Team Number :	97094

Problem Chosen :	A

2019APMCM summary sheet

Abstract

Iron tailings are the waste after ore dressing and the main component of industrial solid waste. Over 10 billion tons of tailings and waste rock are discharged every year in the world. At present, the utilization rate of tailings in China is only 7%, so the comprehensive recovery and utilization of iron tailings has received extensive attention from the whole society.

The main component of iron tailings is silicon dioxide, which is the most difficult component of iron tailings to melt. Due to the poor durability of conventional testing equipment at ultra-high temperatures, the only way to observe the real-time melting rate of silica is to obtain the dynamic visual data of silica in the high-temperature furnace through non-contact. Therefore, the real-time melting rate model of silica based on dynamic visual data is of great guiding significance to the recovery and utilization of iron tailings. Based on the idea of combining KCF target tracking and 3D image SFS modeling, this paper establishes a melting model based on advanced digital image processing technology and measured dynamic visual data, and finally realizes the accurate positioning and melting rate measurement of silica in complex background.

Aiming at problem 1: the original time series images need to be preprocessed before the target tracking of silica. Considering that the video shooting equipment may produce slight jitter, image motion compensation is of great significance. Finally, we choose KCF algorithm for target tracking. When silica exceeds a certain volume, its performance is very good. In the first 90 video images, the error rate was just 1.9%.

Aiming at problem 2: contour extraction of Canny edge detection image can obtain contour information representing 2D image. We selected three obvious features, namely, perimeter, area and minimum circumferential circle radius, to analyze the contour information. In order to eliminate the influence of the image background, the images taken by the crucible with the complete melting of silica were superimposed, and the remaining images were differentiated from the superimposed image to obtain a clear melting image of the background.

Aiming at problem 3:3D modeling was carried out by using SFS algorithm based on 2D grayscale images. The model assumed that the density of silica was uniform and there were no holes, so the volume change diagram and melting rate change diagram of silica were obtained. In order to make the volume measurement more accurate, a weighted calculation was carried out in the specified contour area obtained in problem 2, and the final melt model was established, with an accuracy rate of more than 97.9%.Based on the 3D model, the motion trajectory of the centroid of problem 1 is precisely located.

Key Words: Silicon dioxide, image enhancement , edge sharpening, KCF tracking , SFS modeling

Table of Contents

Melting Representation Model of Silicon Dioxide Analyzed Based on Image

1 Overview

1.1 Background

The main component of iron tailings is silicon dioxide, while the silicon dioxide is the hardest part to melt among the iron tailing components. Therefore, the melting behavior of iron tailing can be represented by the melting behavior of silicon dioxide.

However, the temperature of high-temperature molten tank is over 1,500, in which the service life of routine detection equipment under the environment is very short. To solve the problem, relevant research group firstly adopted a kind of rifted CCD video shooting system with amplification effect for the first time at home and abroad. In the non-contact way, the group has obtained the dynamic visual data of silicon dioxide in the high-temperature molten pool (sequence image under time sequence), and has observed the real-time melting rate of silicon dioxide in the time sequence through video analysis, which provided guidance for the tailing addition and heat compensation in the process of slag cotton preparation, thus indirectly improved the direct fiber forming technology of blast furnace slag.

Time sequential image of silicon dioxide in high-temperature molten pool during the melting process. See the figure below for the situation of adopted experimental equipment:

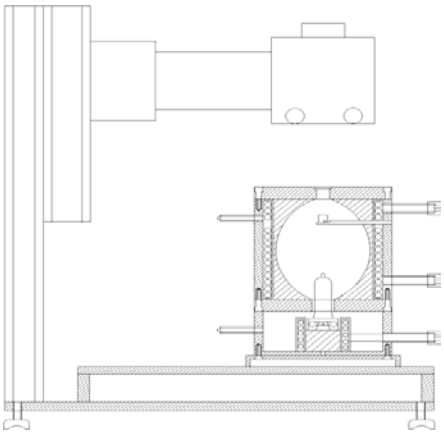

Figure 1-1 Laboratory equipment

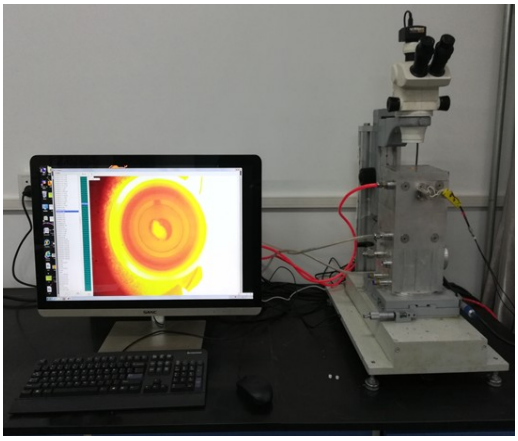

Figure 1-2 Laboratory equipment

1.2 Restatement of the Problem

To reveal the dissolution behavior of iron tailings in blast furnace slag, the main component SiO_2 of iron tailings is adopted for research, researching the melting

process of SiO_2 particles at high temperature, so as to represent the melting of iron tailings. Materials to be prepared for the test are: pure SiO_2 particles for analysis (see Figure 1-3) and corundum crucible with diameter of 8mm (see Figure 1-4) respectively:

Figure 1-3 SiO2

Figure1-4 crucible

(1) The position of silicon dioxide particles in the high-temperature molten pool during the melting process is changing constantly, and the first step to analyze the melting behavior of silicon dioxide is to track the target. Please establish a mathematical model to track the centroid position of silicon dioxide particles during the melting process, and present the motion trail of centroid of silicon dioxide.

(2) Establish indexes which represent the edge outline characteristics of silicon dioxide during the melting process of silicon dioxide (such as shape, perimeter, area, generalized radius, etc.); the participants can select by themselves, as long as they can represent the melting process of silicon dioxide.

(3) For the image given in the attachment is 2D, while the key parameter which represents the melting rate of silicon dioxide is mass, and the mass is in direct proportion to the 3D volume, estimate the actual melting rate of silicon dioxide based on edge outline characteristic indexes of silicon dioxide in the No. 2 question.

2 Model establishment and solution of problem 1

2.1 Analysis of problem 1

The preprocessing effect has a very important influence on the final effect of the tracking algorithm. For the over-bright and over-dark images, gamma transform and histogram equalization should be carried out on the basis of image gray-scale to obtain the image with good illumination correction. For the background noise in the image, the gaussian filtering method is used to denoise the image. Then, we use a variety of edge detection algorithms to detect the edge contour of silicon dioxide, based on this image sharpening.

The location of the center of mass first requires the establishment of a coordinate system, and the description indicates that the position of the corundum crucible in the image is fixed, so we choose the center of the corundum crucible as the origin of the coordinate system to draw the moving trajectory of the center of mass. In order to

compare the advantages and disadvantages of the algorithm, we manually labeled the centers of silica to evaluate the tracking algorithm.
Finally, we tracked the preprocessed image, calculated the silicon dioxide center, and drew the center movement trajectory diagram. The location of the center of mass requires three-dimensional information, and we will solve this in the third question.

2.2 Image enhancement

Image enhancement algorithms are commonly used to adjust the brightness, contrast, saturation and hue of an image to increase its clarity and reduce noise. Image enhancement algorithms are often combined with multiple algorithms to achieve the above functions.

Since the original time series image contains part of the color image, we first conduct the image graying. According to the importance and other indexes, the three components in the color image were weighted and averaged with different weights. Because the human eye has the highest sensitivity to green and the lowest sensitivity to blue. Therefore, a reasonable grayscale image can be obtained by weighted average of the three components of RGB according to the following formula[1].

$$Gray(i,j) = 0.299 \cdot R(i,j) + 0.578 \cdot G(i,j) + 0.114 \cdot B(i,j) \qquad (2-1)$$

Based on the observation of the original time series images, we can see that some of the images are too bright (Figure 2-1) and some of the images are too dark (Figure 2-2). Therefore, we adopt the method of histogram equalization after gamma transform to enhance the images.

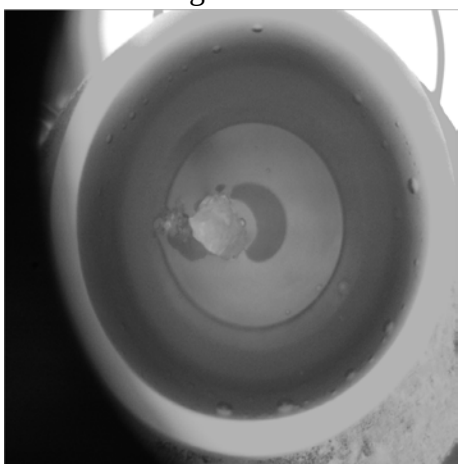

Figure 2-1 Dark picture

Figure 2-2 The highlighted image

Gamma transform is mainly used for image correction, correcting the image with too high or too low gray level to enhance the contrast. The transformation formula is the product operation of each pixel value on the original image[2]:

$$s = cr^{\gamma} \qquad (2-2)$$

The effect of γ transform is similar to that of logarithmic transform. When γ >1 is processed, the low gray value in a narrow range is mapped to the gray value in a wide range, while the high gray value in a wide range is mapped to the gray value in a narrow range.When γ <1, the opposite happens.Its function curve is as follows(Figure 2-3):

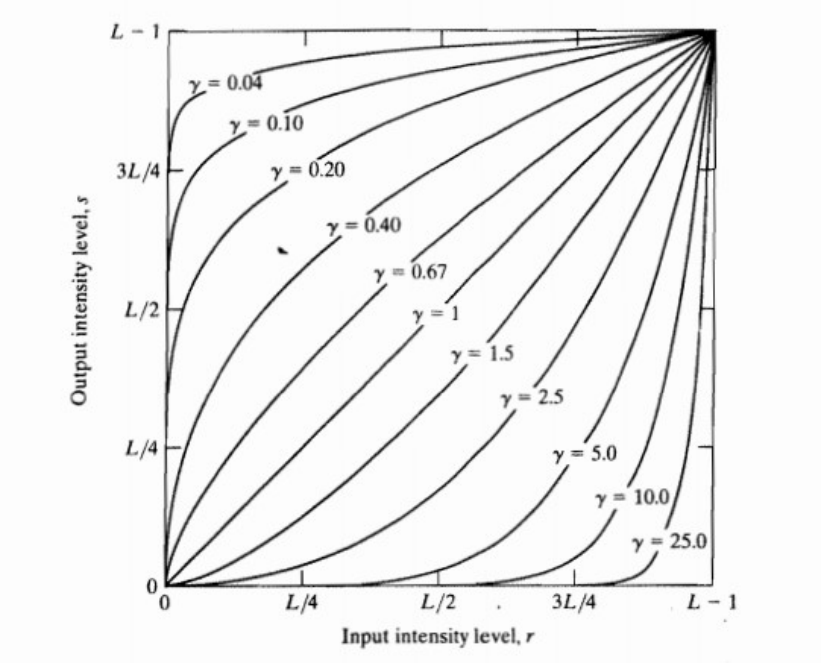

Figure 2-3 function curve

Histogram equalization is the basis of a variety of spatial processing techniques. Histogram operation can be effectively used for image enhancement. The calculation formula of pixel value after equalization is as follows[3]:

$$h(v) = round\left(\frac{(cdf(v) - cdf_{\min})(L-1)}{(M \times N)\text{-cdf}_{\min}}\right) \qquad (2-3)$$

M,N are the width and height of the image, L is the maximum gray level, cdf is the cumulative probability distribution function, and round is the rounding function.

The enhanced grayscale image is shown below:

corresponding to the image enhancement effect in Figure 2-1 (Figure 2-4) and the image enhancement effect in Figure 2-2 (Figure 2-5).

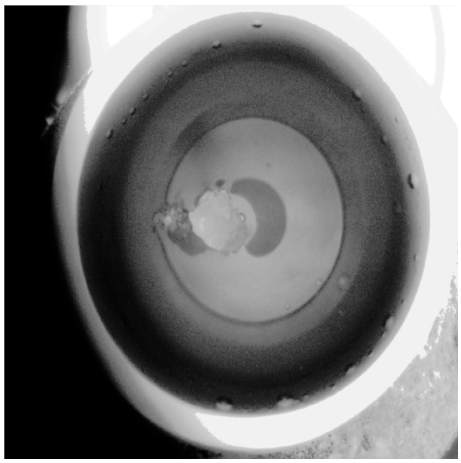

Figure 2-4 Enhanced image

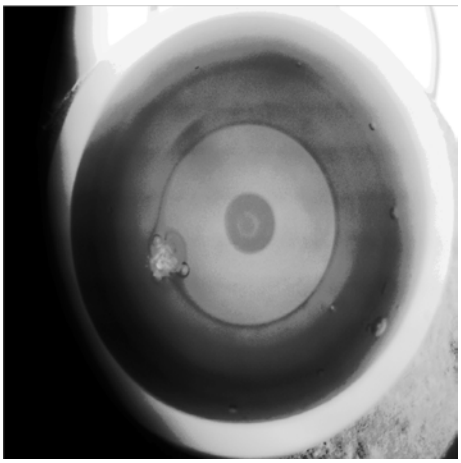

Figure 2-5 Enhanced image

2.3 image filtering

The purpose of image filtering is to extract the features of objects as the feature pattern of image recognition and to eliminate the mixed noise when the image is digitized. However, there are two requirements for image filtering, the first is not to damage the outline and edge of the image and other important information, the second is to make the image clear visual effect. Common image filtering methods include mean filtering, median filtering, bilateral filtering and gaussian filtering. The comparison of algorithm effects is shown in table 2-1:

Table 2-1 comparison of advantages and disadvantages of various filter functions

filter function	advantage	disadvantage
average filtering	simple and fast	Image edges and details tend to blur
median filtering	picture definition is basically maintained	Gaussian noise cannot be handled
bilateral filtering	Edge preserving	High frequency noise cannot be filtered
Gaussian filter	preserving the overall gray distribution	

On the basis of comparing various filtering methods, we chose the gaussian filtering algorithm with good effect. The effect diagram is as follows. Figure 2-6 is the image before filtering, and Figure 2-7 is the image after filtering.

