ABSTRACT

LIU, LONGJU. Precise Optical Positioning Using an Inexpensive USB Microscope Camera. (Under the direction of Dr. John F. Muth).

The availability of inexpensive Universal Serial Bus(USB) Charge Coupled Device(CCD) and Complementary Metal Oxide Semiconductor(CMOS) cameras with simple interfaces for image acquisition and image processing potentially allow other instruments to be constructed. This thesis describes using a USB digital microscope, consisting of a CCD camera with a high power adjustable lens, to investigate techniques to make inexpensive positioning systems with micron ($1x\ 10^{-6}$ m) accuracy.

The first method uses the microscope to image an off-the-shelf micrometer scale and digitally process the image to obtain positions with accuracy comparable to commercial Mitutoyo digital micrometers based on linear variable transformers.

The second method images a digital paper with a coded pattern of closely spaced dots to determine an x-y position. The coding pattern is the one developed and described by the company patents of Anoto. The digital microscope was used to take an image of the dot pattern for positioning. The absolute coordinate of a 6×6 array of dots is determined by the spatial encoding. Image processing techniques are used to determine the relative position between the camera and the paper with the printed dot pattern. This method potentially allows absolute two-dimensional position to be obtained with micrometer accuracy.

Precise Optical Positioning Using an Inexpensive USB Microscope Camera

by
Longju Liu

A thesis submitted to the Graduate Faculty of
North Carolina State University
in partial fulfillment of the
requirements for the degree of
Master of Science

Electrical Engineering

Raleigh, North Carolina

2012

APPROVED BY:

_____	_____
John Muth Committee Chair	Russell Philbrick
_____	_____
Leda Lunardi	Michael Escuti

DEDICATION

To my parents

BIOGRAPHY

Longju Liu, was born in Chengdu, China. He received his Bachelor's degree of Science in Optical Information Science and Technology from Sun Yat-sen University, Guangzhou, China. In August of 2009, he began his Master of Science studies in Electrical Engineering at North Carolina State University.

ACKNOWLEDGMENTS

I would like to thank my advisor, Professor John Muth for guiding my work. He is a considerate and enthusiastic mentor and he has been helping me overcome difficulties throughout my master study. I gained more than just knowledge under his advisement. I would also like to thank the other committee members, Professors Russell Philbrick, Leda Lunardi and Michael Escuti for serving on my committee and for their contributions to my thesis. I also like to thank Prof. Edward Grant for attending my defense.

At last, I would also like to thank my parents in China. Without their inspiration and support I could not have finished my Master's Degree studies.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1 INTRODUCTION

Determining the position of one object in relation to reference point is fundamental in many engineering tasks. This thesis describes the investigation of optical positioning methods using an inexpensive camera. These optical methods have the advantage of being precise and non-contact.

A common method of optical positioning uses optical encoders, which can convert mechanical movement into representative electric signals [1-3]. The signals are coded in a predetermined way that can be processed and decoded using a computer program. Optical encoders are especially useful when the movement of the object of interest is simple and clear, for example, to record how far a robot has moved forward or backward[4,11]. The details about several types of optical encoders are discussed in Chapter 2.

When the movement of the object of interest becomes unpredictable, or is in two or three dimensions, it is not always easy to tell the position using optical encoders. Usually, a number of different types of optical positioning sensors are used to form a complicate positioning system to provide the positional information accurately and precisely [5-7].

For the 2D case we can consider using mapping as an analogy, like the GPS (Global Positioning System) determination of latitude and longitude[8], I would like to be able to obtain coordinates for an optically flat surface. The idea is to encode positional information on a surface and then be able to read those positions optically.

Instead of using multiple complicated and expensive sensors to 'read' the information, we utilize a microscopic digital camera [9] to 'read' the coordinate information. The most significant advantages include the ease of obtaining images and the dramatic drop in cost; the possibility of relatively high resolution and accuracy is maintained at the same time. On the other hand, computation power and storage space for images has been sharply increased to processing digital images.

Conventional Optical encoders are accurate and quick [1,2,10], but they are expensive due to the precision required during manufacturing[10]. They can also be limited functionally, for example, some encoders are only useful for obtaining relative position, and require measurement reinitialized if the reference position is lost[1,2]. The resolution of optical encoders is often limited to a couple of microns [12]. Considering a simple digital microscope composed of a lens and CCD or CMOS imaging chip, a magnification that covers a field of view of about 100 μm in a 1280 x 1024 pixel camera, results in about 0.2 μm per pixel, and suggests that sub-micron resolution could be obtained.

It is quite straightforward to think about patterning the surface of interest to convert it into a 2-D map. The method that we explore was invented by Anoto Group, known as Anoto Dot Pattern Technology [13]. It is found to work well in digital papers, where the absolute position of pattern is obtained by an electronic pen, which contains a camera that reads the encoded position of the dots [14,15].

CHAPTER 2 REVIEW OF PRECISE POSITIONING METHODS

There are several common methods for precise positioning. Each method has its own field of applications, where it has specific advantages and disadvantages. For example, one of the simplest precise positioning devices is a micrometer. It is very useful when the movement is in one dimension. Many translation stages are made that use a micrometer in order to provide accurate movement, are usually read by the human eye. Veriner micrometers using the English system are typically accurate to 10000th of an inch, or about 2.5 μm. Some metric vernier calipers can be read to an accuracy of about 1 μm [16].

Interferometers are another device that can be used for precise positioning, the Michelson Interferometer[17,37] for example. The movement of the object is related to the optical path difference of a split light of laser beam. The interference fringes observed in optical intensity change as the object moves; the bright constructive interference change to dark destructive interference as the optical path changes by one-half wavelength[37]. Changes in the length of one arm of the instrument can be counted using a photodiode as each intensity change represents an increment in motion of one-quarter of a wavelength. For more precise measurements one can measure the relative phase by monitoring the intensity within one fringe. With this type approach with appropriate control electronics allows sub-angstrom linear travel to be monitored.

Optical encoders are also very popular because of their ease of use, and these will be introduced in the following paragraphs. Optical encoders are usually electro-mechanical

devices which convert the mechanical movement, or the position information of a mechanical unit, into representative electric signals by blocking and unblocking light, or by the reflection of light [1-3]. The common components of an optical encoder are a light source, a condenser lens, a scanning reticle, a patterned disk and photovoltaic cells [18].

There are several ways of classifying optical encoders, which depending on whether the mechanical movement is angular or linear, and are classified as rotary or linear optical encoders[18,19]. Each of these two types of optical encoders can be further sub-divided into two main types, known as absolute and incremental [1,2].

2-1 Rotary optical encoders

Rotary optical encoders are utilized to convert angular motion or position of a shaft. If a rotary encoder is designed to code the angular motion, it is called an incremental rotary encoder. The output of this type of encoder usually provides information which can be further processed into measured quantities, such as velocity, position, RPM and distance[1]. Another type of rotary encoder is the absolute rotary encoder, which provides an output of the current position of the shaft and is usually made an angle transducer[1]. Both of these two types are similar in construction and are made largely in the same manner[18,19]. The main differences lie in the pattern on the disk and the method of interpreting the code.

2-1-1 Incremental rotary encoder

One typical incremental rotary encoder has the structure shown in Figure 2-1.

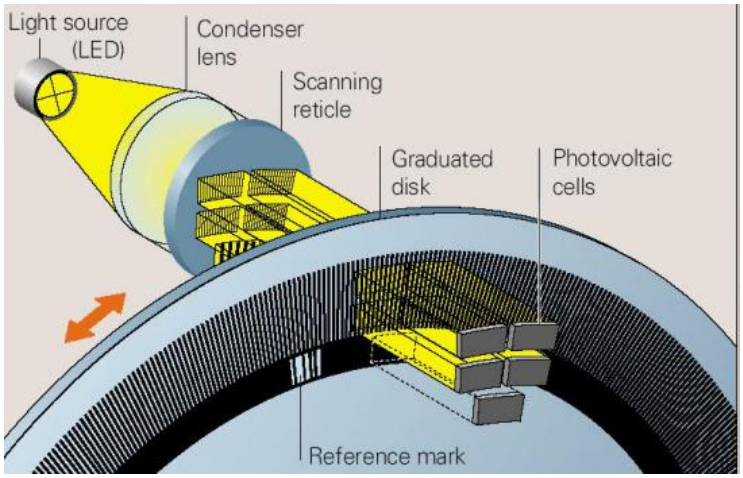


Figure 2-1 The structure of an incremental rotary encoder[18]

The pattern on the periphery of the disk of an incremental rotary encoder is made up with a single track of equally spaced transparent and opaque sectors. By counting the numbers of switching of the transparent and opaque sectors, the motion of disk is measured. The number of such switches that occur per revolution is defined as disk count of the incremental rotary encoder [1].

A transparent section as a reference mark is added to the inner side of the track in order to generate a signal that occurs once per revolution. This signal is used to indicate an absolute position as an index. There are two sensors used in a rotary encoder, which are labeled A and B respectively and shown in Figure 2-3. The outputs of sensor A and sensor B creates two signal trains called channel A and channel B, see Table 2-1 [1,2].

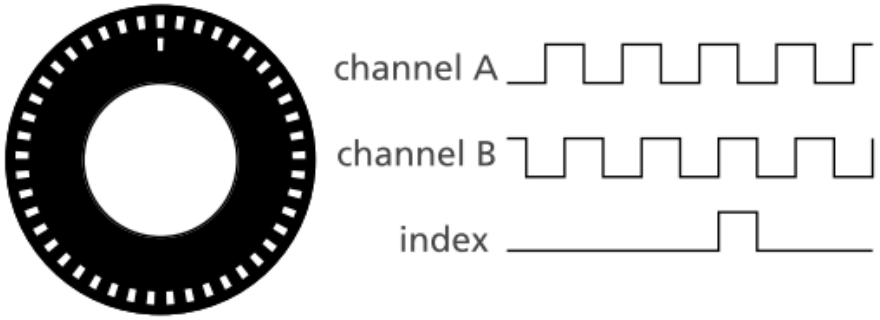


Figure 2-2 Illustration of the disk[1]

The sensor B is shifted a quarter of the pitch of a transparent/opaque section pair from the sensor A so that they are accurately 90 degrees out of phase. There are several reasons that two sensors are used, but the main one is that one output is incapable of determining the direction of rotation. With an additional output, the processing unit is then capable of telling which channel is leading the other one and the rotation direction is determined too [3].

For most encoders, the clockwise direction is defined as the signal of channel A becomes '1' before that of channel B [3].

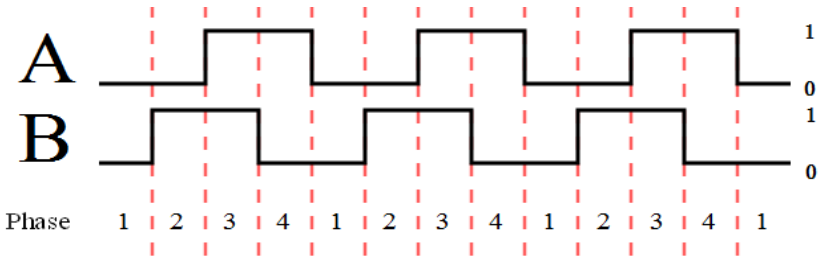


Figure 2-3 The phase change of incremental encoder under clockwise rotation[3]

Table 2-1 Phases of incremental rotary encoder[3]

Clockwise rotation			Counter-clockwise rotation		
Phase	Ch A	Ch B	Phase	Ch A	Ch B
1	0	0	1	1	0
2	0	1	2	1	1
3	1	1	3	0	1
4	1	0	4	0	0

A third signal is generated by the reference mark every complete revolution, and makes it possible for an incremental rotary encoder to determine the angular position.

The incremental rotary encoders are not as accurate as absolute encoders, especially when the rotation speed is too high or too low. Because under these circumstance, misreading errors caused by missing counts or backward counts can easily occur. This would eventually result in a wrong output.

2-1-2 Absolute rotary encoder

A typical absolute rotary encoder has the structure shown in Figure 2-4.

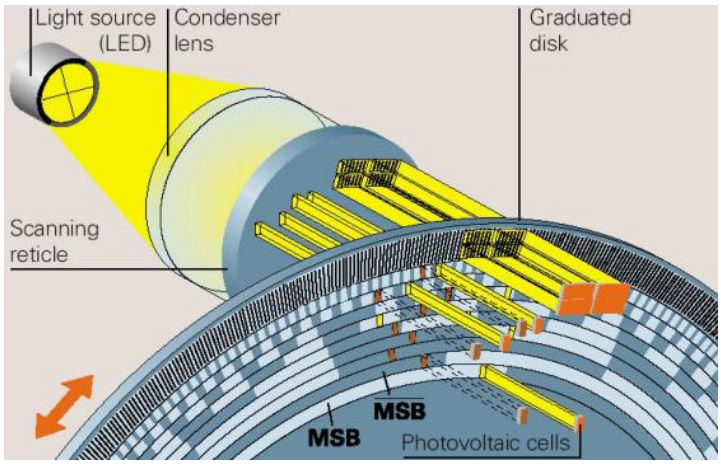


Figure 2-4 The structure of an absolute rotary encoder[18]

Figure 2-4 shows the most significant difference of an absolute rotary encoder and an incremental rotary encoder is that the periphery of the disk is uniquely patterned. The pattern converts the rotation of the shaft into a signal called Gray Code[20,42,43]. The reason that Standard Binary Encoding is not used is that determining the angle becomes impossible when the disk stops between two adjacent sectors. And in fact, the contacts cannot be produced to be perfectly aligned, makes the contacts switches from 1 to 0 or 0 to 1 at different times. If the order of switching is Contact 2, Contact 1 and Contact 3, and the starting sector is sector 1 reading (0, 0, 0), the sequence of the codes is shown as: (0,0,0) ->(0,1,0)->(1,1,0)->(1,1,1)->(0,0,1). The corresponding sequence of sectors, 1->3->7->8, is not acceptable since the next sector after sector 1 (0,0,0) should be sector 2 (0,0,1).

A comparison of the Standard Binary Encoding and the Gray Code is shown for a simple 3-contact absolute encoder in Table 2-2.

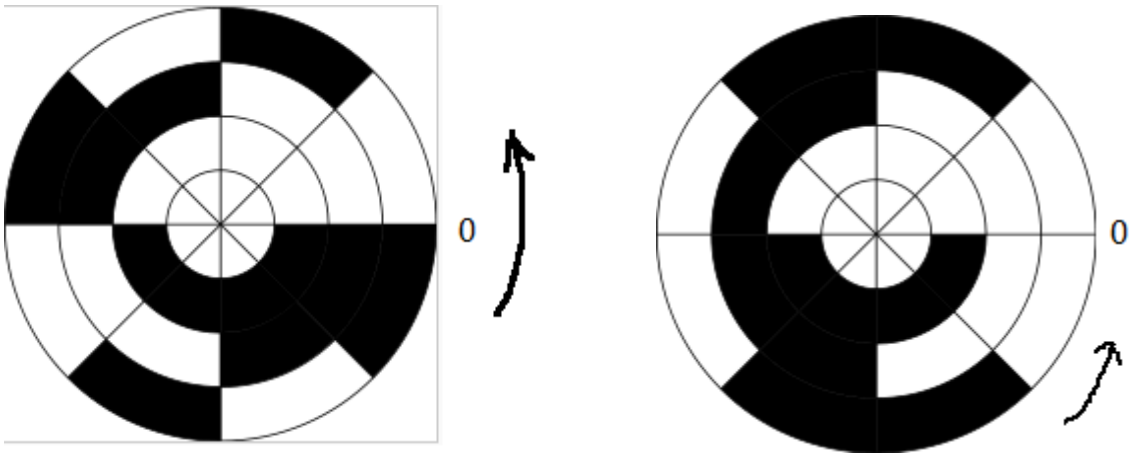


Figure 2-5 (a) A three-contact Standard Binary Encoded disk (b) A three-contact Gray Coded disk[35]

Table 2-2 Comparison of Standard Binary Code and Gray Code[35]

	Standard Binary Encoding			Gray Code			
Sector	Contact 1	Contact 2	Contact 3	Contact 1	Contact 2	Contact 3	Angle(°)
1	0	0	0	0	0	0	0 ~ 45
2	0	0	1	0	0	1	45~ 90
3	0	1	0	0	1	1	90~135
4	0	1	1	0	1	0	135~180
5	1	0	0	1	1	0	180~225
6	1	0	1	1	1	1	225~270
7	1	1	0	1	0	1	270~315
8	1	1	1	1	0	0	315~360

The same problem is examined for the Gray Code. If the order of switching is Contact 2, Contact 1 and Contact 3, and the starting sector is sector 1 reading (0, 0, 0), the sequence of the codes is shown as: (0,0,0) ->(0,0,0)->(0,0,0)->(0,0,0)->(0,0,1). The corresponding

sequence of sectors is 1->1->1->1->2. The change of sector is correct and stable, because there is only one contact that changes state when one sector changes to the next sector in Gray Code.

