start_y, end_x, end_y);
img1_imin = readimage_interest(in_imin_1, start_x, start_y, end_x, end_y);
img1_avg = readimage_interest(in_average_1, start_x, start_y, end_x, end_y);
img1_pp = readimage_interest(in_pp_1, start_x, start_y, end_x, end_y);
img1_e = readimage_interest(in_e_1, start_x, start_y, end_x, end_y);
img1_t = readimage_interest(in_t_1, start_x, start_y, end_x, end_y);

img2_imax = readimage_interest(in_imax_2, start_x, start_y, end_x, end_y);
img2_imin = readimage_interest(in_imin_2, start_x, start_y, end_x, end_y);
img2_avg = readimage_interest(in_average_2, start_x, start_y, end_x, end_y);
img2_pp = readimage_interest(in_pp_2, start_x, start_y, end_x, end_y);
img2_e = readimage_interest(in_e_2, start_x, start_y, end_x, end_y);
img2_t = readimage_interest(in_t_2, start_x, start_y, end_x, end_y);

img_gradient = readimage_interest(gradienLfile, start_x, start_y, end_x, end_y);

w = img1_imax->width;

h = img1_imax->height;
g = img1_imax->grey;

img_gradient_out = (imageP)malloc(sizeof(imageS));
img_gradient_out->image = (int *)malloc(sizeof(int) * w * h);
img_gradient_out->region = (int *)malloc(sizeof(int) * w * h);
img_gradient_out->width = w;
img_gradient_out->height = h;
img_gradient_out->grey = g;

delta_pp = (imageP)malloc(sizeof(imageS));
delta_pp->image = (int *)malloc(sizeof(int) * w * h);
delta_pp->region = (int *)malloc(sizeof(int) * w * h);
delta_pp->width = w;
delta_pp->height = h;
delta_pp->grey = g;

delta_i = (imageP)malloc(sizeof(imageS));
delta_i->image = (int *)malloc(sizeof(int) * w * h);
delta_i->region = (int *)malloc(sizeof(int) * w * h);
delta_i->width = w;
delta_i->height = h;
delta_i->grey = g;

delta_t = (imageP)malloc(sizeof(imageS));
delta_t->image = (int *)malloc(sizeof(int) * w * h);
delta_t->region = (int *)malloc(sizeof(int) * w * h);
delta_t->width = w;
delta_t->height = h;
delta_t->grey = g;

region_count = (int *)malloc(sizeof(int) * 5);
region_avg = (int *)malloc(sizeof(int) * 30);

get_region(img1_imax, img1_imin, img1_avg, img1_pp, img1_e, img1_t,
           img2_imax, img2_imin, img2_avg, img2_pp, img2_e, img2_t,
           img_gradient, img_gradient_out,
delta_pp, delta_i, delta_t, region_count, region_avg,
t_high, t_low);

get_error(img1_imax, img1_imin, img1_avg, img1_pp, img1_e, img1_t,
          img2_imax, img2_imin, img2_avg, img2_pp, img2_e, img2_t,
          delta_pp, delta_i, delta_t, region_count, region_avg);

writeimage(writeimage_gradient, img_gradient_out, w, h, g);

freeimage(img1_imax);
freeimage(img1_imin);
freeimage(img1_avg);
freeimage(img1_pp);
freeimage(img1_e);
freeimage(img1_t);
freeimage(img2_imax);
freeimage(img2_imin);
freeimage(img2_avg);
freeimage(img2_pp);
freeimage(img2_e);
freeimage(img2_t);
freeimage(img_gradient);
freeimage(img_gradient_out);
freeimage(delta_pp);
freeimage(delta_i);
freeimage(delta_t);
free(region_count);
free(region_avg);
static void get_region(img1_imax, img1_imin, img1_avg, img1_pp, img1_e, img1_t, 
   img2_imax, img2_imin, img2_avg, img2_pp, img2_e, img2_t, 
   img_gradient, img_gradient_out, 
   delta_pp, delta_i, delta_t, region_count, region_avg,  
   t_high, t_low)