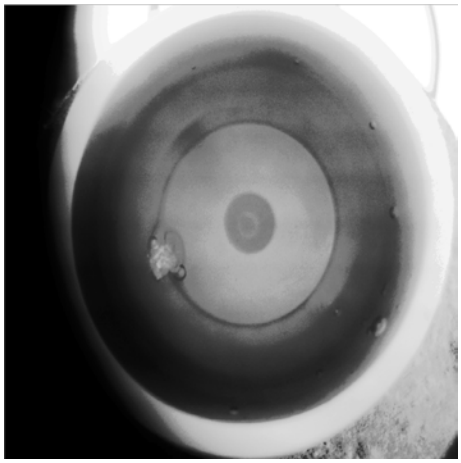

Figure 2-6 Pre-filtered image

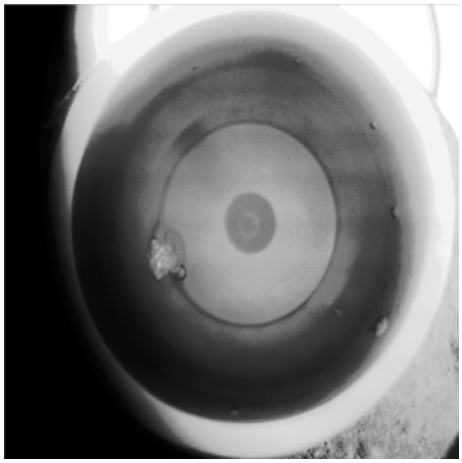

Figure 2-7 Filtered image

2.4 image sharpening

Image sharpening is the contour of a short image, the enhancement of the edge of the image and the part of the grayscale jump, so that the image becomes clear, divided into two categories: spatial domain processing and frequency domain processing.

Image sharpening is to highlight the edge of the object on the image, the outline, or some linear target features, improve the contrast between the object edge and the surrounding pixels. The common method is to add the upper edge contour to the original image as the final image sharpening diagram.

There are many kinds of edge detection algorithms, among which Roberts, Prewitt, Sobel and Canny are commonly used. The Roberts operator is located accurately, but is sensitive to noise because it does not include smoothing. Both Prewitt and Sobel are first-order differential operators. The former is the average filter, while the latter is the weighted average filter. Both of them have a good detection effect for low-noise gray gradient images, but the processing effect for mixed multi-complex noise images is not ideal. Canny operator has the best effect, but it is easy to misjudge the noise as the boundary.

Here, we use Sobel operator and Canny operator fusion algorithm to detect the edge of gray image. This method can not only ensure fine contour, accurate edge positioning, but also have good noise processing ability, which can effectively sharpen the grayscale image.

Canny edge detection algorithm of traditional step mainly include:

gaussian filtering smoothing images;

calculate the gradient magnitude and direction;

to the maximum gradient amplitude inhibition;

using double threshold detection and edge connection. Edge intensity of M (i, j) and point vector theta (i, j) expressed by the following formula:

$$M(i,j) = \sqrt{P_x(i,j)^2 + p_y(i,j)^2} \qquad (2-4)$$

$$\theta(\mathrm{i,j}) = \arctan\left(\frac{P_y(\mathrm{i,j})}{P_x(\mathrm{i,j})}\right) \qquad (2-5)$$

Where Px and Py are defined as follows:

$$P_x = \frac{\partial G(x,y,\sigma)}{\partial x} \cdot f(x,y)$$

$$P_y = \frac{\partial G(x,y,\sigma)}{\partial y} \cdot f(x,y) \qquad (2-6)$$

σ is a constant, f(x,y) is the gray level of any point, əG is the gaussian gradient.

In the classical Sobel operator, the horizontal and vertical direction template is convolved with the image f(x,y) to approximate the gradient value. The horizontal gradient Gx and vertical gradient Gy can be solved by the following formula:

$$G_x = f(i+1,j-1) + 2f(i+1,j) + f(i+1,j+1) - f(i-1,j-1) - 2f(i-1,j) - f(i-1,j+1)$$

$$G_y = f(i-1,j+1) + 2f(i,j+1) + f(i+1,j+1) - f(i-1,j-1) - 2f(i,j-1) - f(i-1,j-1) \qquad (2-7)$$

(*i*,*j*) is the point coordinates, *f*(*i*,*j*) is the gray level of any point.

The fusion ratio of Sobel operator and Canny operator can be 1:1, and the edge detection effect is as follows:

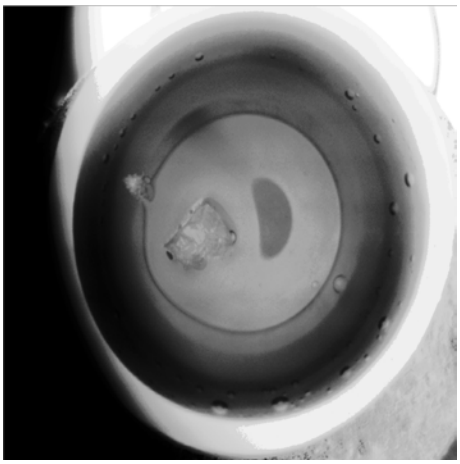

Figure 2-8 gray scale Figure

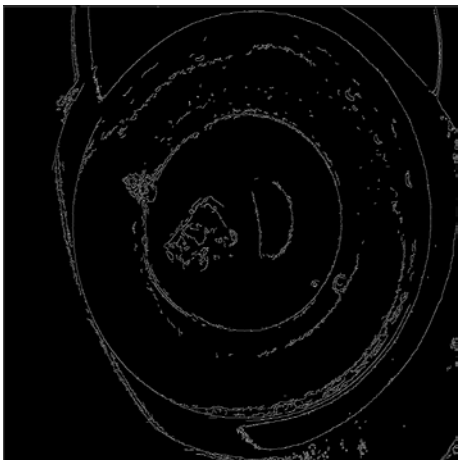

Figure 2-9 Canny image

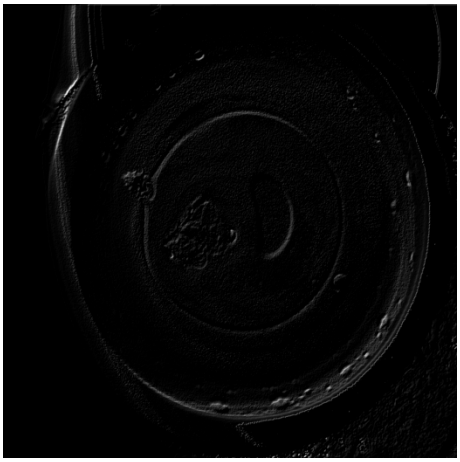

Figure 2-10 Sobel image

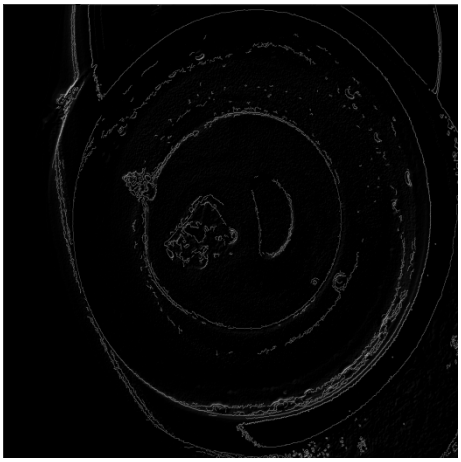

Figure 2-11 edge image

Based on the edge contour, we obtained the following sharp image (Figure 2-12)

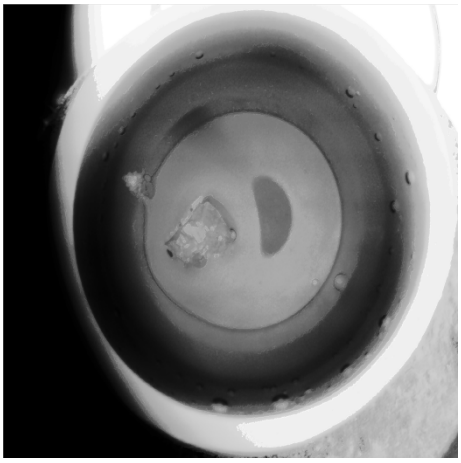

Figure 2-12 sharp image

2.5 target tracking

The task of target tracking is to predict the size and position of the target in subsequent frames given the size and position of the target in the initial frame of a video sequence. The general process is to first determine the initial target box, generate a number of candidate boxes in the next frame, extract the characteristics of these

candidate boxes and score them, and select the candidate box with the highest score as the predicted target.

When we observe the original target sequence images given in the attachment, we find that some images show the phenomenon of image jitter, and the image jitter has a great impact on the tracking algorithm of small target objects. Therefore, we choose KCF algorithm to firstly track and locate the whole image to eliminate jitter, and then use KCF algorithm to accurately track the silica on this basis.

Here is the principle of KCF：

KCF is a discriminative tracking method, which generally trains a target detector in the tracking process, and use the target detector to detect whether the predicted position of the next frame is the target, and then updates the training set with the new detection result to update the target detector.In the training of the target detector, the target region is generally selected as positive sample, while the surrounding region of the target is negative sample. The closer the region is to the target, the more likely it is to be positive sample[4].

KCF uses the cyclic matrix of the target surrounding area to collect the positive and negative samples,meanwhile,using ridge regression training target detector.The operation of the matrix is transformed into the Hadamad product of the vector, that is, the dot product of the element, by the property of the circulant matrix diagonalizable in the Fourier space, which greatly reduces the computation, improves the operation speed and makes the algorithm meet the real-time requirement[5].

We use the tracking results of the previous frame are used to train the correlation filter(ω),The problem can be expressed as:

$$f(x_i) = \omega^T x_i \qquad (2-8)$$

The optimization function consists of the least squares and the regular terms in the convex optimization, which is the most commonly used ridge regression method:

$$min \sum_i (f(x_i) - y_i)^2 + \lambda \|\omega\|^2 \qquad (2-9)$$

(x_i, y_i) is the training sample set, *w* is the weight coefficient of the correlation filter,λ is a constant that controls the structural complexity of the system, *I* is the identity matrix[6].

Set the partial derivative equal to zero to get the solution to ω:

$$\omega = (X^T X + \lambda I)^{-1} X^T y \qquad (2-10)$$

In linear algebra, the cyclic matrix is a special form of Toeplitz matrices, before its row vector of each element is a row vector of each element in turn moves to the right a position to get results.The fast cyclic matrix solution can use the discrete Fourier transform, so have important applications in the numerical analysis.

Here is an example of a cyclic matrix：

$$\begin{bmatrix} c_0 & c_{n-1} & \cdots & c_2 & c_1 \\ c_1 & c_0 & c_{n-1} & \cdots & c_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ c_{n-2} & c_{n-3} & \cdots & c_0 & c_{n-1} \\ c_{n-1} & c_{n-2} & \cdots & c_1 & c_0 \end{bmatrix} \tag{2-11}$$

The diagonalization property of cyclic matrices is used to simplify:

$$X = Fdiag(\hat{x})F^H \tag{2-12}$$

Substitute in the calculation and simplify to obtain the following results:

$$\omega = \frac{\widehat{x^*}\hat{y}}{\widehat{x^*}\hat{x} + \lambda} \tag{2-13}$$

In order to show the tracking effect of the algorithm, we intercept the tracked silica area and show part of it as follows:

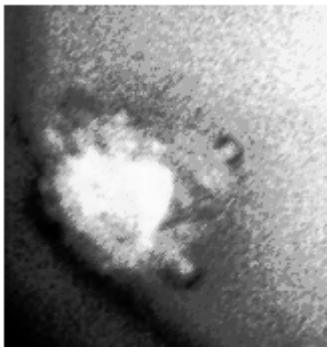

Figure 2-12

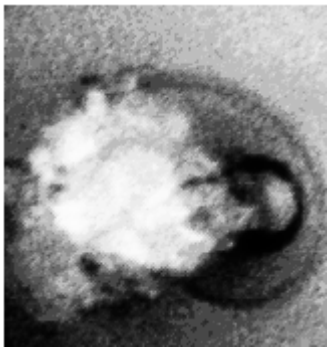

Figure 2-13

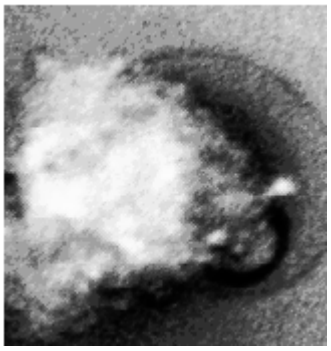

Figure 2-14

Figure 2-15

2.6 coordinate system determination

The description indicates that the position of the corundum crucible is fixed in the image, so we can choose the center of the corundum crucible as the center of the coordinate system, so as to determine the trajectory of the silica. We use manual punctuation method to record the coordinate information of the crucible center point in 114 images, which is relative to the gray image after the initial KCF algorithm dejabs. As for the coordinate information of 114 recorded center points, there may be some recording errors in part of the data, so we adopt the improved single-center point

clustering method for processing. Firstly, the average of all recorded coordinate points is calculated, and then 90% of the data around the average point is selected to calculate the mean again to get the new center point. By observing the original image sequence, it can be seen that the size of silica in the early stage had a slight influence on the judgment of the crucible's center. For this, we weighted and averaged 90% of the data around the center point to get the final center point.