Actually the pattern on the disk of an absolute array can be rather complicated and divide the disk into many more parts instead of $2^3 = 8$, and requires more contacts.

2-2 Linear Optical Encoders

Linear optical encoders are made to encode linear positions using optical techniques. Like rotary optical encoders, linear optical encoders are divided into two main types—incremental and absolute linear encoders [2,19]. The difference of between incremental linear optical encoders and absolute optical encoders is the reference for the graduations on the track [19].

An incremental linear encoder usually has two tracks. The first track has graduations that are equally distributed, and the second track has a series of special coded graduations as reference marks. A schematic view is shown in Figure 2-6. With the help of the reference marks, it's not difficult to determine the displacement and motion direction using incremental linear encoders.

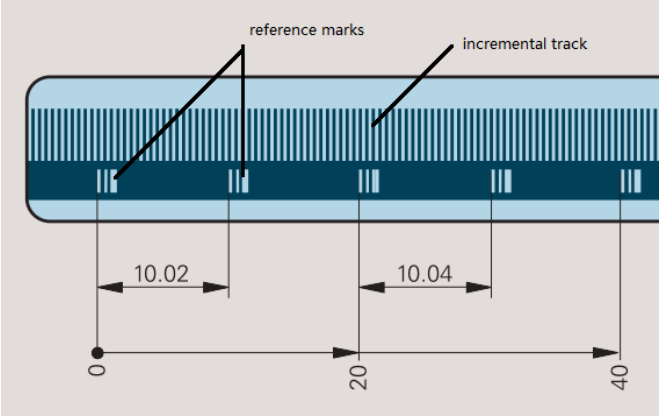


Figure 2-6 A schematic view of the incremental linear encoder [19]

Absolute linear encoders also have two tracks. One track has graduations that are distance-coded, and the other one is incremental with equally distributed graduations. A schematic view is shown in Figure 2-7.

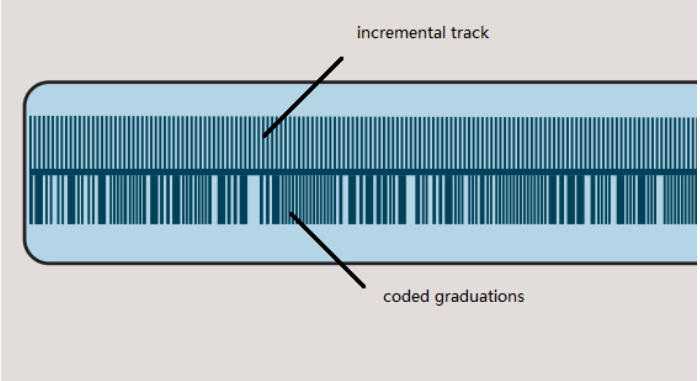


Figure 2-7 A schematic view of the absolute linear [19]

There are several approaches used to code the position using linear encoders. When the movement is relatively large, and the requirement for resolution is relatively low, image

scanning is used [19]. A scanning reticle and a measuring standard scale are placed relative to each other. The scanning reticle is made of transparent materials that parallel light can passes through.

The incident light is modulated by the gaps of these two gratings when the measuring standard is moving. When the gaps are aligned, light is not blocked and can be detected by the photovoltaic cell array. The opposite case occurs when the gaps of one grating is blocked by the line of the other, and no light can pass measuring standard. The illustration is shown in Figure 2-8. On the occasions where the movement is relatively small or high resolution is needed, another method called interferential scanning should be applied. A schematic view is shown in Figure 2-9.

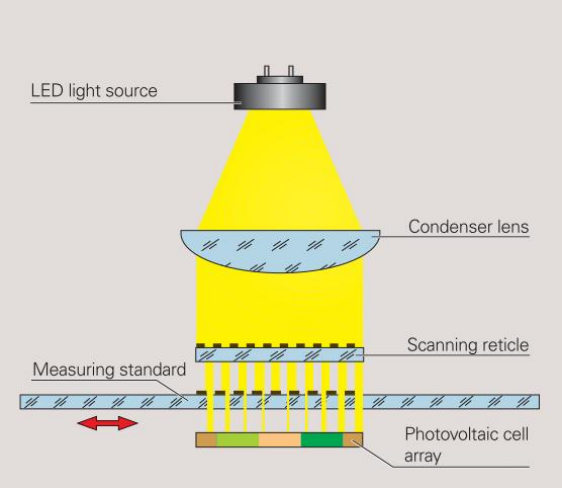


Figure 2-8 Image scanning method [19]

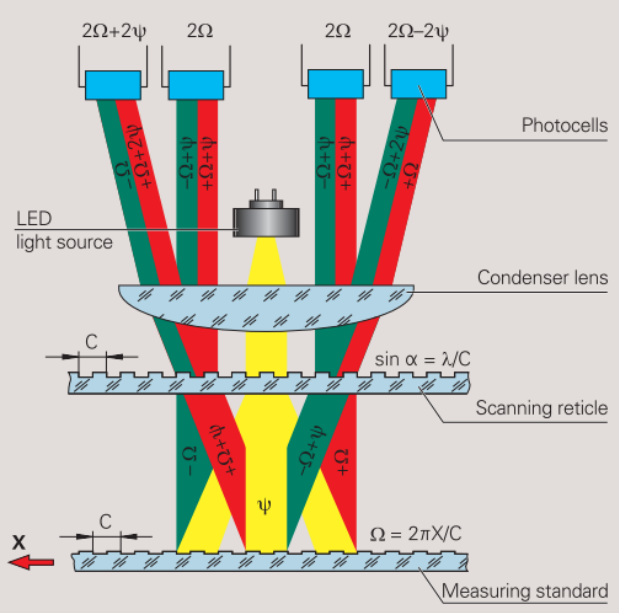


Figure 2-9 Interferential scanning method[19]

When the light passes through the scanning reticle, it is diffracted into three beams with almost equal intensity with the order -1, 0,+1. These beams are diffracted again at the scales of the measuring standard and reflected back to the scanning reticle where interference occurs[19].

If relative motion occurs between the scanning reticle and the measuring standard, there will be phase shifts for the light with order +1 and -1. If the grating moves by one period, the shift of the "+1" light will be 2π in the positive direction while the shift of the "-1" light is 2π in the negative direction. Thus, one period change of the grating causes a change of 2 periods in the signal light.

2-3 Summary

A review of four common optical encoders is presented in this chapter. The purpose is to investigate the features of these encoders and point out their advantages and disadvantages. This examination is to determine if they are suitable for the optical positioning using an ordinary camera.

Each of these optical encoders has favorable properties of high accuracy, high reliability and can code the position without direct contact [18,19]. However they are relatively complex and can require attachment of precision machined encoding strips and multiple detectors. They are also limited to measurement in one dimension and can't freely code the position in higher dimensions.

CHAPTER 3 Obtain the Reading of the Micrometer with a Microscopic Camera

As an example of measuring linear motion we consider using a digital microscope with and off-the-shelf micrometer. The micrometer is widely used for precise measurement of small distance such as moving translation stages and optical components. The principle idea is that by using a digital camera higher accuracy can be obtained rather than reading by eye.

Figure 3-1 The experimental arrangement (left) and the camera used (right)[21]

The experimental arrangement is shown in Figure 3-1, and consists of a digital microscope and bracer to hold the microscope a translation stage with a digital micrometer. The camera is used to take pictures of the scales on the sleeve of the micrometer. The bracer and clamp is used to hold the camera and adjust its position. The translation stage is controlled by a micrometer. It is a Mitutoyo product, Digimatic Micrometer Heads - Series

350 in this thesis[16]. When the thimble of the micrometer is being rotated, the translation stage will move along the same direction. To check the accuracy of our method, the micrometer is also equipped with a digital readout. This readout device uses a linear variable differential transformer with a custom chip that contains calibration data to measure the positions. The rated accuracy of the digital readout is 0.001 mm or 1 μm and are displayed on a small LCD screen [16].

The reading on the LCD screen provides a reference that we compare with our method. The important parameters of the micrometer in this thesis are shown below [16]:

1. Lead: 0.025 inch
2. Range: 1 inch
3. The number of the graduated markings on the thimble: 25, starting from '0' to '24'.
4. The number of the major graduated markings on the sleeve: 11, starting from '0' to '10'.
5. The number of the minor graduated markings on the sleeve: 20 above the axial line, and 40 sub-divisions below the axial line.

From the above parameters, it's clear that if the thimble is screwed out by one turn, the axial movement of the micrometer will be 0.025 inch to the right. And the edge of the thimble will move from the central line of one minor graduation to that of an adjacent minor graduation. Meanwhile, the reading will change by 0.025/25 = 0.001 inch when the thimble

is screwed out by one division on the thimble. If the graduation on the thimble that coincides with the axial line was graduation 0, then it would become graduation 1 after this operation. The reading of the micrometer is determined by how many divisions and sub-divisions are visible on the sleeve, and which graduation on the thimble coincides with the axial line or an estimated position when there is no such a graduation:

$$R = S * 0.02495 + G*0.0010.$$

Where R is Reading, S is the number of complete sub-divisions that can be seen on the sleeve and G is the estimated position on the thimble that coincides with the axial line.

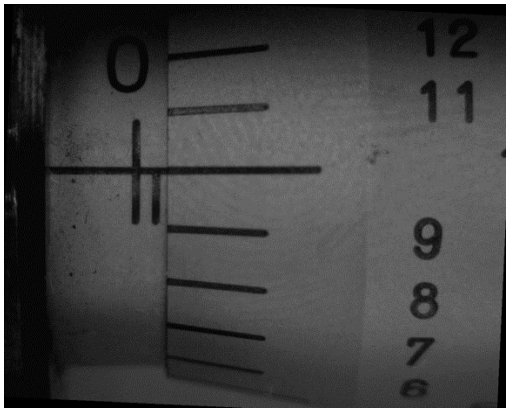


Figure 3-2 Example of experimental image

Thus in our experiments what we want to be able to do is accurately determine an estimate for the graduation. If we know that value to high accuracy we will have an accurate measurement of position. For example as shown in Figure 3-2, Graduation 0 and one sub-division are visible on the sleeve, Graduation 10 on the thimble is seen to coincide with the axial line, and thus R = 1*0.025 +10*0.001 = 0.035 inch.

In general, the idea is to take a picture of the micrometer (Figure 3-3). Then digitize the image to identify the positions of the rotating horizontal lines, and compare the position of those lines with the horizontal reference line. Then using the knowledge of the spacing of the marks on the micrometer determines the position accurately.

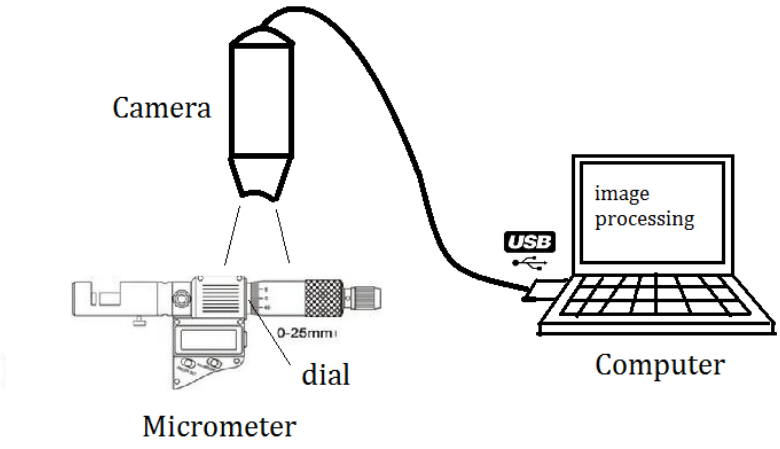


Figure 3-3 Obtaining the reading of a micrometer using a camera

This was accomplished in two steps. First using the LCD readout of the micrometer a series of calibration images were taken for each turn of the thimble from graduation 0 to graduation 24. This establishes a bijection of the axial line and horizontal lines on the thimble. The program ran these 25 images as a batch file and then was used to form a table, where the ration of the distance between the axial line and the horizontal lines corresponds to the linear motion of the micrometer. This process is described in the section titled standard pictures. Using these images, we can then run an experiment where the relative spacing of the lines is

expressed as a ratio. This can then be modified by a calibration factor using the series of calibration images that have been obtained.

3-1 Obtaining standard pictures

A number of images focused on the ROI are taken using the platform shown in Figure 3-1. There are several requirements when taking the images:

1. The images should be taken in the same surroundings with almost the same light source, the same shooting angle and the same distance from the lens;
2. The gray-scale of the images should be as even as possible. Avoid the situation that some parts of the images are too dark or too bright;
3. The images should contain mainly the scales of the micrometer. And the ROI should be made sure as clear as possible.

An example of the obtained images is shown in Figure 3-4.

Figure 3-4 An example reference image of the ROI with a reading of 0.24995

3-2 Processing of the standard images

The processing includes crop, incline correction, threshold, removing noise and converting the image from gray scale to black and white. As long as the images meet the requirements stated above, a batch file can complete the processing quickly for many images at a time. One example of the processed image is shown in Figure 3-4. Unnecessary part of the image has been cropped out. These reference images are then stored for future use.