imageP img1_imax;  
imageP img1_imin;  
imageP img1_avg;  
imageP img1_pp;  
imageP img1_e;  
imageP img1_t;  
imageP img2_imax;  
imageP img2_imin;  
imageP img2_avg;  
imageP img2_pp;  
imageP img2_e;  
imageP img2_t;  
imageP img_gradient;  
imageP img_gradient_out;  
imageP delta_pp;  
imageP delta_i;  
imageP delta_t;  
int *region_count;  
int *region_avg;  
int t_high, t_low;
{
    int i, j;
    int w, h, g;
    int *index;

    w = img1_imax->width;
    h = img1_imax->height;
    g = img1_imax->grey;

    index = (int *)malloc(sizeof(int) * w * h);

    for(i=0; i<=4; ++i)
        region_count[i] = 0;

    for(i=0; i<30; ++i)
        region_avg[i] = 0;

    for(i=0; i<w*h; ++i)
    {
        if(i == 0)
            *(img_gradient_out->image + i) = get_gradient(*(img_gradient->image + i + 1),
            *(img_gradient->image + i + w));
        else if(i == w-1)
            *(img_gradient_out->image + i) = get_gradient(*(img_gradient->image + i - 1),
            *(img_gradient->image + i + w));
    }
else if (i == (h-1) * w)
    *(img_gradient_out->image + i) = get_gradient(*(img_gradient->image + i + 1),
    *(img_gradient->image + i + w));
else if (i == (w*h - 1))
    *(img_gradient_out->image + i) = get_gradient(*(img_gradient->image + i - 1),
    *(img_gradient->image + i - w));
else if (i >= 1 && i < (w-1))
    *(img_gradient_out->image + i) = get_gradient( ((*(img_gradient->image + i - 1) -
    *(img_gradient->image + i + 1)),
    *(img_gradient->image + i + w));
else if ((i % w) == 0)
    *(img_gradient_out->image + i) = get_gradient(*(img_gradient->image + i + 1),
    (*(img_gradient->image + i + w) -
    *(img_gradient->image + i + w));
else if ((i % w) == (w-1))
    *(img_gradient_out->image + i) = get_gradient( ((*(img_gradient->image + i - 1) -
    *(img_gradient->image + i + 1)),
    *(img_gradient->image + i + w));
else if (i >= w * (h-1) && i < (w*h - 1))
    *(img_gradient_out->image + i) = get_gradient( ((*(img_gradient->image + i - 1) -
    *(img_gradient->image + i + 1)),
    *(img_gradient->image + i - w));
else
    *(img_gradient_out->image + i) = get_gradient( ((*(img_gradient->image + i - 1) -
    *(img_gradient->image + i + 1)),
    *(img_gradient->image + i - w));
}

for(i=0; i<w*h; ++i)
{
    if (*(img_gradient_out->image + i) >= t_high)
    {
        index[i] = region5;
        region_count[region5]++;
        region_avg[region5 * 6 + 0] = region_avg[region5 * 6 + 0] + *(img1_pp->image + i);
        region_avg[region5 * 6 + 1] = region_avg[region5 * 6 + 1] + *(img2_pp->image + i);
        region_avg[region5 * 6 + 2] = region_avg[region5 * 6 + 2] + *(img1_imax->image + i) -
        *(img1_imin->image + i);
        region_avg[region5 * 6 + 3] = region_avg[region5 * 6 + 3] + *(img2_imax->image + i) -
        *(img2_imin->image + i);
        region_avg[region5 * 6 + 4] = region_avg[region5 * 6 + 4] + *(img1_t->image + i);
        region_avg[region5 * 6 + 5] = region_avg[region5 * 6 + 5] + *(img1_t->image + i);
    }
    else if (*(img_gradient_out->image + i) < t_low)
    {
        if(*(img1_pp->image + i) >= T_pp_high)
        {
            index[i] = region1;
            region_count[region1]++;
            region_avg[region1 * 6 + 0] = region_avg[region1 * 6 + 0] + *(img1_pp->image + i);
        