The following figure shows the scatter chart recording all coordinate points (Figure 2-16) and the scatter chart recording 90% of coordinate points around the initial average point (Figure 2-17).

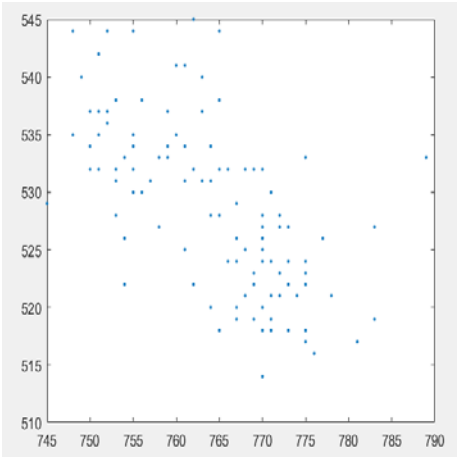

Figure 2-16 the scatter chart of all coordinate points

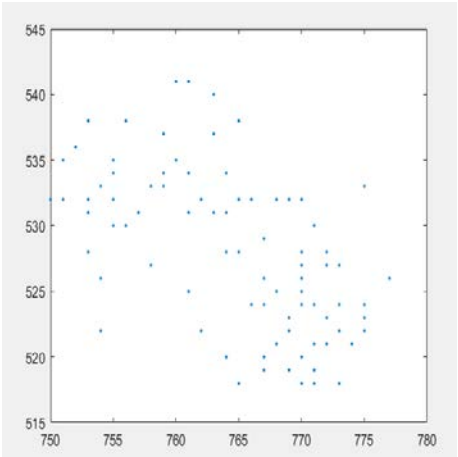

Figure 2-17 the scatter chart recording 90% of coordinate points around the initial average point

In order to evaluate the actual effect of KCF tracking algorithm, we manually labeled the geometric center of silica for evaluation. Due to the error of single annotation, the three members of the team recorded the silica center points in 114 images, and calculated the mean value as the final silica center points.

In order to investigate the movement range of silica in the crucible during melting, the bottom circle radius (radius 1) and the upper inner diameter circle radius (radius 2) of the corundum crucible were recorded. The bottom circle radius and the top circle radius are shown in Figure 2-18:

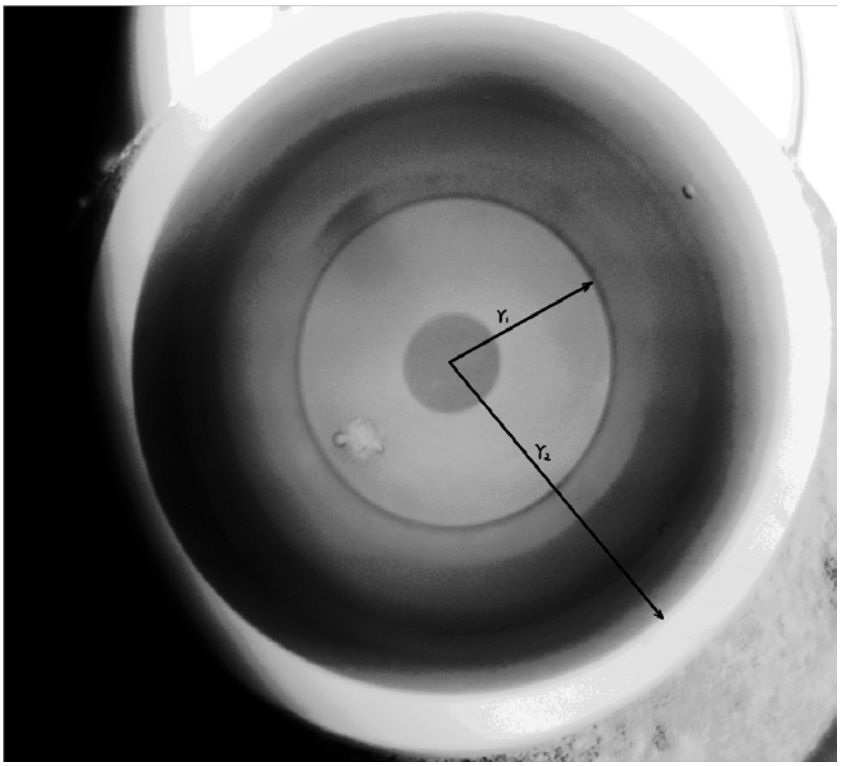

Figure 2-18 the bottom circle radius and the top circle radius

2.7 model evaluation

In Figure 2-19, the actual silica center diagram and the silica center diagram obtained by KCF tracking algorithm are drawn with the crucible center as the center. The inner circle is the bottom circle and the outer circle is the upper inner diameter circle. In Figure 2-20, we compare the pixel distance error of the silica center obtained by the KCF tracking algorithm with the actual silica center. The distance error of the pixel and the radius of the upper inner diameter circle are taken as a percentage to obtain the distance error percentage graph. It can be seen that the KCF tracking algorithm has a better tracking effect when the volume of silica is large, and the average percentage error of the first 90 sequential images is only 1.9%.

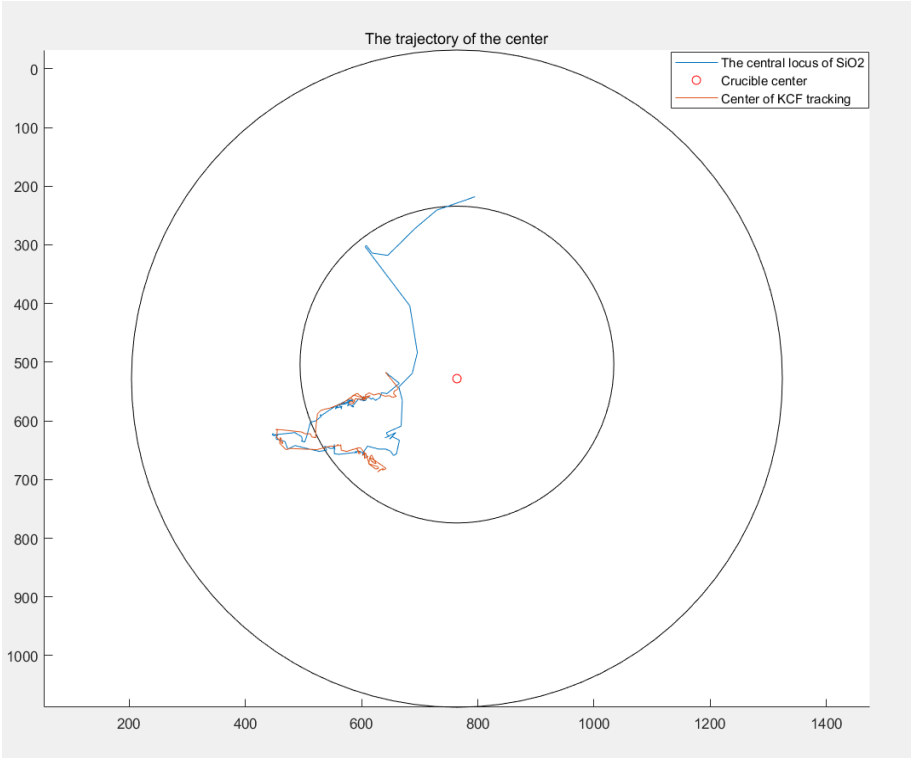

Figure 2-19 the trajectory of the center

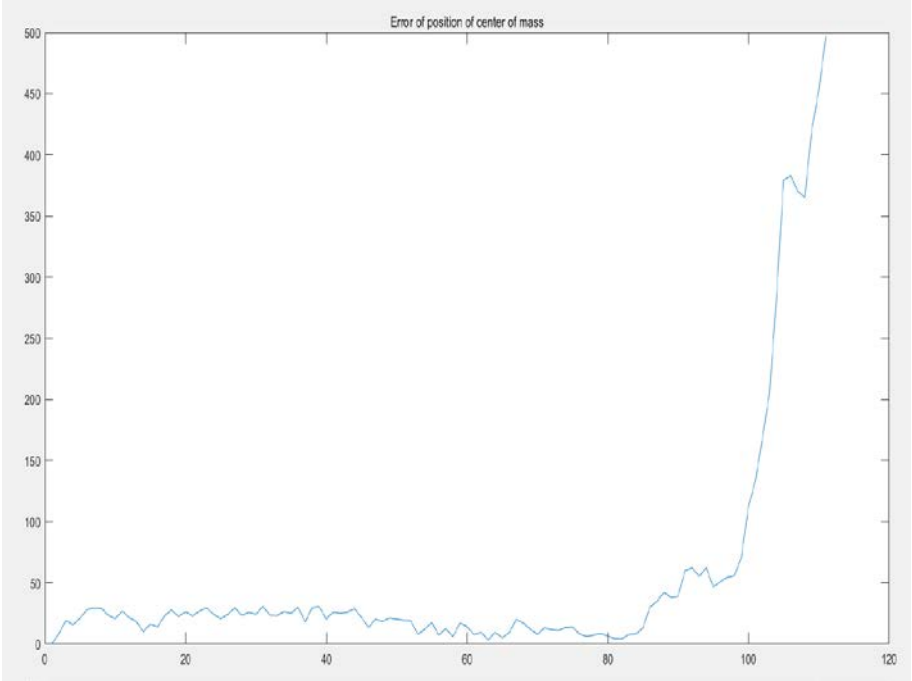

Figure 2-20 error of position of center of mass

In addition, we also plotted the movement rate of the actual silicon dioxide center (Figure 2-21) and the pixel distance from the crucible center (Figure 2-22). As can be seen from Figure 2-22, the movement rate of silica increases gradually with the increase of time. The physical explanation may be that the smaller the mass, the faster the motion under the same thermal pressure. As can be seen from Figure 2-22, the distance from the center of silica to the center of the crucible tends to increase first and then decrease,

which is the effect of touching the inner wall of the crucible. The sudden increase in the final distance is the result of the undirected thermal pressure and increased velocity.

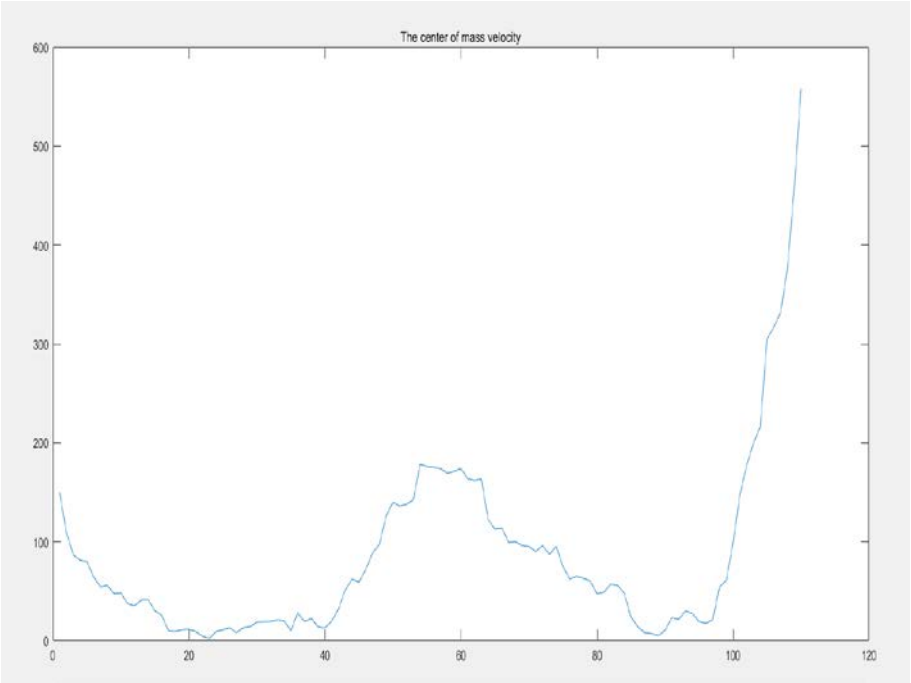

Figure 2-21 the center of mass velocity

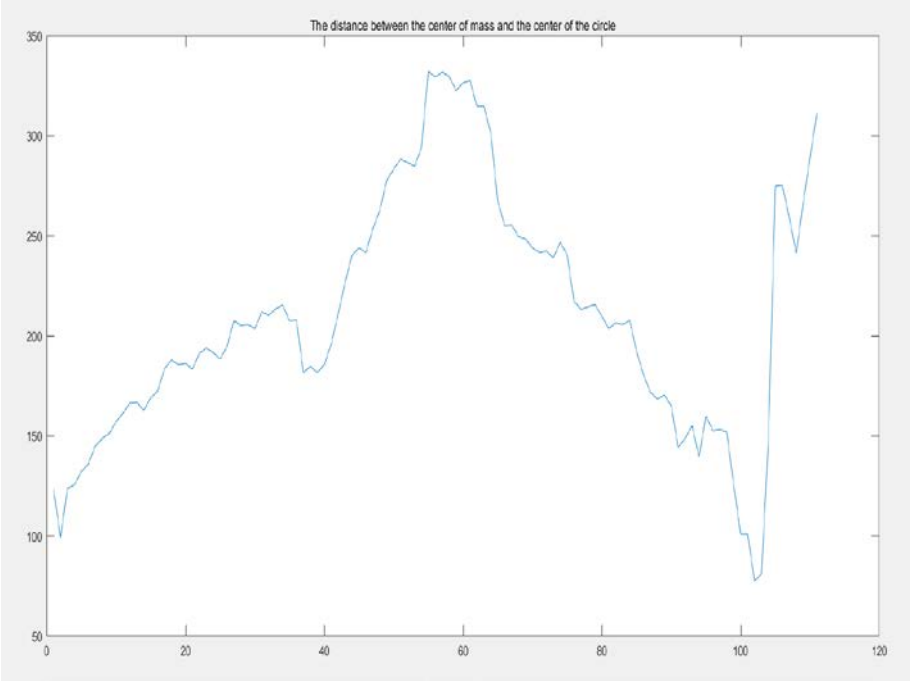

Figure 2-22 the distance between the center of mass and the center of the circle

Finally, the location of the center of mass of silicon dioxide has a great relationship with the volume, which will be solved after 3D modeling of silicon dioxide in question 3.