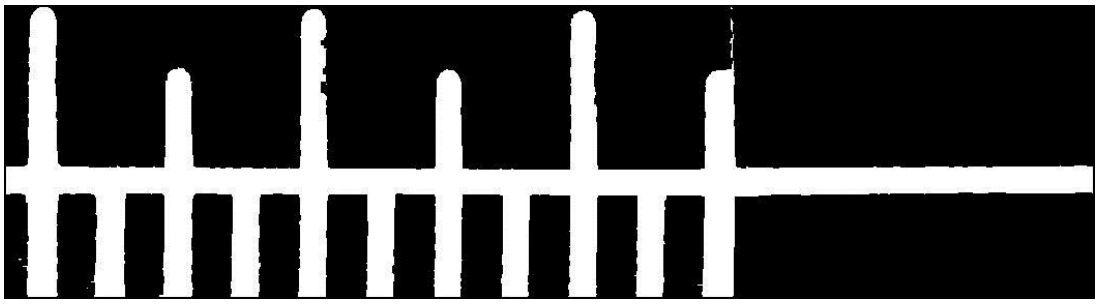


Figure 3-5 A binary image of the processed image

3-3 Calculation of the ratio

The special ratio refers to the ratio of the length of the last interval, which is not a complete sub-division and the average length of the complete sub-divisions. The lengths are obtained in pixels. (Figure 3-5)

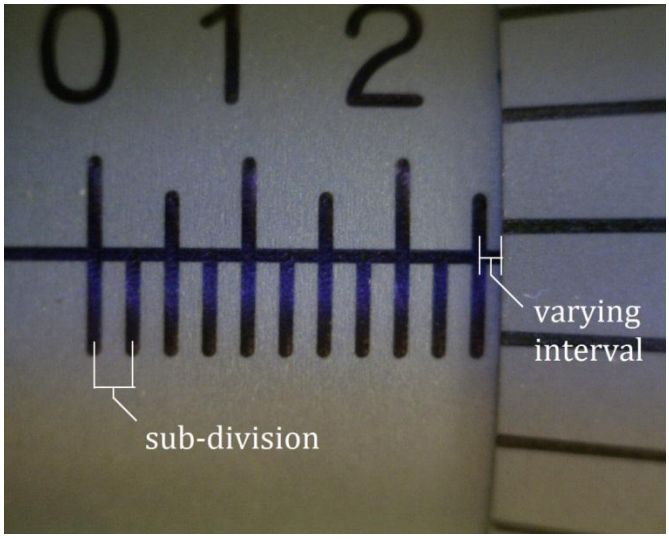


Figure 3-6 A illustration of the sub-division and the varying interval

Continue taking Figure 3-4 as an example, a histogram of the distribution is shown in Figure 3-6. The peaks have shown the locations of the linear graduations on the sleeve. The thin small line shows the position of the edge of the thimble, which has overlapped with the right edge of the last graduation on the sleeve.

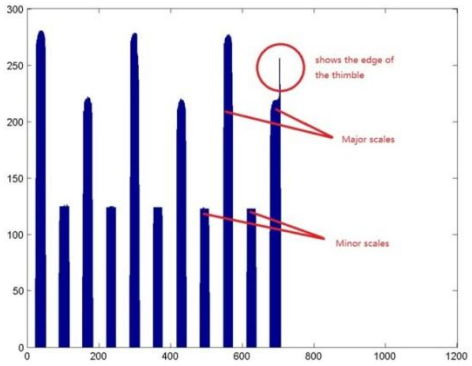


Figure 3-7 An intensity histogram of the reference image

The locations of the linear graduations are shown in Table 3-1. p is the position in pixels.

Table 3-1 Locations of linear graduations of a reference image

NO.	1	2	3	4	5	6	7	8	9	10	11
p	52	117	183	247	313	377	443	508	573	638	706

Use the value of the position of one graduation to subtract that of the former one, the lengths of the sub-divisions are obtained and shown in Table 3-2.

Table 3-2 Lengths of the sub-divisions of a reference image

NO.	1	2	3	4	5	6	7	8	9	10
Length (pixels)	65	66	64	66	64	66	65	65	65	68

The average length of the former 9 sub-divisions is 65.11. The reason why the last sub-division has a larger length is due to the overlapping of the edges of the 11th graduation

and the thimble. Therefore, the position of the last graduation in this image is not reliable. This problem is solved when images were taken for larger readings that the 11th graduation can be distinguished. It is found that the position of the 11th graduation should be 705 by comparison with other images. In this way the last graduation position are corrected for all other similar situations.

As a result, the length of the varying interval is about 1 pixel in this image. The ratio is defined as r,

$$r = \frac{L_{VI}}{L_{SD}} = \frac{1}{65.11} * 100\% = 1.53\%$$

The graduation on the thimble that coincides with the axial line is graduation 0. It is desirable that the ratio is calibrated to 0.00%. Same procedures are done for other 23 reference images and a bijection of the ratio and graduation is obtained and shown in Table 3-3.

Table 3-3 G and corresponding ratio of reference images

Original length	Original ratio	Improved length	Improved ratio	G	Ideal ratio"
1	1.53%	0	0.00%	0	0.00%
4	6.13%	3	4.59%	1	4.00%
6	9.19%	5	7.66%	2	8.00%
8	12.25%	7	10.72%	3	12.00%
11	16.85%	10	15.31%	4	16.00%
14	21.47%	13	19.94%	5	20.00%
15	22.94%	14	21.41%	6	24.00%
18	27.57%	17	26.03%	7	28.00%
21	32.21%	20	30.67%	8	32.00%
23	35.28%	22	33.74%	9	36.00%
27	41.41%	26	39.88%	10	40.00%
31	47.55%	30	46.01%	11	44.00%
33	50.69%	32	49.16%	12	48.00%
35	53.68%	34	52.15%	13	52.00%
38	58.37%	37	56.84%	14	56.00%
40	61.44%	39	59.91%	15	60.00%
43	65.95%	42	64.42%	16	64.00%
46	70.66%	45	69.12%	17	68.00%
48	73.62%	47	72.09%	18	72.00%
51	78.22%	50	76.69%	19	76.00%
54	82.95%	53	81.41%	20	80.00%
57	87.56%	56	86.02%	21	84.00%
60	92.17%	59	90.63%	22	88.00%
64	98.16%	63	96.63%	23	92.00%
66	101.38%	65	99.85%	24	96.00%

The original length and original ratio stands for the interval length and the ratio that obtained from the last step directly. And it's not so accurate and useful as can be seen in the last row of Table 3-4 that the ratio exceeds 100% when graduation 24 coincide with the axial line. We assume that there is a system error and therefore a calibration is needed. The result is the improved length and improved ratio. They are obtained by subtracting 1 pixel from the original length. G is the number of the graduation that coincides with the axial line. The ideal ratio is obtained by dividing G by 25.

The next step is to obtain the reading of an unknown experiment image by interpolating its ratio *r* into Table 3-3.

One processed example image is shown as Figure 3-8.

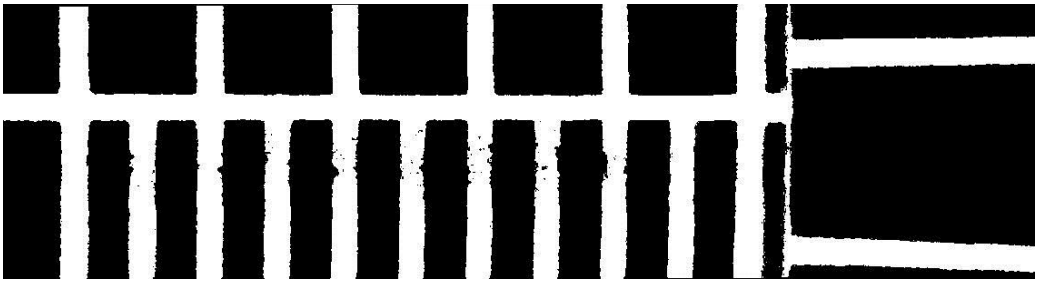


Figure 3-8 A processed example image

The locations of the graduations and the length of the sub-divisions are obtained and shown in Table 3-4.

Table 3-4 The locations of the graduations and the length of the sub-divisions of the experimental image

NO.	1	2	3	4	5	6	7	8	9	10	11	11.X
p	96	169	244	317	392	465	541	614	689	762	840	*870
Length	73	75	73	75	73	76	73	75	73	78	**30	--

* The position of the edge of the thimble

** The length of the varying interval.

The average length of the sub-divisions is 74.4, and the improved ratio is roughly

$$r = \frac{L_{VI}}{L_{SD}} = \frac{30 - \frac{74.4}{65.3} * 1}{74.4} * 100\% = 38.79\%$$

By looking up Table 3-3, G of this image is found to be between 9 and 10. The exactly position is calculated another way. The right part of the image is extracted and shown as Figure 3-8.

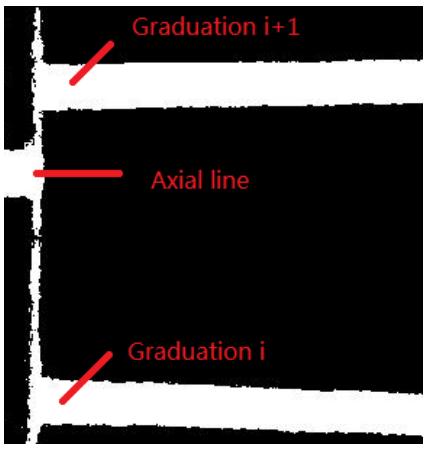


Figure 3-9 The right part used to calculate the position of the axial line

i = 9 in Figure 3-8. As long as the exact position where the axial line coincides with the edge of the thimble is obtained, the G for this image can be determined precisely.

Therefore, another intensity histogram is made and shown in Figure 3-9.

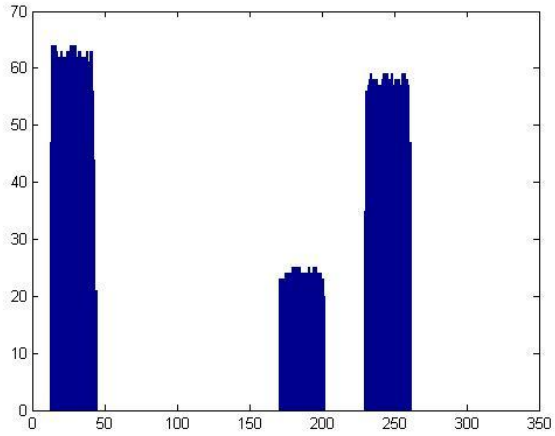


Figure 3-10 The histogram of the right part of the experiment image

The three peaks represent the positions of Graduation 9, the axial line and Graduation 10 respectively.

Table 3-5 Calculation of the precise location of the rotary graduations

	Left edge position (pixel)	Right edge position (pixel)	Mean Value (pixel)
Graduation 9	13	44	28.5
Axial line	171	202	186.5
Graduation 10	230	261	245.5

The ratio of the distance between Graduation 9 and Axial line and the distance between Graduation 9 and Graduation 10, which is defined as r', is found to be:

$$r' = \frac{D_{AL}}{D_{HL}} = \frac{186.5 - 28.5}{245.5 - 28.5} = 0.7281$$

$$G = lower\ graduation + r' = 9 + 0.7281 = 9.7281$$

The final value of the reading is expected to be:

$$Reading_{exp} = 0.02495S + 0.0010G = 0.02495 * 10 + 0.0010 * 9.7281 = 0.25973.$$

The reading on the LCD screen is $Reading_{LCD} = 0.25970$. Take this reading as the reference, the error is

$$error = \frac{Reading_{exp} - Reading_{lcd}}{Reading_{exp}} = \frac{0.25973 - 0.25970}{0.25970} = 0.01\%.$$

It is in good agreement with the digital readout, and demonstrates the principle that high accuracy can be obtained using image processing methods.

Potentially this type of measurement could be integrated into a small package and image processing algorithms could be incorporated on a specialized chip.

CHAPTER 4 ANOTO DOT PATTERN TECHNOLOGY

4-1 Introduction of Anoto Dot Pattern Technology

Anoto Dot Pattern Technology was invented and developed by Anoto AB Group to be used with their product digital pen for electronic paper applications. This technology can establish an absolute position coordinate system to digitize paper by printing a special kind of encoded dot pattern onto them.[13] Thus when one writes on a digital paper, the pen records the coordinates which are then transferred to a computer.[14][15]

The Anoto system is highly developed and commercialized. The information described about their technology is extracted from publicly available information on their website and patents. They also have a software development kit, for third party developers to use their pen. However we thought this only accesses information at a high level. In this portion of the thesis we implement our own method of decoding the dot pattern to obtain the coordinates, and then examine the possibility using this information to obtain precise positioning information.

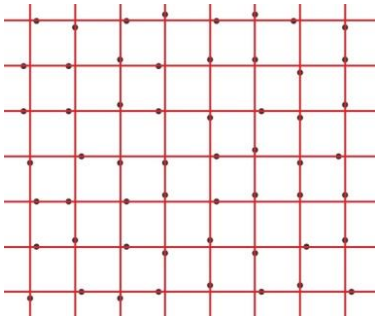


Figure 4-1 Illustration of the Anoto Dot Pattern (with the virtual raster lines)

4-2 Principle of Anoto Dot Pattern Technology

This section summarizes the methods described by Anoto in their patents and is meant to provide enough reference for the reader to understand the MATLAB code in Appendix B.

The dot pattern, which has the form of a grid with a spacing of 0.3 mm, is made up of lots of small black dots. The grid is called a raster in Anoto Patents [13,22] . The raster lines are invisible both to human eye and optical reader. Instead of being located exactly at the intersection of the raster lines (called the "nominal position"), each small dot is displaced slightly near the intersection in one of the four directions: up, down, left and right. One example of the four kinds of dot positions is shown in Figure 4-2 a~d.

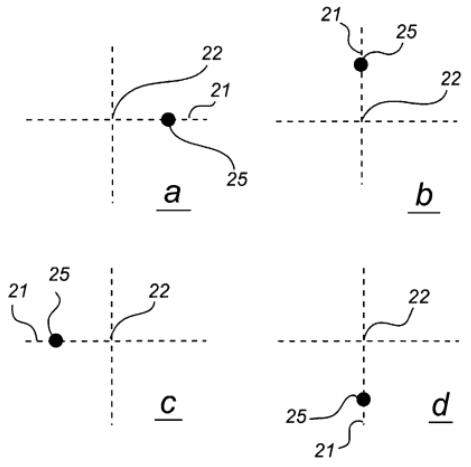


Figure 4-2 Four elements of the Anoto Dot Pattern of four different offset directions[13]

Each kind of the dots, which is called position element, indicates a possible combination of x and y coordinates. Therefore two bits of information are contained in each element, one bit is used to encode a first dimension x, the other is used to encode a second dimension y. An x-code, y-code pair is assigned to each of the four position elements, as shown in Table 4-1.

Table 4-1 Conversion between Offset direction and code pair[13]

Offset direction	x code	y code
Left	1	0
Right	0	1
Up	0	0
Down	1	1

In this way, independent x and y codes can be read simultaneously if given a definite piece of the dot pattern. One example is shown in Figure 4-3.

Figure 4-3 An example of a 8×8 array of Anoto Dot Pattern[13,23]

In order to identify a position, at least a piece of the dot pattern contains 6 ×6 dots is needed [24]. But Anoto Dot Pattern utilizes an 8 ×8 array of dots[13], where the extra two dots are introduced for rotation and error correction purposes. One of the most significant properties that the pattern has is that an array formed by any 8 ×8 dots appears only once all over the entire surface, which makes it possible to identify unique positions.

The reason why the pattern has the above property is that the code which it is encoded into is designed using a special De Bruijn sequence [25,42,47]. This number sequence is cyclic and the length of the sequence is intentionally designed to make any partial sequence of a definite length taken from the main sequence occurs only once. In Anoto's patents[13], the special designed quasi De Bruijin sequence[25] is called the main number sequence (MNS) and has a total of 63 numbers as follows:

0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0,

1, 0, 0, 0, 1, 0, 1, 1, 0, 1, 1, 0, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 0, 0, 1, 1.

Any partial sequence of length 8 has a position value from 0 to 62. For example, partial sequence '0, 0, 0, 0, 0, 0, 1, 0' has the position value '0', and '0, 0, 0, 0, 0, 1, 0, 0, 1', which is '0, 0, 0, 0, 0, 0, 1, 0' shifted right by one, has the position value '1'.

The main number sequence is shifted up and down in columns and left and right in rows by different numbers, and is repeated when it comes to its end. Therefore, the adjacent two columns or two rows have a constant difference number. For example, as for the x direction, If column k contains the main number sequence shifted by l number of rows, and

column k+1 contains the main number sequence shifted by m number of rows, the difference number of these two columns will be (l-m) modulo 63. The difference numbers between columns and rows form another important sequence called primary difference number sequence (PDS). As can be easily observed, although each column and each row start with an identical MNS shifted by different numbers, the PDS does not change from row to row for the x direction, and from column to column for the y direction.