        

```
```c
region_avg[region1 * 6 + 1] = region_avg[region1 * 6 + 1] + *(img2_pp->image + i);
region_avg[region1 * 6 + 2] = region_avg[region1 * 6 + 2] + *(img2_imin->image + i) - *(img1_imin->image + i);
region_avg[region1 * 6 + 3] = region_avg[region1 * 6 + 3] + *(img2_imax->image + i) - *(img1_imax->image + i);
}
else if(*(img1_pp->image + i) < T_pp_high && *(img1_pp->image + i) >= T_pp_low)
{
    index[i] = region2;
    region_count[region2]++;
    region_avg[region2 * 6 + 0] = region_avg[region2 * 6 + 0] + *(img1_pp->image + i);
    region_avg[region2 * 6 + 1] = region_avg[region2 * 6 + 1] + *(img2_pp->image + i);
    region_avg[region2 * 6 + 2] = region_avg[region2 * 6 + 2] + *(img1_imax->image + i) - *(img1_imin->image + i);
    region_avg[region2 * 6 + 3] = region_avg[region2 * 6 + 3] + *(img2_imax->image + i) - *(img2_imin->image + i);
    region_avg[region2 * 6 + 4] = region_avg[region2 * 6 + 4] + *(img1_t->image + i);
    region_avg[region2 * 6 + 5] = region_avg[region2 * 6 + 5] + *(img1_t->image + i);
}
else if(*(img1_pp->image + i) < T_pp_low)
{
    index[i] = region3;
    region_count[region3]++;
    region_avg[region3 * 6 + 0] = region_avg[region3 * 6 + 0] + *(img1_pp->image + i);
    region_avg[region3 * 6 + 1] = region_avg[region3 * 6 + 1] + *(img2_pp->image + i);
    region_avg[region3 * 6 + 2] = region_avg[region3 * 6 + 2] + *(img1_imax->image + i) - *(img1_imin->image + i);
    region_avg[region3 * 6 + 3] = region_avg[region3 * 6 + 3] + *(img2_imax->image + i) - *(img2_imin->image + i);
    region_avg[region3 * 6 + 4] = region_avg[region3 * 6 + 4] + *(img1_t->image + i);
    region_avg[region3 * 6 + 5] = region_avg[region3 * 6 + 5] + *(img1_t->image + i);
}
else
{
    index[i] = region4;
    region_count[region4]++;
    region_avg[region4 * 6 + 0] = region_avg[region4 * 6 + 0] + *(img1_pp->image + i);
    region_avg[region4 * 6 + 1] = region_avg[region4 * 6 + 1] + *(img2_pp->image + i);
    region_avg[region4 * 6 + 2] = region_avg[region4 * 6 + 2] + *(img1_imax->image + i) - *(img1_imin->image + i);
    region_avg[region4 * 6 + 3] = region_avg[region4 * 6 + 3] + *(img2_imax->image + i) - *(img2_imin->image + i);
    region_avg[region4 * 6 + 4] = region_avg[region4 * 6 + 4] + *(img1_t->image + i);
    region_avg[region4 * 6 + 5] = region_avg[region4 * 6 + 5] + *(img1_t->image + i);
}
}
for(i=0; i<w*h; ++i)
{
```
}
*(delta_pp->region + i) = index[i];
*(delta_i->region + i) = index[i];
*(delta_t->region + i) = index[i];
}

for(i = region1; i <= region5; ++i)
for(j = 0; j < 6; ++j)
{
    if(j == 4 || j == 5)
        region_avg[i * 6 + j] = (int)(0.5*(region_avg[i * 6 + j] / region_count[i]));
    else
        region_avg[i * 6 + j] = (int)(region_avg[i * 6 + j] / region_count[i]);
}

static void get_error(img1_imax, img1_imin, img1_avg, img1_pp, img1_e, img1_t,
                      img2_imax, img2_imin, img2_avg, img2_pp, img2_e, img2_t,
                      delta_pp, delta_i, delta_t, region_count, region_avg)

imageP img1_imax;
imageP img1_imin;
imageP img1_avg;
imageP img1_pp;
imageP img1_e;
imageP img1_t;
imageP img2_imax;
imageP img2_imin;
imageP img2_avg;
imageP img2_pp;
imageP img2_e;
imageP img2_t;
imageP delta_pp;