3 Model establishment and solution of problem 2

3.1 problem analysis

There are a lot of information about the edge contour. For a planar image, the obvious features are perimeter, area and minimum circumferential radius. The edge contour can be extracted by using the enhanced grayscale image for Canny operator edge detection.

In order to eliminate the influence of the central pixel of the crucible and the round pixel of the bottom of the crucible in the image, the last three images of the image sequence can be superimposed, and the image effect can be greatly enhanced by subtracting the superimposed graph from the remaining images. According to the silicon dioxide center obtained in problem 1, a minimum rectangular region containing silicon dioxide was cut out, and the area diagram was obtained according to the upper and lower contours. For the possible insufficient filling, the morphological closure operation is carried out.

Finally, the area map was morphologically eroded to obtain a clear edge contour map, and the perimeter was calculated accordingly. For the calculation of the minimum circumferential circle, FindContour can be performed on the obtained clear edge contour graph.

3.2 edge processing

The enhanced grayscale image can be extracted by Canny operator edge detection. As shown in the figure below, Figure 3-1 is the grayscale enhanced, and Figure 3-2 is the contour image detected by Canny edge.

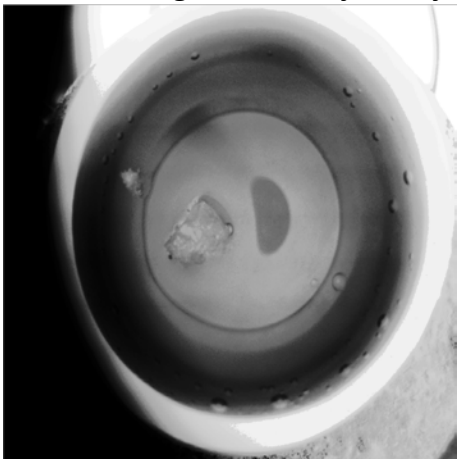

Figure 3-1 grayscale enhanced

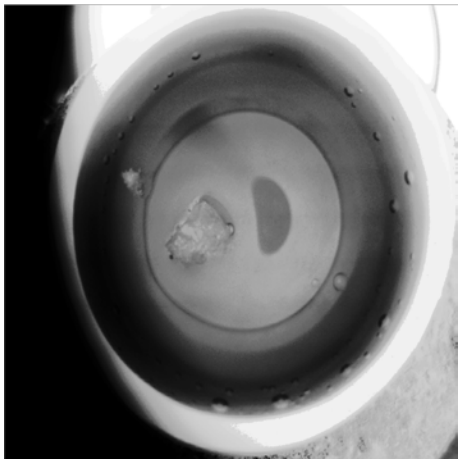

Figure 3-2 the contour image detected by Canny edge

In order to eliminate the influence of the central pixel of the crucible and the circular pixel of the bottom of the crucible in the image, the last three images of the image sequence can be superimposed to serve as the interference background of the contour map. Subtract this overlay from the rest of the image to greatly reduce the

impact of the background. The last three images (Figure 3-3, Figure 3-4, Figure 3-5) and the superimposed image (Figure 3-6) are shown below.

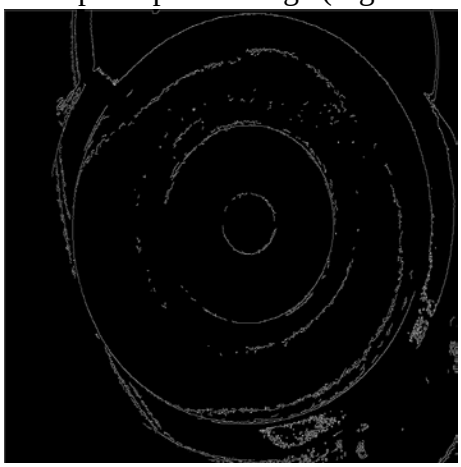

Figure3-3

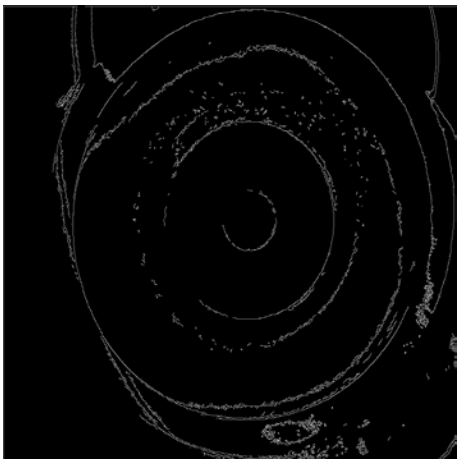

Figure 3-4

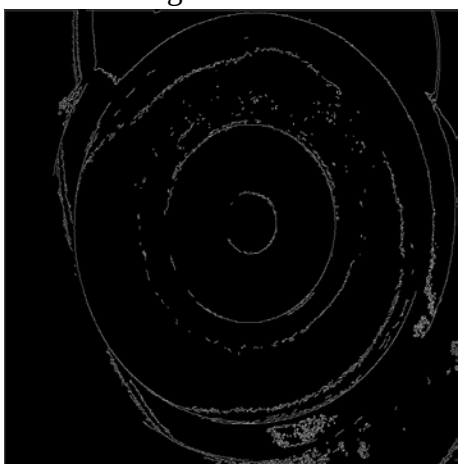

Figure 3-5

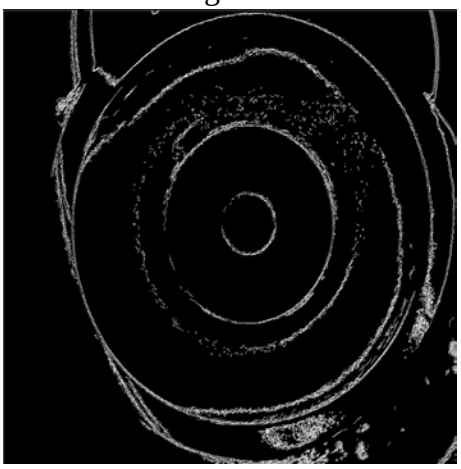

Figure 3-6

Since the superposition diagram may not be able to fully represent the effect of the contour background, we used morphological gradient processing for many times, followed by median filtering and OTSU binarization processing.

Corrosion can reduce the scope of the target area, essentially causing the image's boundary to shrink, which can be used to eliminate small and meaningless objects.The formula is expressed as:

$$A\Theta B = \left\{x,y\middle|(B)_{xy} \subseteq A\right\} \qquad (3-1)$$

Expansion will make the range of the target area larger, and merge the background points in contact with the target area into the target, so that the target boundary expands outwards. The function is to fill some holes in the target region and eliminate small particle noise contained in the target region. The formula is expressed as:

$$A \oplus B = \left\{x,y|(B)_{xy} \cap A \neq \varphi\right\} \qquad (3-2)$$

The principle of median filtering is to replace the value of a point in a digital image with the median value of each point in an area around that point. Take 3×3 window as an example:

$$g(x,y) = median(f(x-1,y-1), f(x,y-1), f(x+1,y-1), f(x-1,y),$$
$$f(x,y), f(x+1,y), f(x-1,y+1), f(x,y+1), f(x+1,y+1)) \quad (3-3)$$

f(x,y) is the pixel value of point (x,y), g(x,y) is the pixel value after median filtering, median is the median function.

OTSU binarization divides image X into image foreground X0 and image background X1. If the image size is M·N, the number of pixels in the image whose gray value is less than the threshold value T is denoted as N0 (image foreground), and the number of pixels whose gray value is greater than the threshold value T is denoted as N1 (image background). So:

$$N0 + N1 = M \cdot N \quad (3-4)$$

In addition, the proportion of foreground pixel in the image is w0, and the average gray level is u0. The proportion of background pixels in the image is w1, and the average gray level is u1. The average gray scale of the image is u, and the inter-class variance is g, then:

$$w0 = \frac{N0}{M \cdot N} \quad (3-5)$$

$$w1 = \frac{N1}{M \cdot N} \quad (3-6)$$

$$w0 + w1 = 1 \quad (3-7)$$

$$u = w0 \cdot \mu0 + w1 \cdot \mu1 \quad (3-8)$$

$$g = w0 \cdot (u0 - u)^2 + (u1 - u)^2 \quad (3-9)$$

The threshold value T, which maximizes the variance between classes, is the binarization threshold.

The resulting background is shown in Figure 3-7. Select one of the image sequences (Figure 3-8), subtract the background image, and get the edge contour image with better quality (Figure 3-9). As we can see, the edge background around the silica is well removed.

Figure 3-7 the resulting background

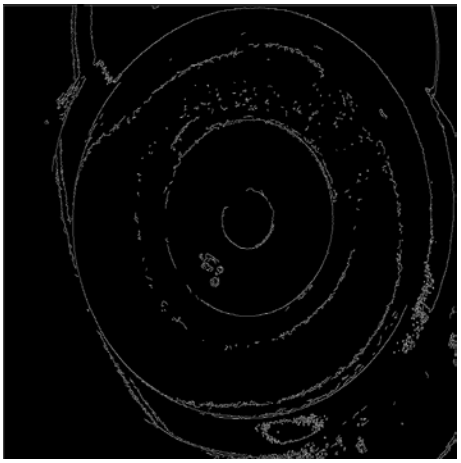

Figure 3-8

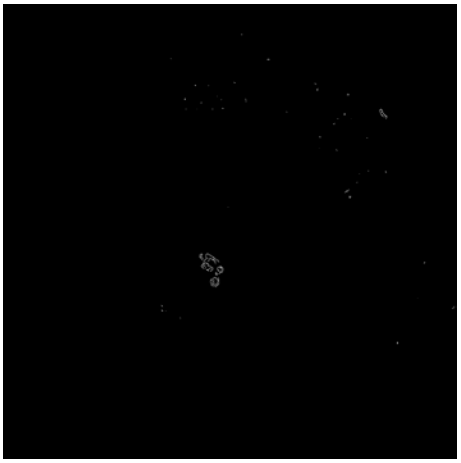

Figure 3-9

3.3 drawing of area diagram and circumference diagram

In the edge contour obtained by the above method, a minimum rectangular region containing silica is cut out according to the silica center obtained by problem 1. In this rectangular area, fill in the image information between the upper and lower edges to get the image as shown in Figure 3-10

Figure 3-10

As we can see, some areas are not fully filled, and black vertical stripes appear. To this end, the area image as shown in Figure 3-11 was obtained by morphological closure operation. Then, morphological corrosion was performed on the area image to obtain the area image after corrosion, as shown in Figure 3-12. The difference between the original area image and the corroded area image results in a clear edge image (as shown in Figure 3-13), which can be used to calculate the perimeter.

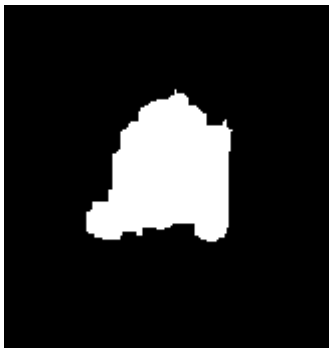

Figure 3-11

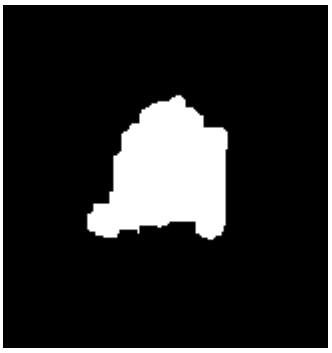

Figure 3-12

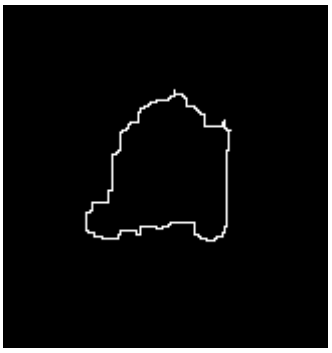

Figure 3-13

Expansion is the expansion of the highlighted part of the image. The effect diagram has a larger highlighted area than the original image. Corrosion is the erosion of the highlighted areas in the original image. The renderings have smaller highlighted areas than the original image. The expansion operation and corrosion operation are shown in formula 3-7 and formula 3-8 respectively:

$$F \oplus S=\max\{F(x+m,y+n)\} \qquad (3-10)$$

$$F \Theta S=\min\{F(x+m,y+n)\} \qquad (3-11)$$

Where F is the grayscale image, (x,y) is the image point coordinates, S is the morphological structure element, and (m,n) is the structure element point coordinates.