An 8×8 array of this kind of dot pattern has eight columns and eight rows which can be encoded into eight vertical partial sequences and eight horizontal partial sequences with each partial sequences has the length of eight. Each partial sequence then has a corresponding position number for x direction and y direction separately. Besides, two partial PDS of length 7 for the x and y directions can also be obtained separately.

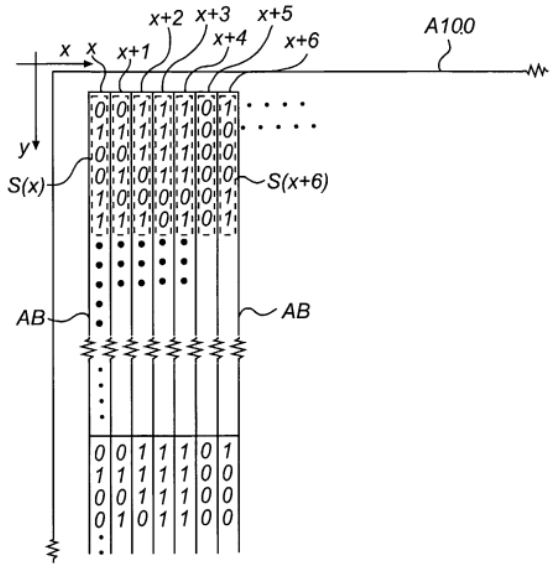


Figure 4-4 How the vertical partial sequences is coded in x direction[13]

What position the partial PDS is in the whole PDS is one of the key pieces of information that tells us about the coordinates. Although the difference numbers are obtained by the differences modulo 63 and are therefore all between 0 and 62, only those with values between 5 and 58, that is 54 = 2*33 numbers are usable. If a difference number is found out of [5, 58], then an error must have occurred.

In practice, the primary difference sequence is further converted into so-called secondary difference number sequences (SDS) for calculation and error-detection and error-correction reasons [17]. For any difference number d belongs to [5, 58], a bijection can be established using rules described below, where a1, a2 and a4 have the values from 0 to 2 and a3 has the values from 0 to 1.

$$d = 5 + a1 + 3 * a2 + 3^2 * a3 + 2 * 3^2 * a4;$$

For example, if d = 35, then d = 5 + 0 + 3 + 9 + 18 = 5 + 0 + 1*3 + 1*32 +1*2*32. So d =35 convert to (a1, a2, a3, a4) is (0, 1, 1, 1). In this way, the primary difference sequence is converted to four secondary difference sequences, as shown in Table 4-5.

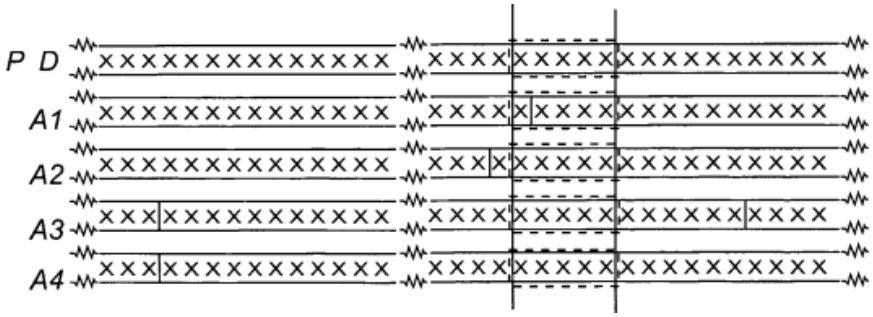


Figure 4-5 Conversion between primary difference number sequence and four secondary difference number sequences[13]

If the position numbers of the partial secondary difference sequences in the whole secondary difference sequences are found, then the position number of the partial PDS can be calculated using Chinese Remainder Theorem[44]. Therefore, four secondary difference number sequences A1~A4 are introduced as below [13].

A1=[0,0,0,0,0,1,0,0,0,0,2,0,1,0,0,1,0,1,0,0,2,0,0,0,1,1,0,0,0,1,2,0,0,1,0,2,0,0,2,0,2,0,1,1,0,
1,0,1,1,0,2,0,1,2,0,1,0,1,2,0,2,1,0,0,1,1,1,0,1,1,1,1,0,2,1,0,1,0,2,1,1,0,0,1,2,1,0,1,1,2,0,0,0,
2,1,0,2,0,2,1,1,1,0,0,2,1,2,0,1,1,1,2,0,2,0,0,1,1,2,1,0,0,0,2,2,0,1,0,2,2,0,0,1,2,2,0,2,02,2,1,
0,1,2,1,2,1,0,2,1,2,1,1,0,2,2,1,2,1,2,0,2,2,0,2,2,2,0,1,1,2,2,1,1,0,1,2,2,2,2,1,2,0,0,2,2,1,1,2,
1,2,2,1,0,2,2,2,2,2,0,2,1,2,2,2,1,1,1,2,1,1,2,0,1,2,2,1,2,2,0,1,2,1,1,1,1,2,2,2,0,0,2,1,1,2,2]

A2=
[0,0,0,0,0,1,0,0,0,0,2,0,1,0,0,1,0,1,0,1,1,0,0,0,1,1,1,1,0,0,1,1,0,1,0,0,2,0,0,0,1,2,0,1,0,1,2,
1,0,0,0,2,1,1,1,0,1,1,1,0,2,1,0,0,1,2,1,2,1,0,1,0,2,0,1,1,0,2,0,0,1,0,2,1,2,0,0,0,2,2,0,0,1,1,2,
0,2,0,0,2,2,2,1,0,1,2,2,0,0,2,1,2,2,1,1,1,1,1,2,0,0,1,2,2,1,2,0,1,1,1,2,1,1,2,0,1,2,1,1,1,2,2,0,
2,2,0,1,1,2,2,2,2,1,2,1,2,2,0,1,2,2,2,0,2,0,2,1,1,2,2,1,0,2,2,0,2,1,0,2,1,1,0,2,2,2,2,0,1,0,2,2,
1,2,2,2,1,1,2,1,2,0,2,2,2]

A3=[0,0,0,0,0,1,0,0,1,1,0,0,0,1,1,1,1,0,0,1,0,1,0,1,1,0,1,1,1,0,1]

A4=[0,0,0,0,0,1,0,2,0,0,0,0,2,0,0,2,0,1,0,0,0,1,1,2,0,0,0,1,2,0,0,2,1,0,0,0,2,1,1,2,0,1,0,1,0,
0,1,2,1,0,0,1,0,0,2,2,0,0,0,2,2,1,0,2,0,1,1,0,0,1,1,1,1,0,1,0,1,1,0,1,2,0,1,1,1,1,0,0,2,0,2,0,1,2,

0,2,2,0,1,0,2,1,0,1,2,1,1,0,1,1,1,2,2,0,0,1,0,1,2,2,2,0,0,2,2,2,0,1,2,1,2,0,2,0,0,1,2,2,0,1,1,2,
1,0,2,1,1,0,2,0,2,1,2,0,0,1,1,0,2,1,2,1,0,1,0,2,2,0,2,1,0,2,2,1,1,1,2,0,2,1,1,1,0,2,2,2,2,0,2,0,
2,2,1,2,1,1,1,1,2,1,2,1,2,2,2,1,0,0,2,1,2,2,1,0,1,1,2,2,1,1,2,1,2,2,2,2,1,2,0,1,2,2,1,2,2,0,2,2,
2,1,1]

They have a length of 236, 233, 31 and 241 respectively. The lengths are chosen relatively prime to each other so that the lowest common multiple of the four sequences will be the product of their lengths. Then the respective places p1, p2, p3 and p4 of the partial sequence of length 5 in these four secondary difference sequences can be determined using the same method of determining the place P of the partial sequence in the primary difference sequence, which has been stated above. And they have a relation listed below:

$$P \equiv p1 \pmod{l1}$$

$$P \equiv p2 \pmod{l2}$$

$$P \equiv p3 \pmod{l3}$$

$$P \equiv p4 \pmod{l4}$$

Where l1, l2, l3, l4 are the lengths of the secondary difference sequences that have values 236, 233, 31, 241 respectively in Anoto's application. As a result, P can be calculated using Chinese Remainder Theorem[26]. In fact, there is no need to calculate the position number P of the partial PDS, since we know the four position numbers p1, p2, p3, p4 of the partial SDS determine P, and then determine x or y coordinates. There is also a bijection

between (p1x, p2x, p3x, p4x) and x, and (p1y, p2y, p3y, p4y) and y, where x, y is the coordinates in x and y directions. Here x stands for the x^{th} column, y stands for the y^{th} row, which gives us the information directly. This is exactly what we utilize in this thesis.

Another factor that is important in Anoto positioning is called the "Section"[13]. Since there are 63 different place numbers for the primary difference sequence, there are 63 possible values which can start the very first x coordinate and first y coordinate respectively. That is in total 63*63 = 3969 different "versions" of the dot pattern[13]. And which two numbers starts the first column and first row are called section number for the x direction and section number for the y direction respectively. This is useful when manage numerous pieces of surface with a huge number of area in sum. Since the surface used in this thesis is far smaller, there is no problem of managing different 'sections' of the surface. Therefore it is unnecessary to calculate the section number. Actually, they are provided as a known parameter in the experiments stated in Chapter 5.

CHAPTER 5 DESIGN OF AN OPTICAL POSITIONING SYSTEM

The design of the optical positioning system consists of the design of a hardware system and a software system. The hardware system contains four devices as shown in Figure 5-1: (a) a translation stage with a micrometer; (b) a Dino Microtouch Camera with an USB connector[21]; (c) a camera bracer; (d) a pedestal.

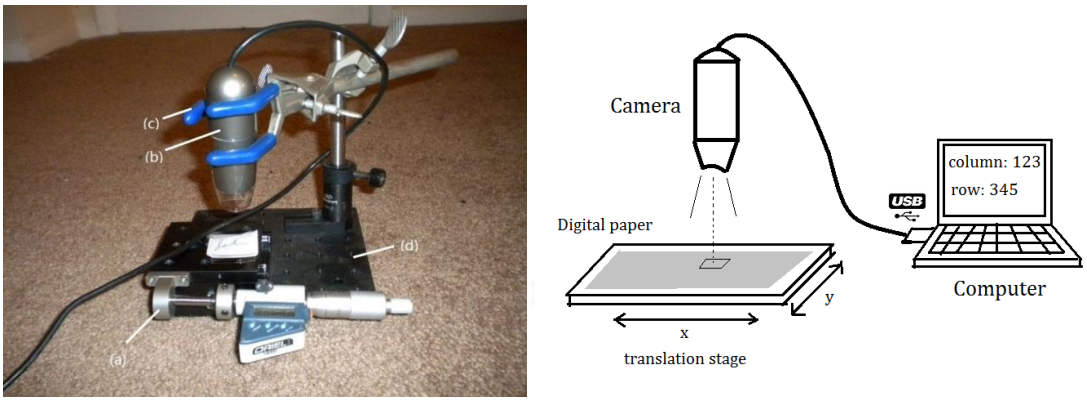


Figure 5-1 The hardware of the optical positioning system

The software system is a program developed using Matlab scripts to process images and make calculations. The entire process of the experiment is schematically shown in the flow chart in Figure 5-2.

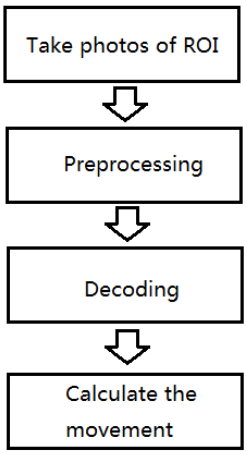


Figure 5-2 Flowchart of the experiment

The Anoto Dot Pattern that we use in this thesis comes from a test paper of their products. We used a Canon Laser jet printer with a high resolution of 1200 dpi to print the dot pattern on to several pieces ordinary paper.

5-1 Image Preparation

An enlarged partial image of the paper with printed dot patterns is shown in Figure 5-3.

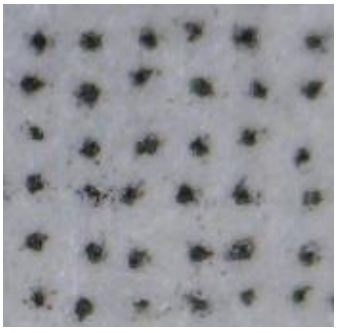


Figure 5-3 Partial image showing the printed 6×6 array at 200X magnification

The number of dots contained in the array in Figure 5-2 is the minimum for enough information to identify a position when it is being decoded. But a typical image taken by the camera could have much more dots as shown in Figure 5-4. Those black curves are written pen strokes. It can be seen that grid spacing of the dot pattern is a little smaller than the width of the pen strokes and the raster lines are not actually printed on the paper.

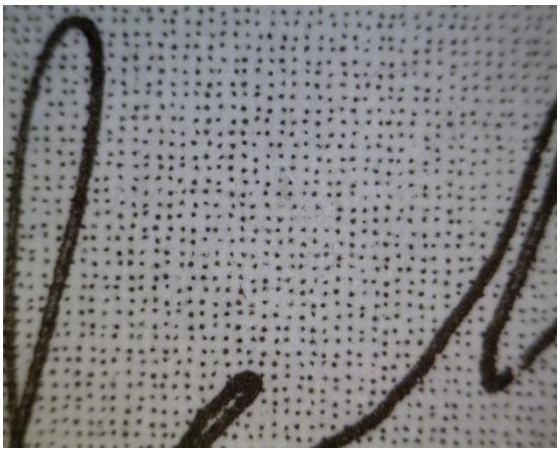


Figure 5-4 A typical image taken by the camera at 60X magnification

5-2 Preprocessing

After the image is taken, the next step is to preprocess it to prepare it for decoding process. The preprocessing section consists of conversion into gray-scale image, thresholding, removing noise and converting to a black and white image. These processes can be done manually to get the best image quality for decoding with a relatively large time cost and human-attention, or automatically using a series of batch scripts with a much smaller time cost which can be applied to many images at a time. Which method to use is mainly

determined by the printing quality of the original image. One set of the Figures showing these four processes is shown in Figure 5-5.