imageP delta_i;
imageP delta_t;
int *region_count;
int *region_avg;
{
    int i, j;
    int w, h, g;
    FILE *fp;
    imageP di_1, di_2;

    float **error_pp;
    float **error_i;
    float **error_t;

    float pp_1, pp_2, i_1, i_2, t_1, t_2;

    error_pp = (float **)malloc(sizeof(float) * 15);
    error_i = (float **)malloc(sizeof(float) * 15);
    error_t = (float **)malloc(sizeof(float) * 15);
w = img1_imax->width;
h = img1_imax->height;
g = img1_imax->grey;

di_1 = (imageP)malloc(sizeof(imageS));
di_1->image = (int *)malloc(sizeof(int) * w * h);
di_1->region = (int *)malloc(sizeof(int) * w * h);
di_1->width = w;
di_1->height = h;
di_1->grey = g;

di_2 = (imageP)malloc(sizeof(imageS));
di_2->image = (int *)malloc(sizeof(int) * w * h);
di_2->region = (int *)malloc(sizeof(int) * w * h);
di_2->width = w;
di_2->height = h;
di_2->grey = g;

for(i=0; i<w*h; ++i)
{
    *(delta_pp->image + i) = *(img1_pp->image + i) - *(img2_pp->image + i);
    *(delta_i->image + i) = (**(img1_imax->image + i) - *(img1_imin->image + i)) - (**(img2_imax->image + i) - *(img2_imin->image + i));
    *(delta_t->image + i) = 0.5 * get_delta_t(**(img1_t->image + i),**(img2_t->image + i));
    *(di_1->image + i) = abs(**(img1_imax->image + i) - *(img1_imin->image + i));
    *(di_2->image + i) = abs(**(img2_imax->image + i) - *(img2_imin->image + i));
}

error_pp[0] = get_e(delta_pp, w, h, region_count, 5);
error_pp[1] = get_max(delta_pp, w, h, region_count, 5);
error_pp[2] = get_min(delta_pp, w, h, region_count, 5);
error_pp[3] = get_signed_avg(delta_pp, w, h, region_count, 5);
error_pp[4] = get_unsigned_avg(delta_pp, w, h, region_count, 5);

error_i[0] = get_e(delta_i, w, h, region_count, 5);
error_i[1] = get_max(delta_i, w, h, region_count, 5);
error_i[2] = get_min(delta_i, w, h, region_count, 5);
error_i[3] = get_signed_avg(delta_i, w, h, region_count, 5);
error_i[4] = get_unsigned_avg(delta_i, w, h, region_count, 5);

error_t[0] = get_e(delta_t, w, h, region_count, 5);
error_t[1] = get_max(delta_t, w, h, region_count, 5);
error_t[2] = get_min(delta_t, w, h, region_count, 5);
error_t[3] = get_signed_avg(delta_t, w, h, region_count, 5);
error_t[4] = get_unsigned_avg(delta_t, w, h, region_count, 5);

pp_1 = get_image_avg(img1_pp);
pp_2 = get_image_avg(img2_pp);
i_1 = get_image_avg(di_1);
i_2 = get_image_avg(di_2);
t_1 = get_image_avg(img1_t);
t_2 = get_image_avg(img2_t);
fp = fopen(error_file_gradient, "w");
fprintf(fp, "region1: gradient < T1, pp >= T_pp_high;\n");
fprintf(fp, "region2: gradient < T1, T_pp_low <= pp < T_pp_high;\n");
fprintf(fp, "region3: gradient < T1, pp < T_pp_low;\n");
fprintf(fp, "region4: T1 <= gradient < T2;\n");
fprintf(fp, "region5: gradient >= T2;\n");

fprintf(fp, "error of pp;\n");
fprintf(fp, "rms max min signed_avg unsigned_avg avg_1 avg_2 number\n");
fprintf(fp, "\n");
for(j=region1; j<=region5; ++j)
{
    fprintf(fp, "region%d:", j+1);
    for(i=0; i<5; ++i)
        fprintf(fp, "%.4f", error_pp[i][j]);
    fprintf(fp, "%10d", region_avg[j * 6 + 0]);
    fprintf(fp, "%10d", region_avg[j * 6 + 1]);
    fprintf(fp, "%10d", region_count[j]);
    fprintf(fp, "\n");
}
fprintf(fp, "\n");

fprintf(fp, "error of I: max - Imin;\n");