Morphological close operation is to expand the image first and then corrode the image. Then the morphological closed operation expression as shown in the formula 3-9 can be obtained:

$$F \cdot S=((F \oplus S) \Theta S) \qquad (3-12)$$

3.4 solve for the minimum radius of the circumscribed circle

The minimum circumscribed circle of silica is drawn as shown in Figure 3-14:

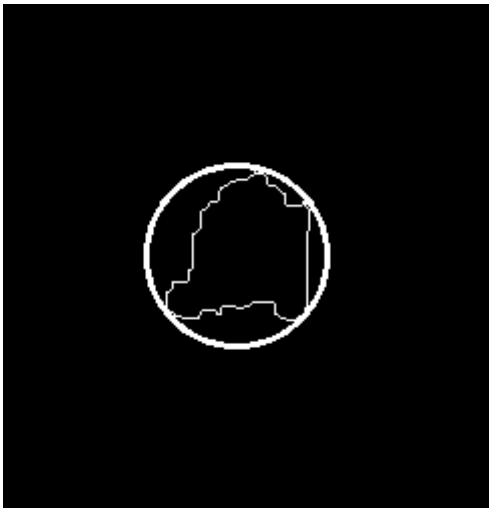

Figure 3-14 the minimum circumscribed circle of silica

3.5 model evaluation

As shown in the Figure, we plotted the variation trend of silicon dioxide area with time (Figure 3-15), the variation trend of silicon dioxide volume with time (Figure 3-16), and the variation trend of silicon dioxide minimum circumscribed circle radius with time (Figure 3-17).

The circumference and the minimum circumferential radius were fitted by RANSAC line. The basic assumptions of RANSAC are as follows: 1) the data consists of "local points". For example, the distribution of data can be explained by some model parameters. 2) "outlier" is the data that cannot adapt to the model; 3) data other than this is noise.

The basic steps of RANSAC algorithm are as follows: 1) randomly select two points; 2) through these two points, the model equation represented by these two points can be calculated; 3) put all data points into this model to calculate the error; 4) find all points that meet the error threshold; 5) then we repeat the process of 1-4 until we reach a certain number of iterations to select the most supported model.

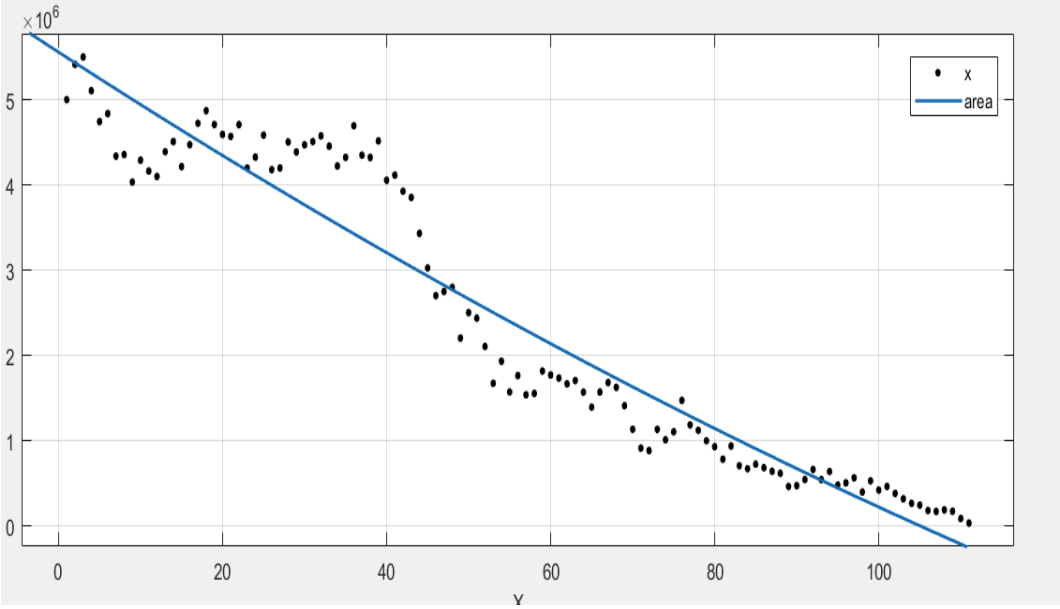

Figure 3-15 the variation trend of silicon dioxide area with time

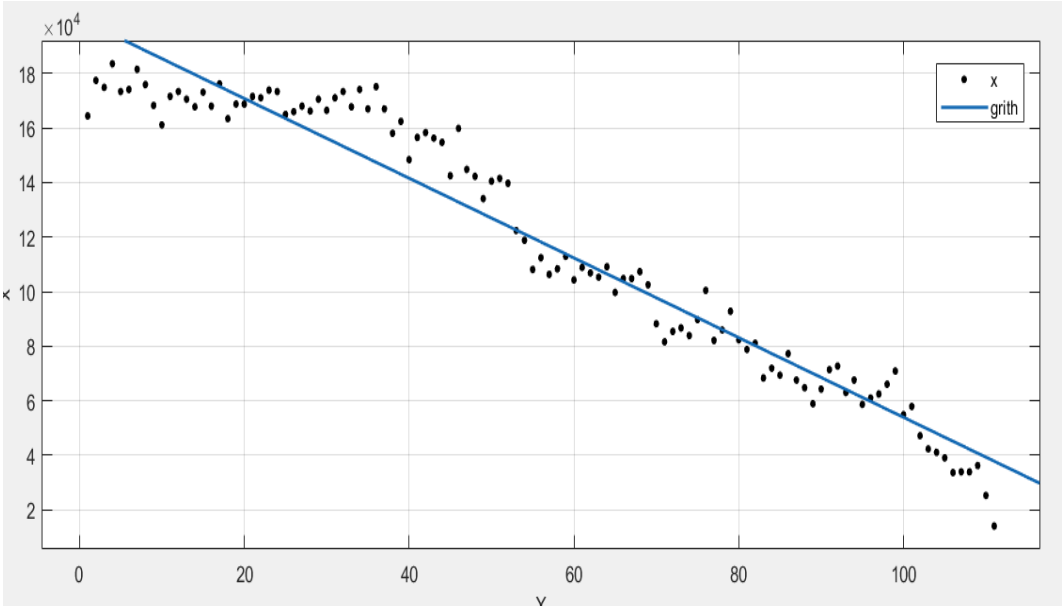

Figure 3-16 the variation trend of silicon dioxide volume with time

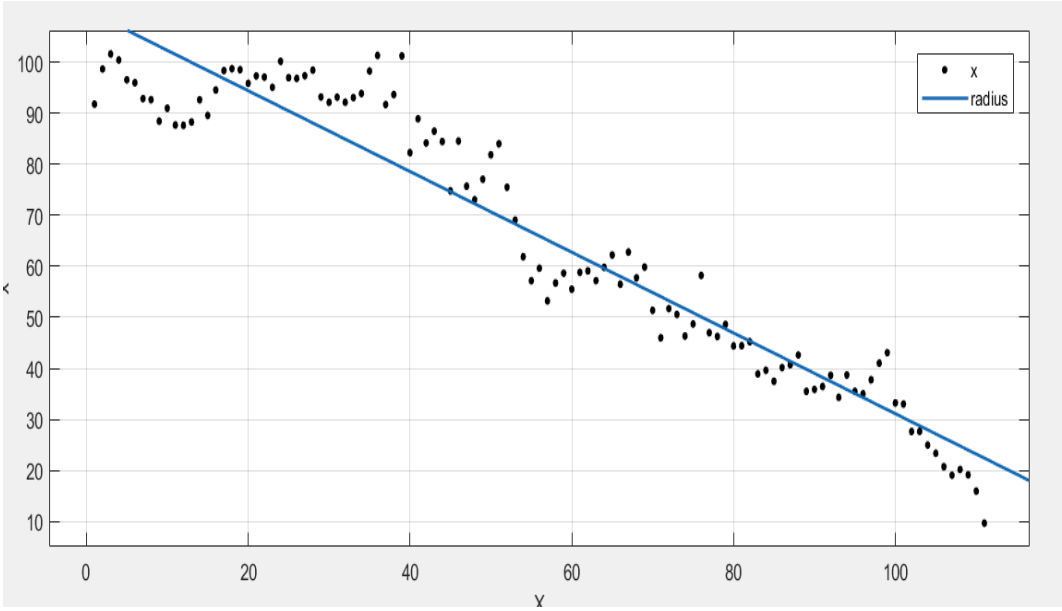

Figure 3-17 the variation trend of silicon dioxide minimum circumscribed circle radius with time

As can be seen, with the increase of time, the area, perimeter and the minimum circumferential circle radius show a downward trend. The decrease of circumference and the minimum circumferential circle radius is a primary function trend, while the decrease of area is a quadratic function trend.

4 Model establishment and solution of problem 3

4.1 problem analysis

According to the 2D image information, the SFS (Shape From Shading) algorithm is used to construct the 3D image. In order to eliminate the influence of background

factors, the sio2 area map obtained from problem 2 was used to accurately locate the sio2 volume change of time series. The actual melting rate of silica can be obtained by calculating the gradient of the volume change diagram.

In addition, after the volume of silica was obtained by 3D modeling, the centroid of silica was calculated.

4.2 3D diagram construction

The 2D to 3D conversion adds binocular parallax depth to the digital image perceived by the brain. Therefore, if you complete the 3D reconstruction, you can greatly improve the immersion effect while viewing the stereo image. However, in order to be successful, the transformation should be done with sufficient accuracy and correctness.

We used SFS algorithm to reconstruct the gray scale image of silica in 3D.

Shape from shading is one of the key technologies of 3D shape recovery in computer vision. Its task is to recover the relative height or normal direction of each point in a single image by using the shading changes on the object surface, laying a foundation for further 3D reconstruction of the object.

The method to restore the shape of light and dark is a method of measuring objects without contact, which only requires one image. In the process of recovery, there is no requirement for camera calibration, nor for some prior information. SFS USES the grayscale information of the image, selects the appropriate reflection model, obtains the normal vector information of the surface, and recovers the shape information through the vector information[7].

Under the lambert-body illumination reflection model, the brightness of the object's surface point is only related to the cosine of the light source and the normal vector of the surface point. The gray-scale value of the object surface captured by the camera is related to the following four points: the geometry of the object surface; The incident intensity and direction of the light source; The direction of the camera relative to the object; The reflective properties of an object's surface. As shown in Figure 4-1:

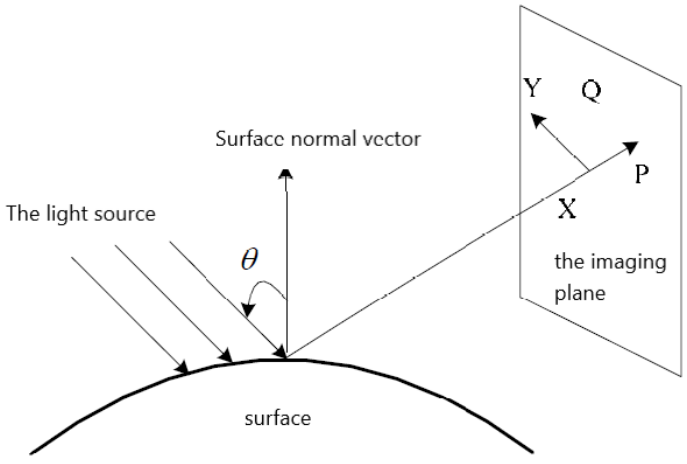

Figure 4-1

The conditional hypothesis of SFS is: 1. The reflection model of the surface is Lambertian diffuse reflection model; 2. The light source is infinite distant point light source; 3. The imaging geometric relation is orthogonal projection.

Assumes that the incident light intensity for I, reflection coefficient as the rho, light source vector $\vec{S} = (s_x, s_y, s_z)$, the body of the table surface method of vector quantity $\vec{N} = (n_x, n_y, n_z)$, the angle between the light source and the surface normal vector is β, the intensity of the reflected light is E, by lambert body reflection model[8]:

$$E = I\,\rho\cos\beta \tag{4-1}$$

According to the Angle relation between vectors,

$$\cos\beta = \frac{\vec{S}\vec{N}}{\left|\vec{S}\right|\left|\vec{N}\right|} = \frac{s_x n_x + s_y n_y + s_z n_z}{\sqrt{s_x^2 + s_y^2 + s_z^2}\sqrt{n_x^2 + n_y^2 + n_z^2}} \tag{4-2}$$

Suppose the surface gradient of the object is(p, q), the light source vector is decomposed to the gradient direction is (p_s, q_s), and the gradient equation (4-2) can be expressed as:

$$\cos\beta = \frac{p_s p + q_s q + 1}{\sqrt{p_s^2 + q_s^2 + 1}\sqrt{p^2 + q^2 + 1}} \tag{4-3}$$

Then:

$$E = I\rho\frac{p_s p + q_s q + 1}{\sqrt{p_s^2 + p_y^2 + 1}\sqrt{p^2 + q^2 + 1}} \tag{4-4}$$

In order to find the surface height of the object, the reflection function is firstly discretized and let:

$$p = \frac{\partial z}{\partial x} = z_{i,j} - z_{i,j-1} \qquad q = \frac{\partial z}{\partial y} = z_{i,j} - z_{i-1,j} \tag{4-5}$$

I and j are the number of rows and columns of the image respectively. In this way, the reflection function can be rewritten as:

$$f\left(z_{i,j}\right) = E - R\left(z_{i,j} - z_{i,j-1}, z_{i,j} - z_{i-1,j}\right) \tag{4-6}$$

The Taylor series expansion of equation (4-6) is as follows:

$$f\left(z_{i,j}\right) \approx f\left(z_{i,j}^{n-1}\right) + \left(z_{i,j} - z_{i,j}^{n-1}\right)\frac{\partial f}{\partial z_{i,j}}\left(z_{i,j}^{n-1}\right) \tag{4-7}$$

Let the NTH iteration of its height value be $z_{i,j} = z_{i,j}^{n}$,, then:

$$z_{i,j}^{n} = z_{i,j}^{n-1} - \frac{f\left(z_{i,j}^{n-1}\right)}{\frac{\partial f\left(z_{i,j}^{n-1}\right)}{\partial z_{i,j}}} \tag{4-8}$$

$$z_{i,j}^{n} = z_{i,j}^{n-1} - \frac{f\left(z_{i,j}^{n-1}\right)}{\frac{\partial f\left(z_{i,j}^{n-1}\right)}{\partial z_{i,j}}} \qquad (4-9)$$

In this way, given an initial value, the final surface height can be calculated iteratively[9].