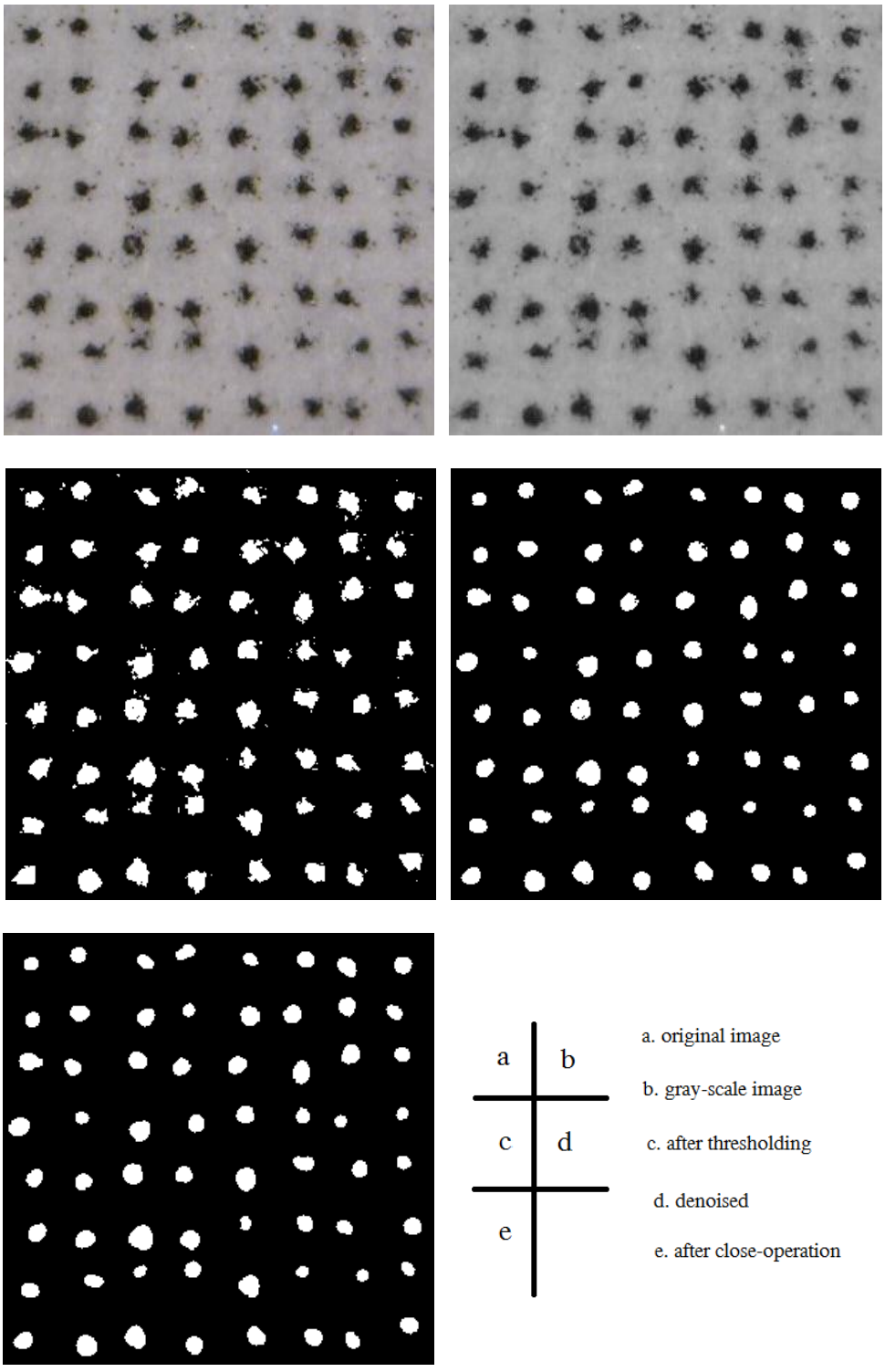
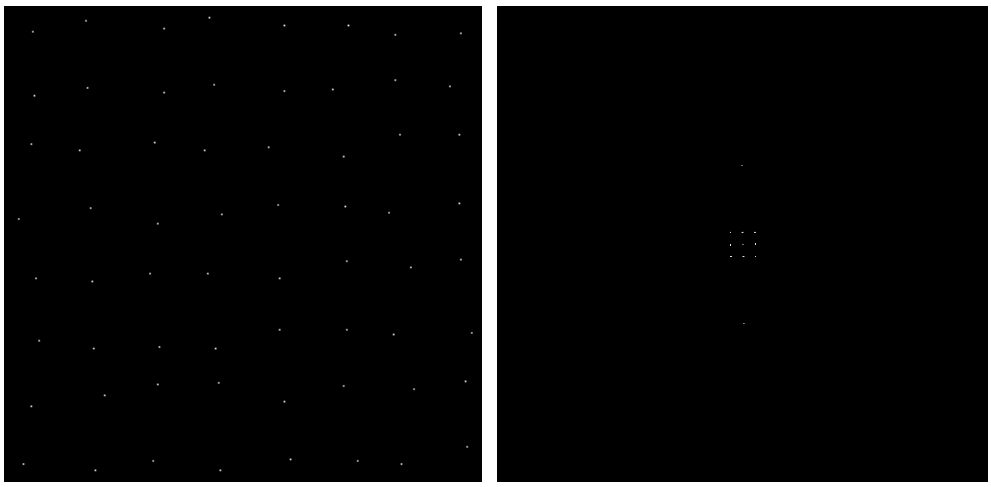


Figure 5-5 The preprocess steps

5-3 Decoding

The image shown as Figure 5-6 (e) is ready for decoding. Firstly, Each blob in the preprocessed image is identified using Matlab scripts[27,28]. Then the blobs will be replaced with their own centroids, or the center of gravity. The coordinates of these centroids are obtained in pixels. Using these coordinates, a new image containing dots of only one pixel is obtained as shown in Figure 5-6 (a).



(a) **(b)**

Figure 5-6 An example image of one-pixel dots and its FFT

Figure 5-6 (b) is a binary image of the Fast Fourier Transform of Figure 5-6 (a). It is thresholded to filter out unnecessary information but the small grid which is made up of nine dots in the middle. The relative position of the nine dots tells us if a rotation correction is needed for the original image[29]. This step is essential to make sure the decoding process goes on smoothly, reducing the chance of occurrences of errors. If the original image is tilted,

the 9-dot grid will look like a parallelogram other than a rectangular. The value of the angle used to correct the rotation can be calculated by using the slopes of the sides of the parallelogram. Then the original image could be rotated back to balance the tilt.

Next, the centroids of the blobs in the image are divided into columns and rows. If there are M×N blobs in the image, the coordinates of the centroids are recorded into two M×N matrices, one for the x-coordinates and the other for the y-coordinates.

As for the x-coordinates, from the relative positions inside a same column, the dots could be divided into 3 kinds: left (L), medium (M), and right (R), depending on which direction the dot is offset in x dimension. If the number of dots is relatively large, there is a big chance that all three kinds of the dots can be found in a single column. But if the number of dots is relatively small, for example only 6~10 dots per column, there is a good chance that only two kinds of the dots (L and M, or M and R) can be found in a column. Sometimes extreme situation happens that only one kind of the dots is found in a single column. Fortunately, the three situations can be easily identified by comparing the width of each column to the mean value of themselves.

In our embodiment, if the width of the column is larger than 2/3 of the mean value, we know the column contains all three kinds of dots. If the width is smaller than 1/3 of the mean value, then the column has probably only one kind of dots. And if the width of the column is between 1/3 and 2/3 of the mean value, then the column should have 2 kinds of dots.

It is easy to find out in which direction the dots are offset for the first kind of column. Since all we need to do is further divide the dots in the column into three kinds. The smallest 1/3 is the left ones, the medium 1/3 the medium ones, the largest 1/3 the right ones.

Things get more complicate when it comes to the latter two kinds of columns. The solution we give is to take the other coordinates into account based on the property that Anoto Dot Pattern has. We know that there are in total four directions the dots can be displaced in, two in x dimension, the other two in y dimension. And we assign a direction indicator as list in the table below:

Table 5-1 Conversion between offset direction and offset indicators

Offset direction	Indicator in x dimension	Indicator in y dimension
Left	-1	0
Right	1	0
Up	0	1
Down	0	-1

As can be seen in Table 5-1, one of the two indicators must be 0, but they cannot be 0 simultaneously. This property is useful that once we cannot determine the offset direction of the dot in one dimension, we can search in another dimension. Usually the problem will be solved.

Once the offset direction of each dot is determined, it is important to make sure it's correct. Since the quality of the printing or image can be low and the image can be distorted. The dots obtained through above procedures might not be located exactly where it should be.

After the error-correction section, we can decode the dot patterns into codes both in x and y direction as shown in Table 4-1. And then we can utilize the x codes and y codes, following the directions stated in Chapter 4 to determine the coordinates of columns and rows. An example of the process is given below. A preprocessed binary image is shown in Figure 5-7.

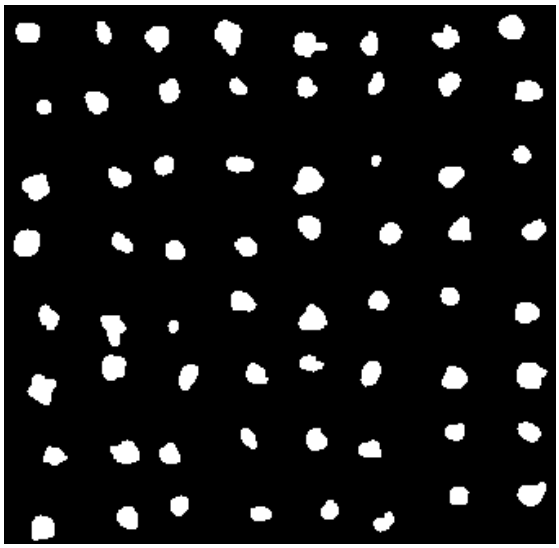


Figure 5-7 A preprocessed image ready for decoding (60X)

The first thing we need to do is to check if the dot pattern contained in this image is inclined. If it is not, then the virtual raster lines should be parallel to the coordinate axis. And this is what we desire. Otherwise, we must find out how much it is inclined and rotate the image to the opposite direction to cancel the effect. Of course, sometimes it is not easy to judge whether the image is inclined and needs rotation-correction by human eye, since the incline angle maybe relatively small. But we can make use of the Fourier Transform, usually

the Fast Fourier Transform of the image and solve this problem in Frequency Domain. The FFT of Figure 5-7 is shown in Figure 5-8 (a) below.

Before the use is made of it, a threshold is made to filter out unnecessary information. This process needs to be done manually to make sure all the information, except the nine dots in the middle of the image, is filtered. These nine dots form a small parallelogram. The angle made by the sides of the parallelogram and the horizontal and vertical directions tells if the virtual raster lines are parallel to the coordinate axis.

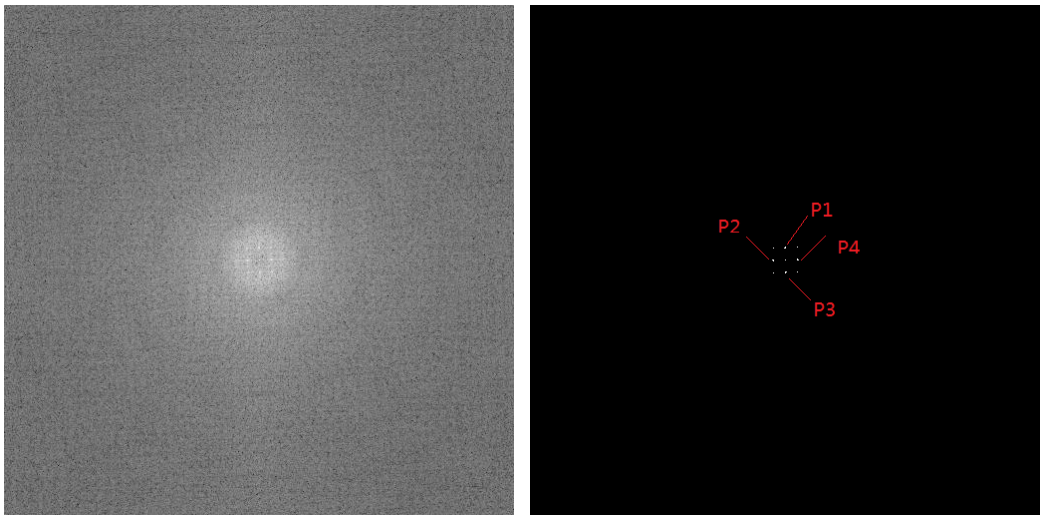


Figure 5-8 Using FFT of the one-pixel dots image to correct the incline

Since the 9-dot parallelogram is made up of 4 smaller parallelograms, we can look into the smaller parallelograms for simplicity. According to the geometry relations of these dots, the calculations are made as follows:

1. Locate the center dot, and its coordinate is obtained as P_c (257.00, 257.00).

2. Find the four nearest dots to the center dot in all the quadrants. It is done by searching the dot that has the smallest Euclidean distance to the center dot in all the four quadrants. And they are P_1(257.33, 269.33), P_2(244.67, 257.33), P_3(256.67, 244.67) and P_4(269.33, 256.67).

3. Determine the four slopes of the straight lines P_1P_c, P_2P_c, P_3P_c and P_4P_c. The result is shown in Table 4-2

Table 5-2 Calculation of the incline angle

	P_1P_c	P_2P_c	P_3P_c	P_4P_c
Slope	0.9996	0.0270	0.9996	0.0270
Angle	88.45 º	1.55 º	88.45 º	1.55 º

Analyze the angles in Table 5-2, and it is found that the dot pattern is 1.55 degree inclined. Rotate the original image -1.55 degree to for rotation-correction. The corrected image is shown as Figure 5-9.

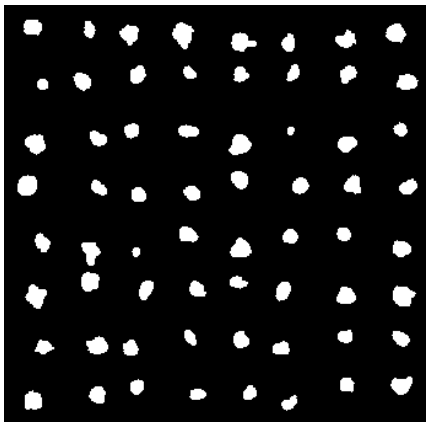


Figure 5-9 The incline-corrected image

Then the x, y coordinates of the centroids are obtained and shown in the Table 5-3:

Table 5-3 x, y coordinates (in pixels) of the centroids (a) the x coordinates (b) the y coordinates

(a) the x coordinates

x =

23.8882	70.1414	102.9944	146.4739	194.4541	232.6148	279.4568	319.9058
32.0571	64.4841	109.5850	151.7176	193.5169	236.4660	280.9653	329.1477
26.2906	77.1894	104.6807	150.8235	192.9598	234.3143	280.1384	323.8421
19.5740	77.7895	110.2787	153.9225	192.7378	242.1329	285.0637	330.1533
31.8074	71.2842	108.3256	150.4902	193.5891	233.9500	277.5943	324.5912
26.6396	70.6021	116.2535	157.9924	191.2804	228.1611	279.4046	325.8442
33.0250	76.8115	103.7752	152.4022	193.6879	226.5610	278.8224	324.1136
24.2951	76.9195	108.9911	158.4141	200.9216	233.3529	280.0526	324.7249

(b) the y coordinates

y =

321.3851	319.4646	315.7360	315.3936	309.2222	308.5164	311.2593	316.5288
274.9571	277.4395	282.7823	284.3059	282.9661	284.1359	283.2708	276.7443
226.6601	230.8409	237.3193	236.8529	225.7098	237.1714	226.5409	238.1158
192.6233	190.7368	185.1639	186.5271	197.5061	192.5175	193.1720	191.4818
146.1556	138.4737	138.6512	152.8039	141.2277	151.2833	153.0849	141.3459
102.6757	114.5916	107.9859	108.3030	113.8785	106.9463	102.8844	102.8095
61.2833	62.3874	60.4961	69.1304	67.2057	60.2927	69.9533	68.3258
17.6721	22.4497	28.9821	22.9394	24.0686	15.7647	30.6930	30.2540

Use the first column as an example. The minimum value of x coordinates is 19.5740, and the maximum value of x coordinates is 33.0250. Thus the difference of these two is $\Delta 1$ = 33.0250-19.5740 = 13.4510. We can calculate the difference of maximum and minimum for every column. The result is shown in Table 5-4.

Table 5-4 Differences of maximum and minimum of each column

Column	Δ_1	Δ_2	Δ_3	Δ_4	Δ_5	Δ_6	Δ_7	Δ_8
Difference	13.4510	13.3054	13.2591	11.9402	9.6412	15.5719	7.4694	10.2475

Therefore, the maximum difference is 15.5719.

1/3 of the maximum difference: 5.1906

2/3 of the maximum difference: 10.3813

The columns that have a difference larger than 10.3813 (Column Class A): Column 1, Column 2, Column 3, Column 4 and Column 6

The columns that have a difference between 5.1906 and 10.3813 (Column Class B): Column 5, Column 7 and Column8.