fprintf(fp, "rms max min signed_avg unsigned_avg avg_1 avg_2 number\n");
fprintf(fp, "\n");
for(j=region1; j<=region5; ++j)
{
    fprintf(fp, "region%d:", j+1);
    for(i=0; i<5; ++i)
        fprintf(fp, "%.4f", error_i[i][j]);
    fprintf(fp, "%10d", region_avg[j * 6 + 2]);
    fprintf(fp, "%10d", region_avg[j * 6 + 3]);
    fprintf(fp, "%10d", region_count[j]);
    fprintf(fp, "\n");
}
fprintf(fp, "\n");

fprintf(fp, "error of t;\n");
fprintf(fp, "rms max min signed_avg unsigned_avg avg_1 avg_2 number\n");
fprintf(fp, "\n");
for(j=region1; j<=region5; ++j)
{
    fprintf(fp, "region%d:", j+1);
    for(i=0; i<5; ++i)
        fprintf(fp, "%.4f", error_t[i][j]);
    fprintf(fp, "%10d", region_avg[j * 6 + 4]);
    fprintf(fp, "%10d", region_avg[j * 6 + 5]);
    fprintf(fp, "%10d", region_count[j]);
    fprintf(fp, "\n");
}
fprintf(fp, "n");

close(fp);

free(error_pp);
free(error_i);
free(error_t);

#include <sys/types.h>
#include <unistd.h>
#include <stdio.h>
#include <math.h>
#include <malloc.h>
#include <stdlib.h>

typedef struct {
    int width;    /* number of columns in the image */
    int height;   /* number of rows in the image */
    int grey;     /* number of gray levels in the image */
    int *image;   /* pointer to the image data */
    int *region;  /* pointer to the region index */
} imageS, *imageP;

typedef struct {
    int *value;
    int *region;
} deltaS, *deltaP;

/* definition of constants */
#define pi 3.1415926
#define black 0
#define white 255
#define positive 1
#define negative 1
#define ONE 1
#define ZERO 0
#define NO -888888

/* definition of file names */
#define in_imax_1 "radio-nowarp-Imax"
#define in_imin_1 "radio-nowarp-Imin"
#define in_average_1 "radio-nowarp-Iavg"
#define in_pp_1 "radio-nowarp-pp"
#define in_e_1 "radio-nowarp-e"
#define in_L_1 "radio-nowarp-l"
#define in_imax_2 "radio-warp-Imax"
#define in_imin_2 "radio-warp-Imin"
#define in_average_2 "radio-warp-Iavg"
#define in_pp_2 "radio-warp-pp"
#define in_e_2 "radio-warp-e"
#define in_L2 "radio-warp-t"

#define error_fife_tO "radio_tO.txt"
#define error_fife_interest "radio_interest.txt"
#define error_fife_gradient "radio_gradient.txt"
#define writeimage_gradient "radio_gradient.img"

/** definition of thresholds */
#define T_pp_high 35
#define T_pp_low 10
#define T_i_high 40
#define T_i_low 15
#define T_e 20

/** definition of regions */
#define region1 0
#define region2 1
#define region3 2
#define region4 3
#define region5 4
#define region6 5
#define region7 6
#define region8 7
#define region9 8
#define region10 9

/** extern functions */
extern imageP readimage();
extern imageP readimage_interest();
extern void writeimage();
extern void freeimage();
extern int get_delta_t();
extern float *get_e();
extern float get_image_avg();
extern float *get_max();
extern float *get_min();
extern float *get_signed_avg();
extern float *get_unsigned_avg();
extern float get_e_interest();
extern float get_max_interest();
extern float get_min_interest();
extern float get_signed_avg_interest();
extern float get_unsigned_avg_interest();
/** Program Name: error_def.h **
* Function: Some basic functions used by programs of getting errors *
* Implemented by: Yanhong Zhou *