As shown in the figure below, the SFS algorithm performs very well in silica 3D modeling. Figure 4-2 is the grayscale image enhanced by the image, and Figure 4-3 is the 3D image after 3D reconstruction.

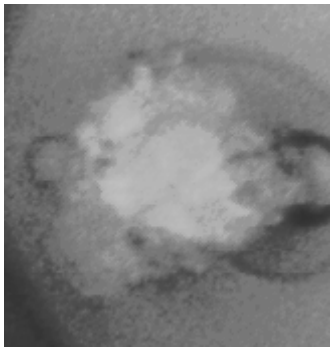

Figure 4-2 the grayscale image enhanced by the image

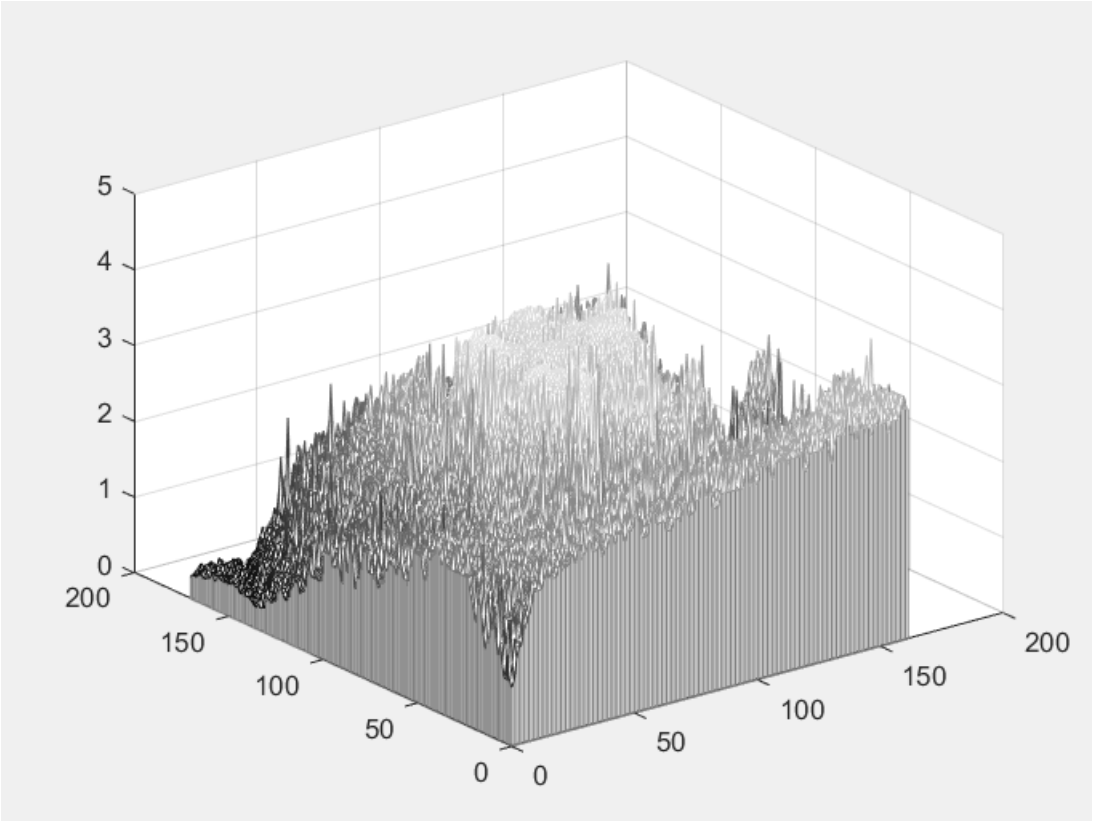

Figure 4-3 the 3D image after 3D reconstruction

4.3 melting rate model

According to the 3D image constructed by SFS algorithm, we can preliminarily

solve the volume of silica. However, due to the influence of grayscale image background, the volume determination error at the later stage of melting is very large. Here, we use the area diagram constructed by problem 2 to conduct 3D modeling of the location of silicon dioxide, and obtain the 3D diagram shown in Figure 4-4.

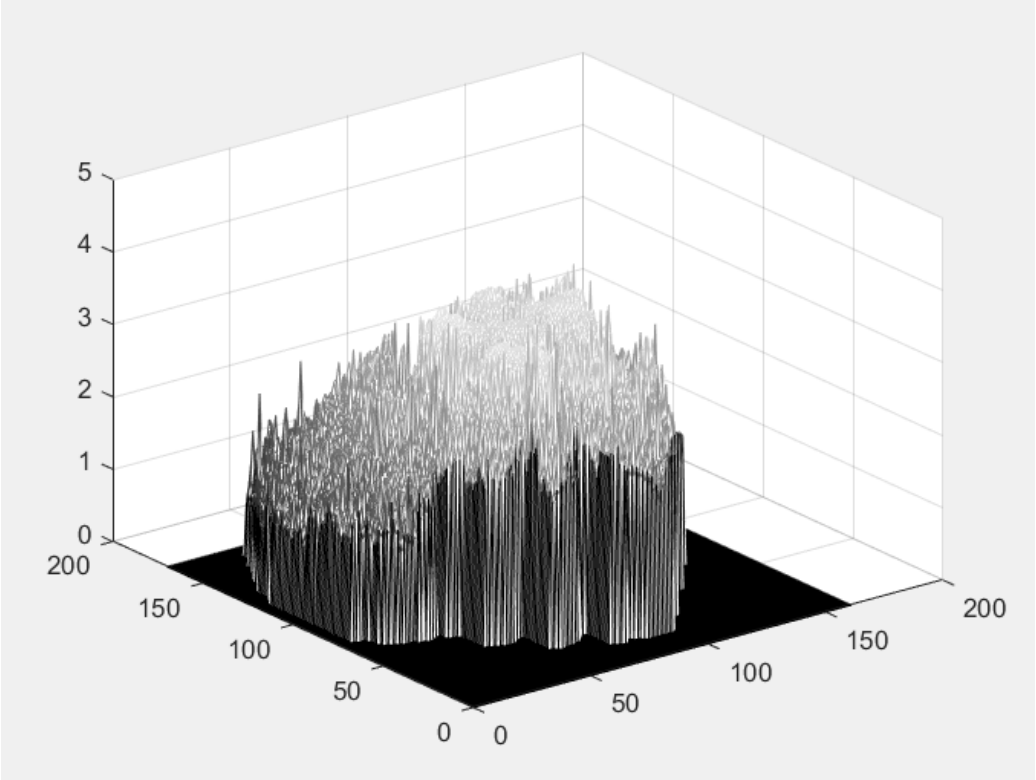

Figure 4-4 the 3D diagram

On this basis, we draw the curve diagram of the volume of silica changing with time, as shown in Figure 4-5. It can be seen that the change of volume is similar to the gaussian model, So we used the following model for fitting and achieved a model accuracy of 97.93%.

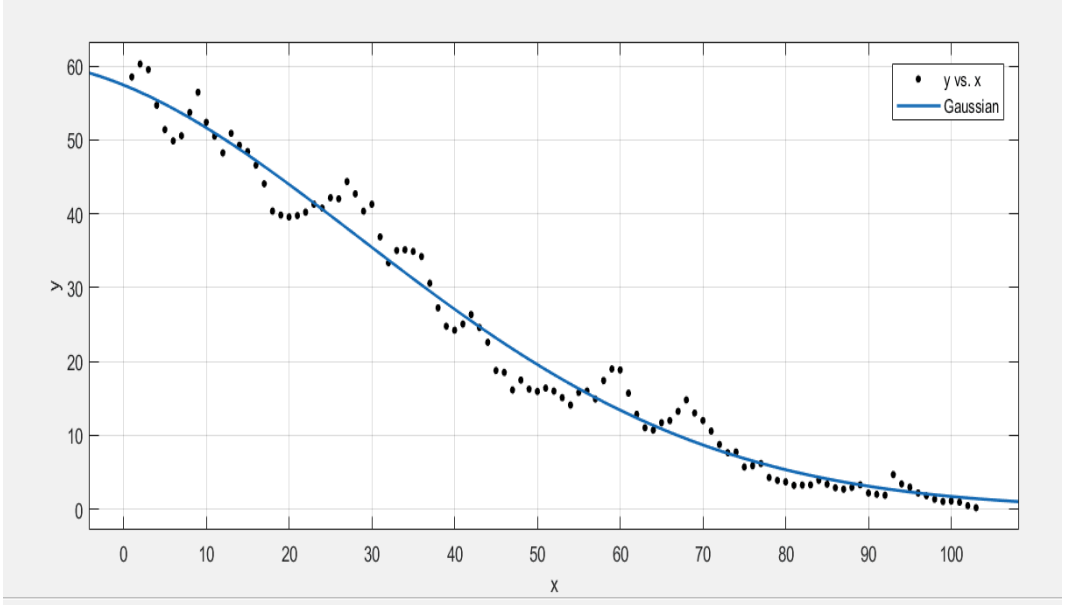

Figure 4-5 the volume of silica changing with time

We differentiate the fitted gaussian model to obtain the melting rate diagram of silica, as shown in Figure 4-6. In the early period of time series, the melting rate of silicon dioxide gradually accelerated as it was heated more fully. In the later period of time series, as the volume of silica gradually decreased, the melting rate also showed a decreasing trend until 0.

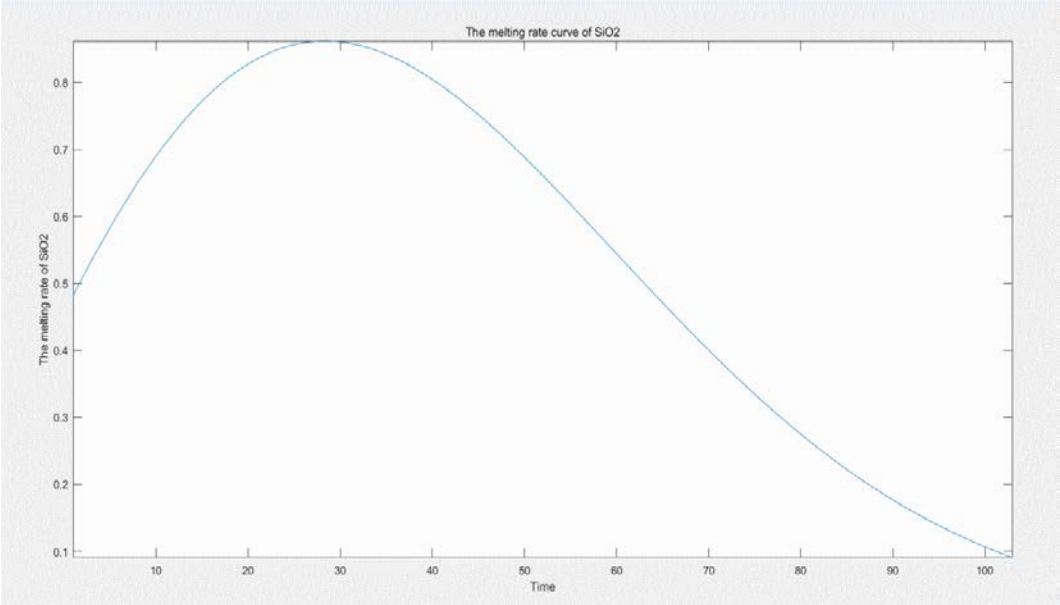

Figure 4-6 he melting rate diagram of silica

4.4 trajectory of center of mass

Since the 3D information of silica cannot be obtained in problem 1, we calculate the silica centroid here. In this model, we assume that the density of silica is uniform and there are no voids. According to the formula of the center of mass of the object:

$$\bar{x} = \frac{\int\int x\mu(x,y)\,dxdy}{\int\int \mu(x,y)\,dxdy} \qquad (4-10)$$

$$\bar{y} = \frac{\int\int y\mu(x,y)dxdy}{\int\int \mu(x,y)dxdy} \qquad (4-11)$$

μ (x,y) is the density function

The height of a point of silicon dioxide is taken as the weight of the point, and it is drawn into a grayscale map for display, as shown in Figure 4-7:

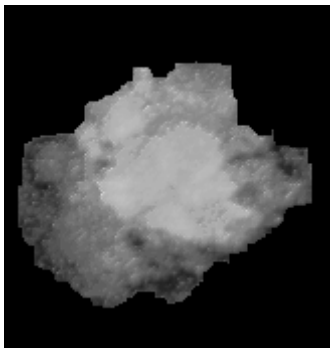

Figure 4-7

The final trajectory of the center of mass of silica is obtained, as shown in Figure 4-8, where we draw the trajectory of the center of silica made in problem 1 together.