As for the Columns in Class A, there are three kinds of dots (considering only in x direction, so the dots that offset upwards and downwards count as one 'medium' kind). Again, take the first Column as an example, divide the spacing between the dot with the minimum x coordinate and the dot with the maximum x coordinate into three intervals of equal length in x direction: [19.5740, 24.0577), [24.0577, 28.5414) and [28.5414, 33.0250] . Assign offset indicator '-1' to the dots whose x coordinate falls in the first interval (left ones), '0' to the dots whose x coordinate falls in the second interval (medium ones), and '1' to the dots whose x coordinate falls in the last interval.

As for the Columns in Class B, there are only two possible combinations of offset directions for the dots--the left ones and the medium ones, or the medium ones and the right ones. In this situation, we can first divide the spacing between the dot with the minimum x coordinate and the dot with the maximum x coordinate into two intervals of equal length. Take the fifth column of the array for example, the intervals are [191.2804, 196.1010) and

[196.1010, 200.9216]. Assign offset indicator '-2' to the dots whose x coordinate falls in the first interval and offset indicator '2' to the dots whose x coordinate falls in the second interval. And the offset direction indicators are then obtained and shown in Table 5-5.

Table 5-5 Unmatched Offset direction indicators in x, y directions (a) in the x direction and (b) in the y direction

(a) in the x direction

-1	0	-1	-1	-2	0	-2	-2
1	-1	0	0	-2	0	-2	2
0	1	-1	0	-2	0	-2	-2
-1	1	0	0	-2	1	2	2
1	0	0	0	-2	0	-2	-2
0	0	1	1	-2	-1	-2	2
1	1	-1	0	-2	-1	-2	-2
0	1	0	1	2	0	-2	-2

(b) in the y direction

1	1	0	0	-1	-1	-1	0
-2	-2	2	2	2	2	2	-2
-1	0	1	1	-1	1	-1	1
0	0	-1	-1	1	0	0	0
0	-1	-1	1	-1	1	1	-1
-1	1	0	0	1	0	-1	-1
-2	-2	-2	2	2	-2	2	2
-1	0	1	0	0	-1	1	1

The offset indicator '-2'and '2' are only temporary. They do not tell the actual offset direction of the dots, but just indicate the position relation inside a column. In order to find out the actual offset direction, we need to account for the offset indicator in y direction, which has been shown in Table 5- 5 (b). It can be easily found that the common feature of the columns with '2' and '-2' is that there is no '0' in that column.

Now focus on the offset indicator of column 2 in y direction. They are -1, 2, -1, 1, -1, 1, 2, 0. Together with those in x direction, they are made into pairs as (-2,-1), (-2,2), (-2,-1), (-2,1), (-2, -1), (-2,1), (-2, 2), (2,0). We know that the offset indicators in x and y direction should not be 0 simultaneously, but at least one of them must be 0. Thus, the factor pairs are improved to (0,-1), (-2, 2), (0,-1), (0,1), (0,-1), (0,1), (-2, 2), (2, 0). Then we get an inference that in the second column, the offset indicator '-2' is actually '0', and therefore the offset indicator '2' should be '1'.

Besides, notice that the last pair is (2,0), Because '2' indicates that the dot is in the relatively right half of the dots distribution and its counterpart in y direction is already '0', it must be '1'. This is consistent with the above inference. Finally, the offset indicator for the fifth column in x direction should be: 0, 0, 0, 0, 0, 0, 0, 1. Similar process are done for the other two columns. The updated results of all the columns are shown in Table 5-6 (a). The process for the y direction is similar and the results are shown in Table 5- 6(b). The whole process is like a Sudoku problem but is much easier.

Table 5-6 Offset direction indicators after first correction (a) in the x direction and (b) in the y direction

(a) in the x direction

−1	0	−1	−1	0	0	0	0
1	−1	0	0	0	0	0	1
0	1	−1	0	0	0	0	0
−1	1	0	0	0	1	1	1
1	0	0	0	0	0	0	0
0	0	1	1	0	−1	0	0
1	1	−1	0	0	−1	0	−1
0	1	0	1	1	0	0	0

(b) in y direction

1	1	0	0	−1	−1	−1	0
0	0	1	1	1	1	1	0
−1	0	1	1	−1	1	−1	1
0	0	−1	−1	1	0	0	0
0	−1	−1	1	−1	1	1	−1
−1	1	0	0	1	0	−1	−1
0	0	0	1	1	0	1	0
−1	0	1	0	0	−1	1	1

However, the offset indicators are not finally verified. As can be seen in Table 5-6(a) and (b), the indicator pairs at {1,1},{3,3} and {1,8} are (-1,1), (-1,1) and (0, 0) respectively ({L,M} means the L^{th} row and the M^{th} column). These indicator pairs cannot create correct x-code and y-code. The solution in this thesis is to reconstruct the virtual raster lines [23].

Once the offset indicators are obtained for both x and y directions, it becomes easy to establish the virtual raster lines. The dot pattern with raster lines are shown in Figure 5-10.

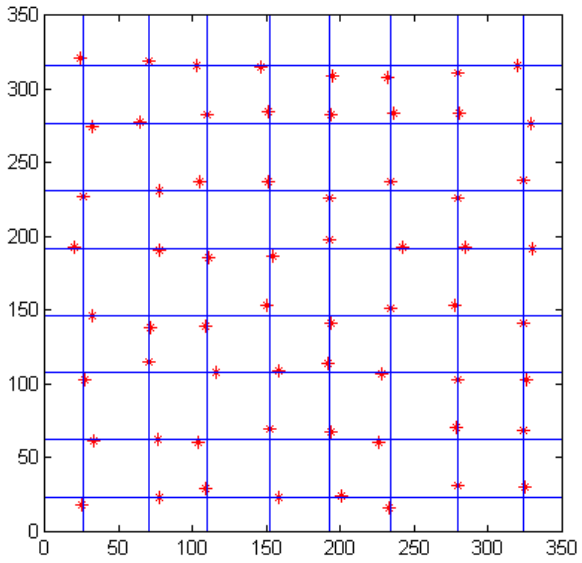


Figure 5-10 The dot pattern with reconstructed visible raster lines

Since the raster lines are obtained through the positions of the dots, which may have been distorted during processing, the raster lines may not equally spaced. However, with the aid of the raster lines, the offset directions of each dot become even clear. And this is a complementary method to make sure offset indicator is correctly obtained. Take the indicator pair at {1,1} for example.

The distance between the dot and the raster line in the x direction:

Dx = 23.8882 - 25.7418 = -1.8536;

The distance between the dot and the raster line in the y direction:

Dy = 321.3851 – 315.8861 = 5.4990;

And for the absolute value, abs(Dy) >abs(Dx).

Thus the offset direction is determined to be in the y direction. The indicator is assigned '1' with the sign determined by the sign of Dy. Therefore, the indicator pairs at {1,1} has been corrected from (-1,1) to (0,1).

Table 5-7 Final offset direction indicators (a) in the x direction and (b) in the y direction

(a) in the x direction

0	0	-1	-1	0	0	0	-1
1	-1	0	0	0	0	0	1
0	1	0	0	0	0	0	0
-1	1	0	0	0	1	1	1
1	0	0	0	0	0	0	0
0	0	1	1	0	-1	0	0
1	1	-1	0	0	-1	0	-1
0	1	0	1	1	0	0	0

(b) in the y direction

1	1	0	0	-1	-1	-1	0
0	0	1	1	1	1	1	0
-1	0	1	1	-1	1	-1	1
0	0	-1	-1	1	0	0	0
0	-1	-1	1	-1	1	1	-1
-1	1	0	0	1	0	-1	-1
0	0	0	1	1	0	1	0
-1	0	1	0	0	-1	1	1

The x, y codes then are obtained by converting the indicator pairs through the bijection shown in Table 4-1. The results are shown in Table 5-9:

Table 5-8 Corresponding x, y codes

xcode =

0	0	1	1	1	1	1	1
0	1	0	0	0	0	0	0
1	0	0	0	1	0	1	0
1	0	1	1	0	0	0	0
0	1	1	0	1	0	0	1
1	0	0	0	0	1	1	1
0	0	1	0	0	1	0	1
1	0	0	0	0	1	0	0

ycode =

0	0	0	0	1	1	1	0
1	0	0	0	0	0	0	1
1	1	0	0	1	0	1	0
0	1	1	1	0	1	1	1
1	1	1	0	1	0	0	1
1	0	1	1	0	0	1	1
1	1	0	0	0	0	0	0
1	1	0	1	1	1	0	0

Then we can decode in two dimensions respectively.

For the x direction, from Table 5-5(a), six vertical partial sequences are obtained as follows: 00110101, 01001000, 10011010, 10010000, 10101000, 10000111, 10100100 and 10001110. The corresponding places of these partial sequences in the main number sequences are

47, 14, 46, 15, 32, 18, 13, 57.

The primary difference sequence is further obtained as

30, 32, 32, 17, 49, 58, 44.

The secondary difference sequences are obtained using Table 5-9:

Table 5-9 An example of converting primary difference number sequence into secondary difference number sequences

d	30	32	32	17	49	58	44
a1	1	0	0	0	2	2	0
a2	2	0	0	1	2	2	1
a3	0	1	1	1	0	1	0
a4	1	1	1	0	2	2	2

The four partial sequences of length 5 are ‘10002’ for A1, ‘20012’ for A2, ‘01110 ’for A3, ‘11102’ for A4. The corresponding place numbers are listed below:

p1 = 119

p2 = 147

p3 = 25

p4 = 178

As long as the section number in x direction is known, or the four such numbers for the first 8×8 array of the dot pattern are obtained, we can assign a column coordinate.

For the first 8×8 array, p1 to p4 are obtained as

p1= 32

p2= 57

p3= 12

p4= 96

Since the p1 to p4 get larger by one consecutively when the region of interest is moved one column right, an infinite table can be established like below:

Table 5-10 The relation between column number and the four place numbers

	Column 1	Column 2	Column 3	…	Column x	…
p1	32	33	34	…	119	…
p2	57	58	59	…	147	…
p3	12	13	14	…	25	…
p4	96	97	98	…	178	…

The ranges of p1, p2, p3 and p4 are 0 to 235, 0 to 232, 0 to 30 and 0 to 240 respectively. Identical combinations of p1, p2, p3 and p4 occur every $l_1 \times l_2 \times l_3 \times l_4 = 236 \times 233 \times 31 \times 241 = 410815348$ columns. That is 410815348×0.3 mm = 123244604.4 mm =123.2 km, which is much larger than the dimension of test papers used in this thesis.

By looking up the Table 5-10, which is established using Matlab, the column corresponding to (119, 147, 25, 178) is found to be column 324, which means that the first column of the 8×8 array is column 324 of the whole surface. So the last column of the 8×8 array is column 331.

A similar process is made for the y direction. Instead of using vertical partial sequences in Table 5-5, horizontal partial sequences are used. They are 00001110, 1000001, 11001010, 01110111, 11101001, 10110011, 11000000 and 11011100. The place numbers in

secondary difference sequences are (50, 87, 11, 207). And the place numbers for the first row of the dot pattern are (224, 19, 4, 155), then a similar table to Table 5-10 is obtained for the y direction, shown as Table 5-11.

Table 5-11 The relation between row number and the four place numbers

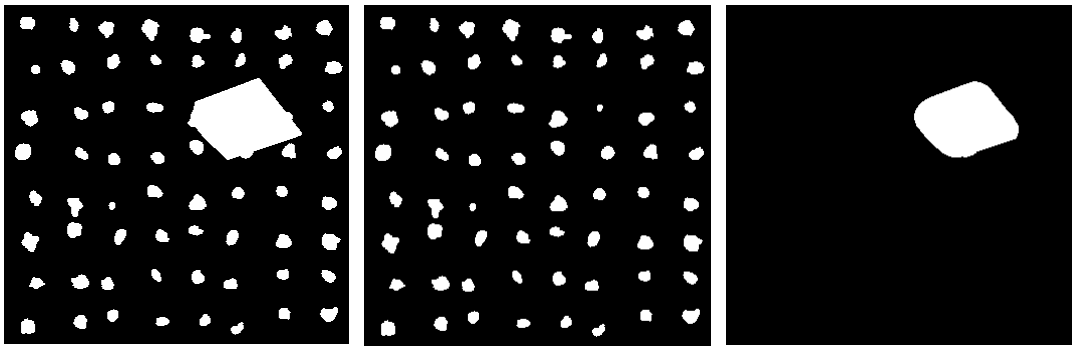
	p1	p2	p3	p4
Row 1	224	19	4	155
Row 2	225	20	5	156
Row 3	226	21	6	157
…	…	…	…	…
Row y	50	87	11	207
…	…	…	…	…

By looking up the Table 5-11, the row coordinate corresponding to (50, 87, 11, 207) is Row 535. Thus the rows that are included in the 8×8 array of interest are Row 535 to Row 542. The decoding is complete and successful.

5-4 Calculate the movement distance

With any 6×6 array or 8×8 array of the dot pattern being successfully decoded, we can quickly get the accurate position information of the object that we're interested in on the digitized surface just by taking pictures of the region of interest [30,31]. If we record the position information of an object at a first position and that of the object at a second position, then the distance that the object translated from the first position to the second one.

As an example we can consider the case of when the position information of the background dot array is known. This is shown in Figure 5-11. This can be a little impractical since it assumes that the image is taken before and after the object is on the page, but it shows in principle how the method can be used. However, in the case of a pen where the dots are reflective in the infrared, and the ink from the pen is transparent to the infrared, this method should work. In both cases- as an extension of this- is if the field of view is wide enough and as long as one 8×8 can be used to obtain the absolute coordinate position, that information can be used to extrapolate information about the occluded dots. This information can be used to subtract dots from the image.



(a) (b) (c)

Figure 5-11 (a) the preprocessed image of the region of interest, with the object (the circle) at its starting position (b) the known dot pattern of the background, which will provide the position of the region (c) extracted from (a), which is a binary image of the object with the position relation unchanged from (a). 60× magnification.

Since we already know the position of the 8×8 array in the whole dot pattern, what we need to do is to locate the object in the array. This can be simplified to locate the centroid of the object in the array. This is done as below.

1. Get the coordinate in pixels of the centroid of the object in the image. In this example it's P_o(238.7545, 224.2730).

2. Compare the coordinate of the dot P_o with the raster lines. And it can be found that the dot is located between the 6th column and 7th column, 3rd row and 4th row.

3. Add a decimal part to the dot's location to indicate its precise coordinate in columns and rows.

   The coordinate of the 6th column is x = 234.1396;

   The coordinate of the 7th column is x = 279.4906;

   The coordinate of the 3rd row is y = 230.8409.

   The coordinate of the 4th row is y = 192.1063.

   Object_Column = (238.7545 – 234.1396)/ (279.4906-238.7545) +6 = 6.1018;

   Object_Row = (224.2730-230.8469)/(192.1063-230.8469) +3 = 3.1696.

4. The column and row coordinates of the centroid in the dot pattern are obtained as:

   C = 6.1018 + 324 -1 =329.1018;

   R = 3.1696 +535 -1 = 537.1696.

The same process can be done when the object has moved to its finishing position. And we can then get another coordinate of its centroid, the coordinate at the finishing position. If the value is (923.2411, 345.5511), then using basic knowledge of vector and Euclidean geometry, we can calculate the distance the object moved as:

$$\text{Distance} = \sqrt{(923.2411 - 329.1018)^2 + (345.5511 - 537.1696)^2} = 624.27$$

This is based on the spacing of the grid in x direction and that of y direction are equal, which of course is not really necessary. Then as we know beforehand, the spacing of the grid in both directions is 0.3 mm, thus Distance 624.27 $\times$ 0.3 mm = 187.3 mm.