**

#include "polar_def.h"

imageP readimage(filename)
char *filename;
{
    imageP img;
    int w, h, g;
    int i;
    FILE *fd;
    char junk[2];

    fd = fopen(filename, "r");

    img = (imageP)malloc(sizeof(imageS));

    fscanf(fd, "%s", junk);
    fscanf(fd, "%d", &(img->width));
    fscanf(fd, "%d", &(img->height));
    fscanf(fd, "%d", &(img->grey));

    w = img->width;
    h = img->height;
    g = img->grey;

    img->image = (int *)malloc(w*h*sizeof(int));
    img->region = (int *)malloc(w*h*sizeof(int));

    for(i=0; kw*h; ++i)
    {
        fscanf(fd, "%d", (img->image+i));
        *(img->region + i) = 99;
    }

    fclose(fd);
    return(img);
}

imageP readimage_interest(filename, start_x, start_y, end_x, end_y)
char *filename;
int start_x, start_y, end_x, end_y;
{
    imageP img;
    int w, h, g;
    int i, j;
    int count;

    for(i=0; i<w; ++i)
    {
        fscanf(fd, "%d", (img->image+i));
        *(img->region + i) = 99;
    }

    fclose(fd);
    return(img);
}
FILE *fd;
int junkw, junkh, junkimage;
char junk[3];

fd = fopen(filename, "r");

img = (imageP)malloc(sizeof(imageS));

fscanf(fd, "%s", junk);
fscanf(fd, "%d", &junkw);
fscanf(fd, "%d", &junkh);
fscanf(fd, "%d", &img->grey);

w = end_x - start_x + 1;
h = end_y - start_y + 1;

count = 0;

img->image = (int *)malloc(w*h*sizeof(int));
img->region = (int *)malloc(w*h*sizeof(int));

img->width = w;
img->height = h;

for(i=0; i < (start_y - 1) * junkw; ++i)
  fscanf(fd, "%d", &junkimage);
for(i=start_y; j<=end_y; ++i)
  for(j=start_x; j<start_x; ++j)
    fscanf(fd, "%d", &junkimage);
  for(j=start_x; j<=end_x; ++j)
    {fscanf(fd, "%d", (img->image+count));
    *(img->region + count) = 99;
    count++;
    }
  for(j=end_x+1; j<junkw; ++j)
    fscanf(fd, "%d", &junkimage);

fclose(fd);
return(img);
}

void writeimage(filename, img, w, h, g)
char *filename;
imageP img;
int w, h, g;
{
  int i;
  FILE *fd;
fd = fopen(filename, "w");

fprintf(fd, "%s\n", "P2");
fprintf(fd, "%d %d\n", w, h);
fprintf(fd, "%d\n", g);

for(i=0; i<w*h; ++i)
{
    fprintf(fd, "%%d \n", *(img->image+i));
    if( (i+1) % 17 == 0 ) fprintf(fd, "In");
}
fclose(fd);

void freeimage(img)
imageP img;
{
    free(img->image);
    free(img->region);
    free(img);
}

float get_gradient(x, y)
int x, y;
{
    float gradient;

    gradient = sqrt(x * x + y * y);

    return(gradient);
}

******************************************************************************
* Program Name:    lut.c
* Function:        Generating look-up table for intensity linearization
* Implemented by:  Yanhong Zhou
******************************************************************************
#include <stdio.h>
#include <math.h>
#define PI 3.1415926535897932
#define ConsL 0.9
#define ConsL2 0.591
#define ConsL3 0.361
#define ConsL4 0.198
#define ConsL5 0.09
#define ConsL6 0.031

int read_in_data(MAX, data, in_file)
int MAX;
double data[256];
FILE *in_file;
{
    int data_count[256];
    double const_value[7];
    double sin_2_alpha[6];
    int i;
    int x;
    int y;

    for (i = 0; i < 256; i++) {
        data_count[i] = 0;
        data[i] = 0;
    }

    const_value[0] = Const_1;
    const_value[1] = 1;
    const_value[2] = Const_2;