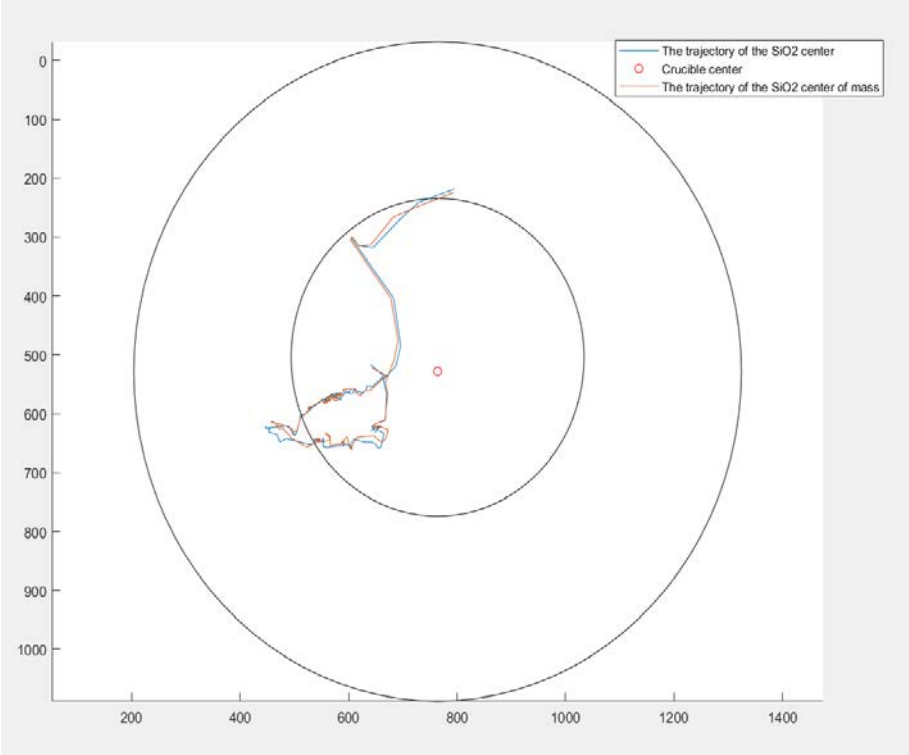

Figure 4-8 the final trajectory of the center of mass of silica

5 References

[1] Rafael C. Gonzalez, Richard E. Woods. Digital Image Processing, Second Edition[M]. Publishing House of Electronics Industry, 2007.
[2] Pratt W K. Digital image processing: 3rd edition[M]. New York: Wiley Inter-science, 1991.
[3] Jasonal Silverman, Gail L. Rosen, Steve Essinger. Application in Digital Image Processing[J]. Mathematics Teacher, 2013.
[4] Henriques J F, Caseiro R，Martins P, et al. High speed tracking with kernelized correlation filters[J]. IEEE Transaction on Pattern Analysis and Machine Intelligence 2015.
[5] Jun Gao, Zhao Xie. Image understanding theory and method[M]. Beijing：Science Press, 2009.
[6] Montero A S, Lang J, LAGANIÈRE R. Scalable kernel correlation filter with sparse feature integration[C]. Proceedings of IEEE International Conference on Computer Vision Workshop. Santiago: IEEE, 2015:587-594.
[7] Jianjun Hao, Zitao Liu, Qiang Jiang, et al. Measurement of cavity volume based on machine vision[J]. Advanced Materials Research, 2011.
[8] Na Zhang. Study on the volume measurement method of irregular object based on computer vision[D]. ShaanXi: Shaanxi University of Science & Technology, 2015.
[9] Guoqing Wei, Hirzinger G. Parametric shape from shading by radial basis function[J]. IEEE Transactions on Pattern Analysis and Machine Intelligence, 1997.

Appendix

Image preprocessing

```
void pretreatment()
{

    Mat backSrc = imread("D:\\makeModel\\images\\kcf_1\\0610.bmp", 0);

    equalizeHist(backSrc, backSrc);


    for (int i = 497; i <= 610; i++)
    {
        Mat imageSrc = imread("D:\\makeModel\\images\\kcf_1\\" + format("%04d",
i) + ".bmp", 0);

        equalizeHist(imageSrc, imageSrc);

        GaussianBlur(imageSrc, imageSrc, Size(3, 3), 0, 0);

        Mat imageCanny, imageSobel, imageCS;
        Mat x_grad, y_grad, imageED;

        Canny(imageSrc, imageCanny, 30, 60);


        Sobel(imageSrc, x_grad, CV_16S, 1, 0, 3);


        Sobel(imageSrc, y_grad, CV_16S, 0, 1, 3);

        imageSobel = x_grad + y_grad;
        i
       imageCanny = imread("D:\\makeModel\\images\\canny\\" + format("%04d", i)
+ ".bmp", 0);
        imageSobel = imread("D:\\makeModel\\images\\sobel\\" + format("%04d", i)
+ ".bmp", 0);

        imageCS = 0.5 * imageCanny + 0.5 * imageSobel;


        Mat imageGrayCS = imageSrc + 0.2 * imageCS;
```

```
        Mat element = getStructuringElement(MORPH_RECT, Size(3, 3));
        morphologyEx(imageSrc, imageED, MORPH_GRADIENT, element);


        Mat imageGrayED = imageSrc + 1 * imageED;

}
```

KCF tracking algorithm

```
KCFTracker::KCFTracker(bool hog, bool fixed_window, bool multiscale, bool lab)
{

    // Parameters equal in all cases
    lambda = 0.0001;
    padding = 2.5;
    //output_sigma_factor = 0.1;
    output_sigma_factor = 0.125;


    if (hog) {        // HOG
        // VOT
        interp_factor = 0.012;
        sigma = 0.6;
        // TPAMI
        //interp_factor = 0.02;
        //sigma = 0.5;
        cell_size = 12;
        _hogfeatures = true;

        if (lab) {
            interp_factor = 0.005;
            sigma = 0.4;
            //output_sigma_factor = 0.025;
            output_sigma_factor = 0.1;

            _labfeatures = true;
            _labCentroids = cv::Mat(nClusters, 3, CV_32FC1, &data);
            cell_sizeQ = cell_size*cell_size;
        }
        else{
            _labfeatures = false;
```

```
            }
        }
        else {      // RAW
            interp_factor = 0.075;
            sigma = 0.2;
            cell_size = 1;
            _hogfeatures = false;

            if (lab) {
                printf("Lab features are only used with HOG features.\n");
                _labfeatures = false;
            }
        }


        if (multiscale) { // multiscale
            template_size = 96;
            //template_size = 100;
            scale_step = 1.05;
            scale_weight = 0.95;
            if (!fixed_window) {
                //printf("Multiscale does not support non-fixed window.\n");
                fixed_window = true;
            }
        }
        else if (fixed_window) {    // fit correction without multiscale
            template_size = 96;
            //template_size = 100;
            scale_step = 1;
        }
        else {
            template_size = 1;
            scale_step = 1;
        }
}

// Initialize tracker
void KCFTracker::init(const cv::Rect &roi, cv::Mat image)
{
        _roi = roi;
        assert(roi.width >= 0 && roi.height >= 0);
        _tmpl = getFeatures(image, 1);
        _prob = createGaussianPeak(size_patch[0], size_patch[1]);
        _alphaf = cv::Mat(size_patch[0], size_patch[1], CV_32FC2, float(0));
```

```
        //_num = cv::Mat(size_patch[0], size_patch[1], CV_32FC2, float(0));
        //_den = cv::Mat(size_patch[0], size_patch[1], CV_32FC2, float(0));
        train(_tmpl, 1.0); // train with initial frame
    }
// Update position based on the new frame
cv::Rect KCFTracker::update(cv::Mat image)
{
        if (_roi.x + _roi.width <= 0) _roi.x = -_roi.width + 1;
        if (_roi.y + _roi.height <= 0) _roi.y = -_roi.height + 1;
        if (_roi.x >= image.cols - 1) _roi.x = image.cols - 2;
        if (_roi.y >= image.rows - 1) _roi.y = image.rows - 2;

        float cx = _roi.x + _roi.width / 2.0f;
        float cy = _roi.y + _roi.height / 2.0f;


        float peak_value;
        cv::Point2f res = detect(_tmpl, getFeatures(image, 0, 1.0f), peak_value);

        if (scale_step != 1) {
                // Test at a smaller _scale
                float new_peak_value;
                cv::Point2f new_res = detect(_tmpl, getFeatures(image, 0, 1.0f / scale_step),
new_peak_value);

                if (scale_weight * new_peak_value > peak_value) {
                        res = new_res;
                        peak_value = new_peak_value;
                        _scale /= scale_step;
                        _roi.width /= scale_step;
                        _roi.height /= scale_step;
                }

                // Test at a bigger _scale
                new_res = detect(_tmpl, getFeatures(image, 0, scale_step), new_peak_value);

                if (scale_weight * new_peak_value > peak_value) {
                        res = new_res;
                        peak_value = new_peak_value;
                        _scale *= scale_step;
                        _roi.width *= scale_step;
                        _roi.height *= scale_step;
                }
        }
```

```
    // Adjust by cell size and _scale
    _roi.x = cx - _roi.width / 2.0f + ((float) res.x * cell_size * _scale);
    _roi.y = cy - _roi.height / 2.0f + ((float) res.y * cell_size * _scale);

    if (_roi.x >= image.cols - 1) _roi.x = image.cols - 1;
    if (_roi.y >= image.rows - 1) _roi.y = image.rows - 1;
    if (_roi.x + _roi.width <= 0) _roi.x = -_roi.width + 2;
    if (_roi.y + _roi.height <= 0) _roi.y = -_roi.height + 2;

    assert(_roi.width >= 0 && _roi.height >= 0);
    cv::Mat x = getFeatures(image, 0);
    train(x, interp_factor);

    return _roi;
}


// Detect object in the current frame.
cv::Point2f KCFTracker::detect(cv::Mat z, cv::Mat x, float &peak_value)
{
    using namespace FFTTools;

    cv::Mat k = gaussianCorrelation(x, z);
    cv::Mat res = (real(fftd(complexMultiplication(_alphaf, fftd(k)), true)));

    //minMaxLoc only accepts doubles for the peak, and integer points for the coordinates
    cv::Point2i pi;
    double pv;
    cv::minMaxLoc(res, NULL, &pv, NULL, &pi);
    peak_value = (float) pv;

    //subpixel peak estimation, coordinates will be non-integer
    cv::Point2f p((float)pi.x, (float)pi.y);

    if (pi.x > 0 && pi.x < res.cols-1) {
        p.x += subPixelPeak(res.at<float>(pi.y, pi.x-1), peak_value, res.at<float>(pi.y, pi.x+1));
    }

    if (pi.y > 0 && pi.y < res.rows-1) {
        p.y += subPixelPeak(res.at<float>(pi.y-1, pi.x), peak_value, res.at<float>(pi.y+1, pi.x));
```

```
	}

	p.x -= (res.cols) / 2;
	p.y -= (res.rows) / 2;

	return p;
}

// train tracker with a single image
void KCFTracker::train(cv::Mat x, float train_interp_factor)
{
	using namespace FFTTools;

	cv::Mat k = gaussianCorrelation(x, x);
	cv::Mat alphaf = complexDivision(_prob, (fftd(k) + lambda));

	_tmpl = (1 - train_interp_factor) * _tmpl + (train_interp_factor) * x;
	_alphaf = (1 - train_interp_factor) * _alphaf + (train_interp_factor) * alphaf;


	/*cv::Mat kf = fftd(gaussianCorrelation(x, x));
	cv::Mat num = complexMultiplication(kf, _prob);
	cv::Mat den = complexMultiplication(kf, kf + lambda);

	_tmpl = (1 - train_interp_factor) * _tmpl + (train_interp_factor) * x;
	_num = (1 - train_interp_factor) * _num + (train_interp_factor) * num;
	_den = (1 - train_interp_factor) * _den + (train_interp_factor) * den;

	_alphaf = complexDivision(_num, _den);*/

}

// Evaluates a Gaussian kernel with bandwidth SIGMA for all relative shifts between input images X and Y, which must both be MxN. They must also be periodic (ie., pre-processed with a cosine window).
cv::Mat KCFTracker::gaussianCorrelation(cv::Mat x1, cv::Mat x2)
{
	using namespace FFTTools;
	cv::Mat c = cv::Mat( cv::Size(size_patch[1], size_patch[0]), CV_32F, cv::Scalar(0) );
	// HOG features
	if (_hogfeatures) {
		cv::Mat caux;
		cv::Mat x1aux;
```

```
            cv::Mat x2aux;
            for (int i = 0; i < size_patch[2]; i++) {
                x1aux = x1.row(i);     // Procedure do deal with cv::Mat multichannel bug
                x1aux = x1aux.reshape(1, size_patch[0]);
                x2aux = x2.row(i).reshape(1, size_patch[0]);
                cv::mulSpectrums(fftd(x1aux), fftd(x2aux), caux, 0, true);
                caux = fftd(caux, true);
                rearrange(caux);
                caux.convertTo(caux,CV_32F);
                c = c + real(caux);
            }
        }
        // Gray features
        else {
            cv::mulSpectrums(fftd(x1), fftd(x2), c, 0, true);
            c = fftd(c, true);
            rearrange(c);
            c = real(c);
        }
        cv::Mat d;
        cv::max(( (cv::sum(x1.mul(x1))[0] + cv::sum(x2.mul(x2))[0])- 2. * c) / (size_patch[0]*size_patch[1]*size_patch[2]) , 0, d);

        cv::Mat k;
        cv::exp((-d / (sigma * sigma)), k);
        return k;
}

// Create Gaussian Peak. Function called only in the first frame.
cv::Mat KCFTracker::createGaussianPeak(int sizey, int sizex)
{
        cv::Mat_<float> res(sizey, sizex);

        int syh = (sizey) / 2;
        int sxh = (sizex) / 2;

        float output_sigma = std::sqrt((float) sizex * sizey) / padding * output_sigma_factor;
        float mult = -0.5 / (output_sigma * output_sigma);

        for (int i = 0; i < sizey; i++)
            for (int j = 0; j < sizex; j++)
            {
```

```
                int ih = i - syh;
                int jh = j - sxh;
                res(i, j) = std::exp(mult * (float) (ih * ih + jh * jh));
            }
        return FFTTools::fftd(res);
}

// Obtain sub-window from image, with replication-padding and extract features
cv::Mat KCFTracker::getFeatures(const cv::Mat & image, bool inithann, float scale_adjust)
{
        cv::Rect extracted_roi;

        float cx = _roi.x + _roi.width / 2;
        float cy = _roi.y + _roi.height / 2;

        if (inithann) {
            int padded_w = _roi.width * padding;
            int padded_h = _roi.height * padding;

            if (template_size > 1) {    // Fit largest dimension to the given template size
                if (padded_w >= padded_h)    //fit to width
                    _scale = padded_w / (float) template_size;
                else
                    _scale = padded_h / (float) template_size;

                _tmpl_sz.width = padded_w / _scale;
                _tmpl_sz.height = padded_h / _scale;
            }
            else {    //No template size given, use ROI size
                _tmpl_sz.width = padded_w;
                _tmpl_sz.height = padded_h;
                _scale = 1;
                // original code from paper:
                /*if (sqrt(padded_w * padded_h) >= 100) {      //Normal size
                    _tmpl_sz.width = padded_w;
                    _tmpl_sz.height = padded_h;
                    _scale = 1;
                }
                else {      //ROI is too big, track at half size
                    _tmpl_sz.width = padded_w / 2;
                    _tmpl_sz.height = padded_h / 2;
                    _scale = 2;
                }*/
```

```
            }

            if (_hogfeatures) {
                // Round to cell size and also make it even
                _tmpl_sz.width = ( ( (int)(_tmpl_sz.width / (2 * cell_size)) ) * 2 * cell_size ) + cell_size*2;
                _tmpl_sz.height = ( ( (int)(_tmpl_sz.height / (2 * cell_size)) ) * 2 * cell_size ) + cell_size*2;
            }
            else {    //Make number of pixels even (helps with some logic involving half-dimensions)
                _tmpl_sz.width = (_tmpl_sz.width / 2) * 2;
                _tmpl_sz.height = (_tmpl_sz.height / 2) * 2;
            }
        }

        extracted_roi.width = scale_adjust * _scale * _tmpl_sz.width;
        extracted_roi.height = scale_adjust * _scale * _tmpl_sz.height;


        extracted_roi.x = cx - extracted_roi.width / 2;
        extracted_roi.y = cy - extracted_roi.height / 2;

        cv::Mat FeaturesMap;
        cv::Mat z = RectTools::subwindow(image, extracted_roi, cv::BORDER_REPLICATE);

        if (z.cols != _tmpl_sz.width || z.rows != _tmpl_sz.height) {
            cv::resize(z, z, _tmpl_sz);
        }


        if (_hogfeatures) {
            IplImage z_ipl = z;
            CvLSVMFeatureMapCaskade *map;

            getFeatureMaps(&z_ipl, cell_size, &map);
            normalizeAndTruncate(map,0.2f);
            PCAFeatureMaps(map);

            size_patch[0] = map->sizeY;
            size_patch[1] = map->sizeX;
            size_patch[2] = map->numFeatures;
```

```
            FeaturesMap                                                    =
cv::Mat(cv::Size(map->numFeatures,map->sizeX*map->sizeY), CV_32F, map->map);
// Procedure do deal with cv::Mat multichannel bug

           FeaturesMap = FeaturesMap.t();

            freeFeatureMapObject(&map);


            if (_labfeatures) {
                  cv::Mat imgLab;
                  cvtColor(z, imgLab, CV_BGR2Lab);
                  unsigned char *input = (unsigned char*)(imgLab.data);

                  // Sparse output vector
                  cv::Mat          outputLab          =          cv::Mat(_labCentroids.rows,
size_patch[0]*size_patch[1], CV_32F, float(0));

                  int cntCell = 0;
                  // Iterate through each cell
                  for (int cY = cell_size; cY < z.rows-cell_size; cY+=cell_size){
                        for (int cX = cell_size; cX < z.cols-cell_size; cX+=cell_size){
                              // Iterate through each pixel of cell (cX,cY)
                              for(int y = cY; y < cY+cell_size; ++y){
                                    for(int x = cX; x < cX+cell_size; ++x){

                                          float l = (float)input[(z.cols * y + x) * 3];
                                          float a = (float)input[(z.cols * y + x) * 3 + 1];
                                          float b = (float)input[(z.cols * y + x) * 3 + 2];


                                          float minDist = FLT_MAX;
                                          int minIdx = 0;
                                          float *inputCentroid = (float*)(_labCentroids.data);
                                          for(int k = 0; k < _labCentroids.rows; ++k){
                                                float  dist  =  (  (l  -  inputCentroid[3*k])  *  (l  -
inputCentroid[3*k]) )
                                                      + ( (a - inputCentroid[3*k+1]) *
(a - inputCentroid[3*k+1]) )
                                                      + ( (b - inputCentroid[3*k+2]) *
(b - inputCentroid[3*k+2]) );
                                                if(dist < minDist){
                                                      minDist = dist;
```

```
                                        minIdx = k;
                                    }
                                }
                                // Store result at output
                                outputLab.at<float>(minIdx, cntCell) += 1.0 /
cell_sizeQ;
                                //((float*) outputLab.data)[minIdx *
(size_patch[0]*size_patch[1]) + cntCell] += 1.0 / cell_sizeQ;
                            }
                        }
                        cntCell++;
                    }
                }

                size_patch[2] += _labCentroids.rows;
                FeaturesMap.push_back(outputLab);
            }
        }
        else {
            FeaturesMap = RectTools::getGrayImage(z);
            FeaturesMap -= (float) 0.5; // In Paper;
            size_patch[0] = z.rows;
            size_patch[1] = z.cols;
            size_patch[2] = 1;
        }

        if (inithann) {
            createHanningMats();
        }
        FeaturesMap = hann.mul(FeaturesMap);
        return FeaturesMap;
}


void KCFTracker::createHanningMats()
{
        cv::Mat hann1t = cv::Mat(cv::Size(size_patch[1],1), CV_32F, cv::Scalar(0));
        cv::Mat hann2t = cv::Mat(cv::Size(1,size_patch[0]), CV_32F, cv::Scalar(0));

        for (int i = 0; i < hann1t.cols; i++)
            hann1t.at<float > (0, i) = 0.5 * (1 - std::cos(2 * 3.14159265358979323846 *
i / (hann1t.cols - 1)));
        for (int i = 0; i < hann2t.rows; i++)
            hann2t.at<float > (i, 0) = 0.5 * (1 - std::cos(2 * 3.14159265358979323846 *
```

```
i / (hann2t.rows - 1)));

        cv::Mat hann2d = hann2t * hann1t;
        // HOG features
        if (_hogfeatures) {
                cv::Mat hann1d = hann2d.reshape(1,1); // Procedure do deal with cv::Mat multichannel bug

                hann = cv::Mat(cv::Size(size_patch[0]*size_patch[1], size_patch[2]), CV_32F, cv::Scalar(0));
                for (int i = 0; i < size_patch[2]; i++) {
                        for (int j = 0; j<size_patch[0]*size_patch[1]; j++) {
                                hann.at<float>(i,j) = hann1d.at<float>(0,j);
                        }
                }
        }

        else {
                hann = hann2d;
        }
}


float KCFTracker::subPixelPeak(float left, float center, float right)
{
        float divisor = 2 * center - right - left;

        if (divisor == 0)
                return 0;

        return 0.5 * (right - left) / divisor;
}
```