CHAPTER 6 CONCLUSION AND FUTURE WORK

6-1 Summary of the work

The purpose of this thesis is to design a platform using an inexpensive camera for optical position measurements. In order to realize the positioning function, several common optical encoders were considered, and a method for reading a micrometer was implemented. The Anoto Dot Pattern was investigated because of its advantages, such as those listed below:

1. Two-dimensional measurement- Anoto Dot Pattern is actually a special 2-D "barcode" that can code a large area and make optical measurements on a plane surface possible.

2. Ease of sample analysis. A laser printer with a resolution higher than 600 dpi is sufficient for the job. The pattern can be printed to any surfaces that are printable.

3. Inexpensive instrument. Images taken by an inexpensive camera are sufficient for processing at useful resolution.

The printed dots are irregular in shape, which leads to errors in centroid position in pixels of the dots. However, each raster line has several dots, so error in raster lines is potentially less than that of an individual dot. The spacing of raster lines is also known, which can be used to potentially improve the accuracy. For a lower magnification image, there are more dots in the field of view that give a higher accuracy, but they also bring a larger number of microns/pixel (about 6.7 μm/pixel at 60X magnification). For a higher

magnification image, since there are fewer dots in the field of view, the ability to do statistics determine absolute accuracy is lost, but relative accuracy within an 8×8 dot array can be high with a smaller number of microns/pixel(about 2.1 μm/pixel at 200X magnification). One possible significant improvement to obtain higher accuracy would be printing of nice circular uniform sized dots. But this also requires a better device with higher printer quality.

6-2 Future Directions

The measurement platform built in this thesis can register the positions of a region of interest accurately (to at least one hundredth millimeter) with the help of Anoto Dot Pattern. By comparison of the position information of a starting position and that of a finishing position, a displacement can also be measured accurately. Since the measurement is non-contact and can detect tiny displacements, it is very appropriate to apply this optical measurement to situations where the sample is small and a non-contact measurement is required.

Future work will implement the system and calibrate it against a standard to obtain a better appreciation of the limit of accuracy that can be obtained. Once the dot pattern is decoded, and the raster patterns are obtained, one has both the absolute position reference and grid, which is a position that is relative to the coordinates system of the camera. While the grid is not equally spaced, which may be due to problems induced during the printing process, or distortions in the image due to the acquisition process, one has considerable information that potentially can be extracted to determine the relative position. Since

typically the field of view contains many dots, (and one knows the intended relative positions and sizes) one can also obtain statistics about the mean spacing and size of the dots which also provide dimensional reference. Presently, with the above method, the variation observed in the spacing of the raster lines is approximately 20 μm. Thus presently, one can obtain the relative position of the paper with the camera to about 20 μm. Taking the above statistical processes into consideration, or by having carefully printed high fidelity patterns prepared, one can reasonably expect micron or even sub-micron positional accuracy.

REFERENCES

[1] Computer Optical Products, Inc, "Optical Encoder Applications," [Online]. Available: http://www.opticalencoder.com/pdf/introduction.pdf.

[2] Encoder Products Company, "Optical Encoder Guide," [Online]. Available: http://www.encoder.com/literature/optical-encoder-guide.pdf.

[3] A. Moore, "Understanding Quadrature Encoding," 26 January 2009. [Online]. Available: http://prototalk.net/forums/showthread.php?t=78.

[4] J. Borenstein, H. Everett, L. Feng and D. Wehe, "Mobile Robot Positioning- Sensors and Techniques," *Journal of Robotic Systems,* vol. 14, no. 4, Special Issue on Mobile Robots, pp. 231-249, 1994.

[5] J. L. David, J. S. Frichtel and L. L. Zuppan, "Optical Positioning System". United States of America Patent 3,727,055, 10 April 1973.

[6] G. J. Raymond, "Optical Positioning Method and System". United States of America Patent 4,991,966, 12 Febrary 1991.

[7] Y.-H. Chen and A.-S. Cheng, "Optical Positioning Apparatus and Positioning Method Thereof". United States of America Patent 2011/0110559 A1, 12 May 2011.

[8] Kowoma, "Position Determination with GPS," Kowoma, 19 April 2009. [Online]. Available: http://www.kowoma.de/en/gps/positioning.htm.

[9] M. Werner, M. Kessel and C. Marouane, "Indoor Positioning Using Smartphone Camera," in *2011 International Conference on Indoor Positioning and Indoor Navigation (IPIN)*, GUIMARAES, PORTUGAL, 2011.

[10] W.-H. Yeh, W. Bletscher and M. Mansuripur, "High Resolution Optical Shaft Encoder for Motor Speed Control Based on an Optical Disk Pick-up," *Review of Scientific Instruments,* pp. 3068-3071, 1998.

[11] Society of Robots, "SENSORS - ROBOT ENCODER," [Online]. Available: http://www.societyofrobots.com/sensors_encoder.shtml.

[12] J. Ganssle, "Encoders provide a sense of place," 19 July 2005. [Online]. Available: http://eetimes.com/discussion/break-points/4025565/Encoders-provide-a-sense-of-place.

[13] M. P. Pettersson, "Method and Device for Decoding a Position-Coding Pattern". World Intellectual Property Organization Patent 03/038741 A1, 2003.

[14] D. Bergstrom and O. Sandstrom, "Method in Electronic Pen, Computer Program Product, and Electronic Pen". United States of America Patent 8,094,139 B2, 22 Februray 2006.

[15] L. Wiebe and P. Ericson, "Method for Transmitting Information". United States of America Patent 7,356,012, 8 April 2008.

[16] Mitutoyo America Corporation, "Product Overview- Digimatic Micrometer Heads Series 350," Mitutoyo America Corporation, [Online]. Available: http://www.mitutoyo.com/TerminalMerchandisingGroup.aspx?group=1546.

[17] D. W. Ball, "Michelson Interferometer," in *Field Guide to Spectroscopy*, Bellingham, WA, SPIE Press, 2006, p. 31.

[18] JOHANNES HEIDENHAIN GmbH, "Rotary Encoders," April 2005. [Online]. Available: http://www.auto-met.com/heidenhain/08PDF/Rotary%20Encoders.pdf.

[19] JOHANNES HEIDENHAIN GmbH, "Linear Encoders for Numerically Controlled Machine Tools," June 2007. [Online]. Available: http://www.heidenhain.com/fileadmin/pdb/media/img/571_470-23.pdf.

[20] A. Hiltgen, K. Paterson and M. Brandestini, "Single-track Gray codes," *Information Theory, IEEE Transactions on,* vol. 42, no. 5, pp. 1555-1561, 1996.

[21] "Dino-Lite am413t," Dino-Lite, [Online]. Available: http://www.onyougo.com/dino-lite-am413t-am413t-microscopes_pi988483.

[22] T. Edso and P. Ericson, "Identification of Virtual Raster Pattern". United Stateds of America Patent 2002/0044138 A1, 18 April 2002.

[23] M. P. Pettersson, "Reconsctruction of Virtual Raster". United States of America Patent 7,543,753 B2, 9 June 2009.

[24] A. Olsson and M. P. Perttersson, "Method and Device for Data Decoding". United States of America Patent 7,195,166 B2, 27 March 2007.

[25] E. Aboufadel, T. Armstrong and E. Smietana, *Position Coding*, Cornell University Library, 2007.

[26] F. T. Howard, "A Generalized Chinese Remainder Theorem," *The College Mathematics Journal*, vol. 33, no. 4, pp. 279-282, 2002.

[27] P. N. Blossey, "Howto: Counting Blobs (and Finding Their Properties) in MATLAB," University of Washington, April 2008. [Online]. Available: http://www.atmos.washington.edu/~bloss/blobcount/.

[28] Xiezhh, "How to Find Four Closest Points Near a Given Point in MATLAB," 29 January 2010. [Online]. Available: http://www.ilovematlab.cn/viewthread.php?tid=64976&sid=PX4f90.

[29] M. P. Pettersson and T. Edso, "Orientation Discrimination in a Position-Coding Pattern". United States of America Patent 7,248,250, 24 July 2007.

[30] A. Olsson, "Method and Device For Identifying Objects in Digital Images". United States of America Patent 7,283,676 B2, 16 October 2007.

[31] A. Bjoklund, "Coding and Decoding of Data". United States of America Patent 7,950,588 B2, 31 May 2011.

[32] I. Zinovik, D. Kroening and Y. Chebiryak, "Computing Binary Combinatorial Gray Codes Via Exhaustive Search With SAT Solvers," *Information Theory, IEEE Transactions on,* vol. 54, no. 4, pp. 1819-1823, 2008.

[33] C. Savage, "A Survey of Combinatorial Gray Codes," *SIAM Review,* vol. 39, no. 4, pp. 605-629, 1997.

[34] J. Salvi, J. Pages and J. Batlle, "Pattern codification strategies in structured light systems," *Pattern Recognition 37,* no. 37, pp. 827-849, 2004.

[35] W. Press, S. Teukolsky, W. Vetterling and B. Flannery, "Section 22.3 Gray Codes," in *Numerical Recipes: The Art of Scientific Computing (3rd ed.)*, New York, Cambridge University Press, 2007, pp. 1166-1168.

[36] R. Penne, "A Note on Certain de Bruijn Sequences with forbidden subsequences," *Discrete Mathematics,* no. 310, pp. 966-969, 2010.

[37] A. Michelson and E. Morley, "On the Relative Motion of the Earth and the Luminiferous Ether," *American Journal of Science,* p. 333–345, 1887.

[38] D. Knuth, "Section 4.3.2," in *The Art of Computer Programming, Volume 2: Seminumerical Algorithms, Third Edition*, Addison-Wesley, 1997, pp. 286-291.

[39] D. Hopp, C. Pruss, W. Osten, J. Seybold, K.-P. Fritz, T. Botzelmann, V. Mayer and H. Kueck, "Diffractive Incremental and Absolute Coding Principle for optical rotary sensors," *Applied Optics,* vol. 50, no. 26, pp. 5169-5177, 2011.

[40] E. R. Hauge and J. Mykkeltveit, "The analysis of De Bruijn Sequences of Non-extremal weight," *Discrete Mathematics,* no. 189, pp. 133-147, 1998.

[41] E. R. Hauge and T. Helleseth, "De Bruijn Sequences, irreducible codes and cyclotomy," *Discrete Mathematics,* no. 159, pp. 143-154, 1996.

[42] F. Gray, "Pulse Code". United States of America Patent 94111237.7, March 1953.

[43] E. Gilbert, "Gray Codes and Paths on the n-cube," *Bell Systems Technical Journal*, vol. 37, pp. 815-826, 1958.

[44] T. H. Cormen, C. E. Leiserson, R. L. Rivest and C. Stein, "Section 31.5: The Chinese remainder theorem," in *Introduction to Algorithms, Second Edition*, MIT Press and McGraw-Hill, 2001, p. 873–876.

[45] A. H. Chan, R. A. Games and E. L. Key, "On the Complexities of Periodic Sequences," *Annals of Discrete Mathematics*, no. 17, pp. 159-170, 1983.

[46] V. Becher and P. A. Heiber, "On Extending de Bruijn Sequences," *Information Processing Letters*, no. 111, pp. 930-932, 2011.

[47] A. Alhakim and M. Akinwande, "A Recursive Construction of nonbinary de Bruijn Sequences," *Des. Codes Cryptogr.,* no. 60, pp. 155-169, 2011.

APPPENDIX

Appendix A Matlab scripts used in Chapter 3

This part contains matlab scripts used in Chapter 3.

Document 1

```
% File name: BATPROCESS.m
% BATPROCESS batch processes the binary images stored in a folder, dir_name,
% plotting intensity histogram to find the locations of the graduations on a sleeve.
% Then save the data into an excel file for establishing Table 3-3.
% Input: dir_name - location of the source images.
%        dir_name2 - location of the output images and excel file.
% Output: xSpacing.xlsx - file contain the spacing information of each
%                                    reference image.
%                    name.jpg - intensity histogram of each reference image.

clc;clear all;
dir_name = uigetdir('E:\EsoneruzA\Desktop\','Choose Source Folder');
if isequal(dir_name,0)
    disp('invalid')
else
    tarFiles = dir(dir_name);
end
dir_name2 = uigetdir('E:\EsoneruzA\Desktop\','Choose Destination Folder');

filename = 'xSpacing.xlsx';
ffn = fullfile(dir_name2, filename);

px = zeros(numel(tarFiles)-2,20);
[rpx, cpx] = size(px);
xSpacing = zeros(rpx,cpx-1);

for k = 3:numel(tarFiles)
    BW = imread(fullfile(dir_name,tarFiles(k).name));
    [r,c] = size(BW);
% The name of the image file is specially named by its reading on the LCD screen.
% This is very convenient when labeling each histogram.
    [~, name, ~] = fileparts(tarFiles(k).name);
    BW =BW>100;
    [n,~] = hist(BW);
    n = n(10,:);

    for i= 1:c
        if n(i)<40||(n(i)<75 && i>820)       % Filter out unnecessary parts
            n(i) = 0;
        end
    end


    j = 0;
    for i = 1:c-1                            % Locate the position of peaks
        if n(i)~=0 && n(i+1)==0
            px(k-2,j+1) = i;
            j = j+1;
        end
    end

    for i = 1:rpx
```

```
        for j =1:cpx-1
            if px(i,j+1)~=0
                xSpacing(i,j) = px(i,j+1)-px(i,j);  % Calculate the values of spacings
            end
        end
    end


      histo = bar(n);
      str = ['0.' name];
      num1 = str2double(str);

    saveas(histo, [dir_name2,'\',name,'.jpg']);    % Save histograms as jpeg images.


end
xlswrite(ffn, xSpacing ,'sheet1', 'A1');          % Save data into an excel document.
```

Document 2

```
% Script: Rigtpart.m
% Purpose: process the right part of an experimental image to obtain the exact
% ratio, ratio2_exp, of the distances A and B.
% A is the distance between the point on the thimble which coincides with the axial
% line to the lower graduation.
% B is the distance between the lower graduation and the higher graduation.
% Input: '*****.jpg' - the experimental image.
% Output:  ratio2_exp --  the ratio as describe above.

 clc;clear all;
BW = imread('25973.jpg');  % A typical experimental image
    [r,c] = size(BW);
    BW =BW>100;
    [n,~] = hist(BW);
    n = n(10,:);

    % Filter out unnecessary parts
    for i= 1:c
        if n(i)<20
            n(i) = 0;
        end
    end

    % Locate the ranges of lower graduation, higher graduation and the axial line.
    py=zeros(2,3);
    j = 0;
    for i = 1:c-1
        if n(i)~=0 && n(i+1)==0
            py(2,j+1) = i;
            j = j+1;
        end
    end

    j = 0;
    for i = 1:c-1
        if n(i)==0 && n(i+1)~=0
            py(1,j+1) = i+1;
            j = j+1;
```

```
        end
    end

% Find the means for lower graduation, higher graduation and the axial line.
% Use the means to represent the positions of each line.
% Calculate the ratio using the means.
    pymean = mean(py);
    ratio2_exp = (pymean(2)-pymean(1))/(pymean(3)-pymean(1));
% Plot the intensity histogram
 histo = bar(n)
```

Appendix B Matlab scripts used in Chapter 5

This part contains matlab scripts used in Chapter 5.