    const_value[3] = Const_3;
    const_value[4] = Const_4;
    const_value[5] = Const_5;
    const_value[6] = Const_6;

    sin_2_alpha[0] = 1;
    for (i = 1; i < 6; i++) {
        double alpha;
        int alpha_int;

        fscanf(in_file, "%d", &alpha_int);
        alpha = (double)alpha_int;
        alpha = alpha / 180 * PI;
        sin_2_alpha[i] = sin(alpha) * sin(alpha);
    }

    for (x = 0; x < 6; x++) {
        double MO;

        MO = MAX / 0.9 * sin_2_alpha[x];
        for (y = 0; y < 6; y++) {
            double value;
            int index;

            fscanf(in_file, "%d", &index);
            if (index < 0 || index > 255) {
                return(1);
            }
            value = MO * const_value[y + 1];

            value = (data[index] * data_count[index] + value) / (data_count[index] + 1);
            data_count[index]++;
            data[index] = value;
        }
    }
}
return(0);
}

void linear_convert(data)
double data[256];
{
    int index;
    int index_start;
    int index_end;
    double low;
    double high;
    double step;

    data[0] = 0;
    data[255] = 255;

    index = 0;
    index_end = 0;
    while (index < 255 ){
        index_start = index_end;
        index_end = index_start + 1;
        while ( ((data[index_end] < data[index_start]) || (data[index_end] == 0 )){
            index_end++;
        }
        step = (data[index_end] - data[index_start] ) / (index_end - index_start);
        index = index_start + 1;
        while ( index < index_end ){
            data[index] = data[index - 1 ] + step;
            index ++;
        }
    }
}

void output_data(data, out_file)
double data[256];
FILE *out_file;
{
    int index;

    for( index = 0; index < 256; index ++){
        fprintf(out_file,"%d\n", (int)( data[index] + 0.5 ) );
    }
}

void make_convert_table(MAX, in_file, out_file)
int MAX;
FILE *in_file;
FILE *out_file;
{
    double data[256];

    if ( read_in_data(MAX, data, in_file) == 0 ){
printf(" Data out of range, should be between 0 - 255 ");
exit(2);
}

linear_convert(data);
output_data(data, out_file);
}

void main(argc,argv)
int argc;
char **argv;
{
    FILE *in_file;
    FILE *out_file;
    int MAX;
    int index;

    if (argc != 4){
        printf(" Wrong format
");
        printf(" radio MAX input_filename output_filename \n");
        exit(1);
    }

    in_file  = fopen(argv[2],"r");
    out_file = fopen(argv[3],"w+");

    index = 0;
    MAX = 0;
    while (argv[1][index]){
        MAX = MAX * 10 + (argv[1][index] - '0');
        index++;
    }

    make_convert_table(MAX, in_file, out_file);

    fclose(in_file);
    fclose(out_file);
}

/**************************************************************************************************************
* Program Name: radio_calib
* Function: Linearizing image intensity
* Implemented by: Yanhong Zhou
**************************************************************************************************************

#include <stdio.h>
#include <sys/types.h>
#include <stdlib.h>
#include "def.h"

static void radio_calib();
main(argc, argv)
int argc;
char **argv;
{

int ac;
char **av;
int i;
char *proname;

proname = argv[0];

if (argc < 4)