Edge feature extraction

```
void drawSio2()
{
      vector <int> grith(111, 0);
      vector <int> area(111, 0);

      for (int i = 497; i <= 610-3; i++)
      {
            Mat imageSrc = imread("D:\\makeModel\\images2\\sio2\\" + format("%04d", i) + ".bmp", 0);
```

```
        threshold(imageSrc, imageSrc, 0, 255, THRESH_OTSU);
        imwrite("D:\\makeModel\\images2\\sio2\\" + format("%04d", i) + ".bmp",
imageSrc);
        vector <int> drawBegin(imageSrc.cols, 0);
        vector <int> drawBeginBool(imageSrc.cols, 0);
        vector <int> drawEnd(imageSrc.cols, 0);
        vector <int> drawEndBool(imageSrc.cols, 0);


        for (int j = 0; j < imageSrc.cols; j++)
        {
            int begin = 0, end = imageSrc.rows - 1;
            for (int k = 0; k < imageSrc.rows; k++)
            {
                if (imageSrc.at<uchar>(k, j) == 255)
                {
                    drawBegin[j] = k;
                    break;
                }
            }
            for (int k = imageSrc.rows - 1; k >= 0; k--)
            {
                if (imageSrc.at<uchar>(k, j) == 255)
                {
                    drawEnd[j] = k;
                    break;
                }
            }
        }

        for (int j = 1; j < imageSrc.cols - 1; j++)
        {
            int maxBegin = max(max(drawBegin[j - 1], drawBegin[j]), drawBegin[j
+ 1]);
            int minBegin = min(min(drawBegin[j - 1], drawBegin[j]), drawBegin[j +
1]);
            drawBegin[j] = drawBegin[j - 1] + drawBegin[j] + drawBegin[j + 1] -
maxBegin - minBegin;


            int maxEnd = max(max(drawEnd[j - 1], drawEnd[j]), drawEnd[j + 1]);
            int minEnd = min(min(drawEnd[j - 1], drawEnd[j]), drawEnd[j + 1]);
            drawEnd[j] = drawEnd[j - 1] + drawEnd[j] + drawEnd[j + 1] - maxEnd -
minEnd;
```

```
            }



            for (int j = 0; j < imageSrc.cols; j++)
            {
                for (int k = drawBegin[j]; k < drawEnd[j]; k++)
                    imageSrc.at<uchar>(k, j) = 255;
            }


            Mat element = getStructuringElement(MORPH_RECT, Size(7, 7));
            morphologyEx(imageSrc, imageSrc, MORPH_CLOSE, element);

            area[i - 497] = calpilex(imageSrc);


            Mat imageErode;
            Mat imageSrcAdd = Mat(imageSrc.rows + 20, imageSrc.cols + 20, CV_8UC1,
Scalar(0, 0, 0));


            Mat ROI = imageSrcAdd(Rect(10, 10, imageSrc.cols, imageSrc.rows));
            imageSrc.copyTo(ROI);



            element = getStructuringElement(MORPH_RECT, Size(3, 3));

            morphologyEx(imageSrcAdd, imageErode, MORPH_ERODE, element);
            imageSrcAdd = imageSrcAdd - imageErode;
            imageSrc = imageSrcAdd(Rect(10, 10, imageSrc.cols,
imageSrc.rows)).clone();

            grith[i - 497] = calpilex(imageSrc);
        }

}
```

SFS

```
file_path1 = 'F:\QQ \sio2Equal\';
image_path_list1 = dir(strcat(file_path1,'*.bmp'));
img_num1 = length(image_path_list1);
```

```
file_path2 = 'F:\QQ \sio2Draw\';
image_path_list2 = dir(strcat(file_path2,'*.bmp'));
img_num2 = length(image_path_list2);
sums = zeros(img_num1,1);
for num=1:img_num2
    image1 = imread(strcat(file_path1,image_path_list1(num).name));
    image2 = imread(strcat(file_path2,image_path_list2(num).name));
    [x,y]=size(image1);
    iter=100;
    Sx=0.001;
    Sy=0.001;
    Sz=1.0;
    [szy,szx]=size(image1);
    Zn=zeros(szy,szx);
    Zn1=zeros(szy,szx);
    Si1=zeros(szy,szx);
    Si=zeros(szy,szx);
    Wn = 0.001*0.001; %(regularization constraint)
    for i=1:szy
        for j=1:szx
            Zn1(i,j)=0.0;
            Si1(i,j)=0.01;
        end
    end

    Ps = Sx/Sz;
    Qs = Sy/Sz;
    PQs = 1 + Ps*Ps + Qs*Qs;
    E = double(image1);
    d = max(max(E));
    for I=1:iter
        E=double(image1);
        for i=1:szy
            for j=1:szx
                if (j-1<1) || (i-1 <1) || (j==szx) || (i==szy)
                    p=0.0;
                    q=0.0;    %±ßÉÏµÄÒ»È¦¶ùãµÄpºÍqÖµÎª0
                else
                    p = Zn1(i,j)-Zn1(i,j-1);
                    q = Zn1(i,j)-Zn1(i -1,j);
                end
                pq = 1.0 + p*p + q*q;
        %                   PQs = 1 + Ps*Ps + Qs*Qs;
                E(i,j)=E(i,j)/d;
```

```
                    fZ = -1.0*(E(i,j) - max(0.0,(1 + p*Ps + q*Qs)/(sqrt(pq)*sqrt(PQs))));
                    dfZ = -1.0 *max(1.0,((Ps+Qs)/(sqrt(pq)*sqrt(PQs)) - ( (p+q)*(1 + p*Ps + q*Qs)/(sqrt(pq*pq*pq)*sqrt(PQs)))));
                      Y = fZ + dfZ*Zn1(i,j);

                    k = Si1(i,j)*dfZ/(Wn+dfZ*Si1(i,j)*dfZ);

                    Si(i,j) = (1.0 - k*dfZ)*Si1(i,j);

                    Zn(i,j) = Zn1(i,j) + k*(Y-dfZ*Zn1(i,j));
                end
            end

            for i=1:szy
                for j =1:szx

                    Zn1(i,j)=Zn(i,j);
                    Si1(i,j)=Si(i,j);
                end
            end
        end
        zmap = Zn1;
        mi = min(min(zmap));
        resu = zmap-mi;
        for i=1:szy
            for j=1:szx
                if(image2(i,j)<128)
                    resu(i,j)=0;
                end
                sums(num,1) = sums(num,1) + resu(i,j)*1;
            end
        end
end
```