{
    printf("usage: ");
    printf("radio_calib image lut newimage ");
    exit(1);
}

ac = argc;
av = (char **) malloc(ac * sizeof(char *));
av[0] = argv[0];

for (i = 1; i <= argc; i++)
    av[i] = argv[i];

radio_calib(ac, av);

free(av);
exit(1);
}

static void radio_calib(ac, av)
int ac;
char **av;
{
    char *image = av[1];
    char *input = av[2];
    char *newimage = av[3];
    imageP img;
    imageP newimg;
    FILE *fpin;
    int w, h, g, i;
    int *i;

    img = readimage(image);

    w = img->width;
    h = img->height;
    g = img->grey;
newimg = (imageP)malloc(sizeof(imageS));
newimg->image = (int *)malloc(w*h*sizeof(int));
newimg->mark = (int *)malloc(w*h*sizeof(int));

l = (int *)malloc(sizeof(int) * 256);

fpin = fopen(input, "r");
for(i = 0; i <= 255; ++i)
    fscanf(fpin, "%d", l+i);

for(i=0; i<w*h; ++i)
    *(newimg->image + i) = l[*(img->image + i)];

writeimage(newimage, newimg, w, h, g);

fclose(fpin);
freeimage(img);
freeimage(newimg);
}

/****************************************************************************
 * Program Name ppm2pgm.c *
 * Fuction: Converting image from ppm to pgm *
 * Implemented by: Yanhong Zhou *
 ****************************************************************************/

#include <stdio.h>
#include <sys/types.h>
#include <stdlib.h>
#include <malloc.h>

static void ppm2pgm();

main(argc, argv)
int argc;
char **argv;
{

    int ac;
    char **av;
    int i;
    char *proname;

    proname = argv[0];

    if (argc < 5)
    {
        printf("usage: \n");
        printf("ppm2pgm ppmfile pgmfile-r pgmfile-g pgmfile-b\n");
        exit(1);
    }
ac = argv[0];
for(i=1; i<=argc; i++)
    av[i] = argv[i];

ppm2pgm(ac, av);
free(av);
exit(1);

static void ppm2pgm(ac, av)
int ac;
char **av;
{
    FILE *fpin, *fpout_r, *fpout_g, *fpout_b;
    int w, h, grey, r, g, b, i;
    char p2[2];

    fpin = fopen(av[1], "r");
    fpout_r = fopen(av[2], "w");
    fpout_g = fopen(av[3], "w");
    fpout_b = fopen(av[4], "w");

    fscanf(fpin, "%s", p2);
    fscanf(fpin, "%d %d", &w, &h);
    fscanf(fpin, "%d", &grey);
    fprintf(fpout_r, "P2\n");
    fprintf(fpout_r, "%d %d\n\n", w, h, grey);
    fprintf(fpout_g, "P2\n");
    fprintf(fpout_g, "%d %d
%d\n", w, h, grey);
    fprintf(fpout_b, "P2\n");
    fprintf(fpout_b, "%d %d
%d\n", w, h, grey);

    for(i=0; i<w*h; ++i)
    {
        fscanf(fpin, "%d", &r);
        fprintf(fpout_r, "%d ", r);
        fscanf(fpin, "%d", &g);
        fprintf(fpout_g, "%d ", g);
        fscanf(fpin, "%d", &b);
        fprintf(fpout_b, "%d ", b);
        if( (i+1) % 17 == 0 )
        {
            fprintf(fpout_r, "\n");
            fprintf(fpout_g, "\n");
            fprintf(fpout_b, "\n");
        }
    }
fclose(fpin);
fclose(fpout_r);
fclose(fpout_g);
fclose(fpout_b);
}
Appendix B

Reference


Yanhong Zhou was born in Beijing, China, on August 18, 1969. She received a B.S. from Electrical Engineering Department, Tsinghua University, China.

Yanhong Zhou started her graduate study at EECS Department, Lehigh University from September, 1993. She is currently doing active research in the field of computer vision as a research assistant under Prof. Terrance Boult.
END OF TITLE