Document 1

```
% Title: CENTROIDS.m
% CENTROIDS is the main script.
% It calls a number of functions to do incline-correction reconstruct the virtual
% raster lines and decoding.

% clean up the work space
clc; clear all;
disp('Running centroids.m...'); % Message sent to command window.


[FileName,PathName] = uigetfile({'*.bmp;*.tif;*.jpg;*.gif;*.png','Images
(*.bmp,*.tif,*.jpg,*.gif,*.png)';
   '*.bmp',  'BMP Image(*.bmp)'; ...
   '*.tif','Tiff Image(*.tif)'; ...
   '*.jpg','JPEG Image (*.jpg)'; ...
   '*.gif','GIF (*.gif)'; ...
   '*.png','PNG (*.png)'; ...
   '*.*',  'All Files (*.*)'}, ...
   'Select an Image');

if isequal(FileName,0)
   disp('User selected Cancel')
else
   FullFileName = fullfile(PathName, FileName);
end

OI = imread(FullFileName);
OI = flipud(OI);
figure
%imshow(OI);
[numofBlobs, centroids] = GetCentroids(OI);
%}



% Since centroids is a N×2 array, it's not easy and instinct to utilize
% its data, we set two new N×N arrays x and y to store x-coordinates and
% y-coordinates of the centroids separately.
N = sqrt(numofBlobs);

   xt = zeros(N);
   yt = zeros(N);
   for n=1:N
      for m=1:N
         xt(m,n) = centroids(m+(n-1)*N,1);
         yt(m,n) = centroids(m+(n-1)*N,2);
      end
   end

   [temp,index]=sort(yt,'descend');
   % create y matrix
```

```
  y = temp;

  % create x matrix
  x = zeros(N);
  for m=1:N
     for n=1:N
        x(n,m)=xt(index(n,m),m);
     end
  end


% obtain the offset indicators recX and recY
% obtain the spacing in x and y directions
 [recX,recY,xSpace,ySpace] = dotDirec(x,y);

 [averx,avery,~,~] = Gridestab(recX,recY,x,y);


 for i = 1:N
    for j = 1:N
       if abs(recX(j,i))+abs(recY(j,i))~=1
          if abs(x(j,i)-averx(i))>abs(y(j,i)-avery(j));
             recY(j,i) = 0;
             recX(j,i) = sign(x(j,i)-averx(i));
          else
             recY(j,i) = sign(y(j,i)-avery(j));
             recX(j,i) = 0;
          end
       end
    end
 end


 MarkVal_x = zeros(N);
 MarkVal_y = zeros(N);


 % calculate the mark value
 for i=1:N^2
    if recX(i) == -1 && recY(i) == 0
       MarkVal_x(i) = 3;
    elseif recX(i) == 1 && recY(i) == 0
       MarkVal_x(i) = 1;
    elseif recX(i) == 0 && recY(i) == -1
       MarkVal_y(i) = 4;
    elseif recX(i) == 0 && recY(i) == 1
       MarkVal_y(i) = 2;
    else

    end
 end

 MarkVal = MarkVal_x +MarkVal_y;

% using mark value to determine x, y codes.
 xcode = zeros(N); ycode = zeros(N);
 for j =1:N
    for i = 1:N
       switch MarkVal(j,i)
```

```
            case 1
                xcode(j,i)=0;
                ycode(j,i)=1;
            case 2
                xcode(j,i)=0;
                ycode(j,i)=0;
            case 3
                xcode(j,i)=1;
                ycode(j,i)=0;
            case 4
                xcode(j,i)=1;
                ycode(j,i)=1;
            otherwise
             % when error occurs
                xcode(j,i)=3;
                ycode(j,i)=3;
        end
    end

 end

% find the pixel coordinates of the centroid of the object
[objectX,objectY] = getcent
objectR = 0;
objectC = 0;

for i = 1:N-1
    if objectX >= averx(i) && objectX <=averx(i+1)
        objectC = i + (objectX-averx(i))/(averx(i+1)-averx(i));
    end
end

for j = 1:N-1
    if objectY <= avery(j) && objectY >=avery(j+1)
        objectR = j + (objectY-avery(j))/(avery(j+1)-avery(j));
    end
end

% establish the code book as a reference for decoding
BitSeq =  enCod(0,6);
PaSeq_1SDS = enCod(1,5);
PaSeq_2SDS = enCod(2,5);
PaSeq_3SDS = enCod(3,5);
PaSeq_4SDS = enCod(4,5);
% decoding..
% obtain the PDS
[PVy,PDSy] = deCody(ycode, BitSeq, 6);
[PVx,PDSx] = deCodx(xcode, BitSeq, 6);

% convert PDS into SDS in x direction
[a1x,a2x,a3x,a4x] = PDScoeff(PDSx);
% obtain the place numbers in SDS in x direction
       [p1x,~] = deCodx(a1x,PaSeq_1SDS ,5);
       [p2x,~] = deCodx(a2x,PaSeq_2SDS ,5);
       [p3x,~] = deCodx(a3x,PaSeq_3SDS ,5);
       [p4x,~] = deCodx(a4x,PaSeq_4SDS ,5);

% convert PDS into SDS in y direction
[a1y,a2y,a3y,a4y] = PDScoeff(PDSy);
```

```
% obtain the place numbers in SDS in y direction

      [p1y,~] = deCody(a1y,PaSeq_1SDS ,5);
      [p2y,~] = deCody(a2y,PaSeq_2SDS ,5);
      [p3y,~] = deCody(a3y,PaSeq_3SDS ,5);
      [p4y,~] = deCody(a4y,PaSeq_4SDS ,5);
 % obtain the place numbers in PDS if needed
      %Px = Place_PDS(xcode,1);
      %Py = Place_PDS(ycode,2);
 % obtain the place numbers of the ROI
     Bx = [p1x(1,1),p2x(1,1),p3x(1,1),p4x(1,1)];
     By = [p1y(1,1),p2y(1,1),p3y(1,1),p4y(1,1)];
 % the place numbers of the first column and the first row
 % provided as a reference when establish Table 5-10 and Table 5-11
     Ax = [32,57,12,96];
     Ay = [224,19,4,155];

  % obtain the coordinates in columns and rows of the ROI.
     X = DetCor(Ax,Bx);
     Y = DetCor(Ay,By);

  % obtain the coordinates in columns and rows of the object.
     objectC = X + objectC -1;
     objectR = Y + objectR -1;
     Coordinate = [objectC,objectR]
```

Document 2

```
% Title: GetCentroids.m
%  [numofBlobs, centroids] = GetCentroids(OI) uses binary image information
%  contained in OI to determine the number of blobs, numofBlobs in the original
%  image, locate the coordinates (pixels) of the centroids of each blob and save
%  them in a array called centroids.


function [numofBlobs, centroids] = GetCentroids(OI)
% Maximize the figure window.
set(gcf, 'Position', get(0, 'ScreenSize'));

% Thresholding the image
 thres = 100;  % Set a threshold value for binarize the image
% BW = OI < thres;   % Store the obtained binary image into an array 'BW'
 BW = OI > thres;

% Find the coordinates of centroids of each blob
% Calculate all properties of the binary image
blobMeasurements  = regionprops(BW, 'centroid');
% Put the coordinates into an array 'centorids'
centroids = cat(1, blobMeasurements.Centroid);

% Plot the centroids on the binary image
%imagesc(BW); colormap(gray(256)); title('Centroids of each blob'); axis square;
imshow(BW)
[~,numofBlobs] = bwlabel(BW,8);
%hold on
plot(centroids(:,1), centroids(:,2),'.');axis square
%hold off
```

Document 3

```
% Title: enCod.m
```

```
% BitSeq =  enCod(NumSeq,n) establishes a code book for decoding.
%
% NumSeq-- an integer number that has values 0~4. Indicate the
% kind of the number sequence used.
% 0- main number sequence.
% 1~4- secondary number sequence set A1, A2, A3, A4.
% n— has the value 5, 6 or 8. n=5 indicates encoding SDS, n=6 or 8
% indicate encoding PDS using 6×6 array or 8×8 array.
% BitSeq— the output.

function BitSeq =  enCod(NumSeq,n)

switch NumSeq
    case 0
        NS =
[0,0,0,0,0,0,1,0,0,1,1,1,1,1,0,1,0,0,1,0,0,0,0,1,1,1,0,1,1,1,0,0,1,0,1,0,1,0,0,0,1,
0,1,1,0,1,1,0,0,1,1,0,1,0,1,1,1,1,0,0,0,1,1];
    case 4
        NS =
[0,0,0,0,0,1,0,2,0,0,0,0,2,0,0,2,0,1,0,0,0,1,1,2,0,0,0,1,2,0,0,2,1,0,0,0,2,1,1,2,0,
1,0,1,0,0,1,2,1,0,0,1,0,0,2,2,0,0,0,2,2,1,0,2,0,1,1,0,0,1,1,1,0,1,0,1,1,0,1,2,0,1,1
,1,1,0,0,2,0,2,0,1,2,0,2,2,0,1,0,2,1,0,1,2,1,1,0,1,1,1,2,2,0,0,1,0,1,2,2,2,0,0,2,2,
2,0,1,2,1,2,0,2,0,0,1,2,2,0,1,1,2,1,0,2,1,1,0,2,0,2,1,2,0,0,1,1,0,2,1,2,1,0,1,0,2,2
,0,2,1,0,2,2,1,1,1,2,0,2,1,1,1,0,2,2,2,2,0,2,0,2,2,1,2,1,1,1,1,2,1,2,1,2,2,2,1,0,0,
2,1,2,2,1,0,1,1,2,2,1,1,2,1,2,2,2,2,1,2,0,1,2,2,1,2,2,0,2,2,2,1,1];
    case 2
        NS =
[0,0,0,0,0,1,0,0,0,0,2,0,1,0,0,1,0,1,0,1,1,0,0,0,1,1,1,1,0,0,1,1,0,1,0,0,2,0,0,0,1,
2,0,1,0,1,2,1,0,0,0,2,1,1,1,0,1,1,1,0,2,1,0,0,1,2,1,2,1,0,1,0,2,0,1,1,0,2,0,0,1,0,2
,1,2,0,0,0,2,2,0,0,1,1,2,0,2,0,0,2,0,2,0,1,2,0,0,2,2,1,1,0,0,2,1,0,1,1,2,1,0,2,0,2,
2,1,0,0,2,2,2,1,0,1,2,2,0,0,2,1,2,2,1,1,1,1,1,2,0,0,1,2,2,1,2,0,1,1,1,2,1,1,2,0,1,2
,1,1,1,2,2,0,2,2,0,1,1,2,2,2,2,1,2,1,2,2,0,1,2,2,2,0,2,0,2,1,1,2,2,1,0,2,2,0,2,1,0,
2,1,1,0,2,2,2,2,0,1,0,2,2,1,2,2,2,1,1,2,1,2,0,2,2,2];
    case 3
        NS = [0,0,0,0,0,1,0,0,1,1,0,0,0,1,1,1,1,0,0,1,0,1,0,1,1,0,1,1,1,0,1];
    case 1
        NS =
[0,0,0,0,0,1,0,0,0,0,2,0,1,0,0,1,0,1,0,0,2,0,0,0,1,1,0,0,0,1,2,0,0,1,0,2,0,0,2,0,2,
0,1,1,0,1,0,1,1,0,2,0,1,2,0,1,0,1,2,0,2,1,0,0,1,1,1,0,1,1,1,1,0,2,1,0,1,0,2,1,1,0,0
,1,2,1,0,1,1,2,0,0,0,2,1,0,2,0,2,1,1,1,0,0,2,1,2,0,1,1,1,2,0,2,0,0,1,1,2,1,0,0,0,2,
2,0,1,0,2,2,0,0,1,2,2,0,2,0,2,2,1,0,1,2,1,2,1,0,2,1,2,1,1,0,2,2,1,2,1,2,0,2,2,0,2,2
,2,0,1,1,2,2,1,1,0,1,2,2,2,2,1,2,0,0,2,2,1,1,2,1,2,2,1,0,2,2,2,2,2,0,2,1,2,2,2,1,1,
1,2,1,1,2,0,1,2,2,1,2,2,0,1,2,1,1,1,1,2,2,2,0,0,2,1,1,2,2];
end


SL = length(NS);
bitseq = zeros(SL,n);
BitSeq = zeros(SL,n);



for i = 1:SL

       for j = 1:n
          if i+j-1 <= SL
             bitseq(i,j) = NS(i+j-1);
          else
             bitseq(i,j) = NS(i+j-1-SL);
```

```
        end

    end

end
BitSeq = cellstr(int2str(bitseq));
```

Document 4

```
% Title: deCodx.m
% [PosVal,DS] = deCodx(code, BitSeq, n)decodes the code in the x direction
% n– has the value 5, 6 or 8. n=5 indicates encoding SDS, n=6 or 8
% indicate encoding PDS using 6×6 array or 8×8 array.

function [PosVal,DS] = deCodx(code, BitSeq, n)
[r_code,c_code] = size(code);
switch n
   case 8
      r_codeT = floor(r_code/n); % number of vertical bit sequences of x-code.
      c_codeT = c_code;
      bla = blanks(c_codeT*(n+2*(n-1)));
      bla = zeros(r_codeT,length(bla));
   case 6
      r_codeT = floor(r_code/n); % number of vertical bit sequences of x-code.
      c_codeT = c_code;
      bla = blanks(c_codeT*(n+2*(n-1)));
      bla = zeros(r_codeT,length(bla)); % build a temporary array to contain the
codes.
   case 5
      c_codeT = floor(c_code/n);
      r_codeT = r_code;
      bla = blanks(c_codeT*(n+2*(n-1)));
      bla = zeros(r_codeT,length(bla));

   otherwise
      disp('n must be 5 or 6 or 8')
end

PosVal = zeros(r_codeT,c_codeT);
DS = zeros(r_codeT,c_codeT-1);  %Primary Difference Sequence

switch n
   case 8
      for i = 1: r_codeT
         code8 = char(int2str(code(1+(i-1)*n:i*n,:)'));
         [r8,c8] = size(code8);
         for j = 1:r8
            bla(i, 1+(j-1)*c8:j*c8) = code8(j,:);
         end
      end

   case 6
      for i = 1: r_codeT
         code6 = char(int2str(code(1+(i-1)*n:i*n,:)'));
         [r6,c6] = size(code6);
         for j = 1:r6
            bla(i, 1+(j-1)*c6:j*c6) = code6(j,:);
         end
      end
```

```
    case 5
        for i = 1: r_codeT
            code5 = char(int2str(code(:,1+(i-1)*n:i*n)));
            [~,c5] = size(code5);
             bla(:,1+(i-1)*c5:i*c5) = code5;

        end

end
bla = char(bla);

for i = 1:r_codeT
    for j = 1:c_codeT
        for k = 1:length(BitSeq)
            if isequal(bla(i,1+(j-1)*n+2*(n-1)*(j-1):j*n+2*(n-1)*j),BitSeq{k})
                PosVal(i,j) = k-1;
            end
        end
    end
end

switch n
    case 6
        for i = 1:r_codeT
            for j = 1:c_codeT-1
                DS(i,j) = mod((PosVal(i,j+1)-PosVal(i,j)),63);

            end
        end

    case 8
        for i = 1:r_codeT
            for j = 1:c_codeT-1
                DS(i,j) = mod((PosVal(i,j+1)-PosVal(i,j)),63);

            end
        end

    otherwise
        DS = [];
end
```

Document 5

```
% Title: deCody.m
% [PosVal,DS] = deCody(code, BitSeq, n)decodes the code in the y direction
% n— has the value 5, 6 or 8. n=5 indicates encoding SDS, n=6 or 8
% indicate encoding PDS using 6×6 array or 8×8 array.

function [PosVal,DS] = deCody(code, BitSeq, n)
[r_code,c_code] = size(code);
switch n
    case 8
        c_codeT = floor(c_code/n); % number of vertical bit sequences of x-code.
        r_codeT = r_code;
        bla = blanks(c_codeT*(n+2*(n-1)));
```

```
        bla = zeros(r_codeT,length(bla)); % build a temporary array to contain the
codes.
    case 6
        c_codeT = floor(c_code/n); % number of vertical bit sequences of x-code.
        r_codeT = r_code;
        bla = blanks(c_codeT*(n+2*(n-1)));
        bla = zeros(r_codeT,length(bla)); % build a temporary array to contain the
codes.
    case 5
        r_codeT = floor(r_code/n);
        c_codeT = c_code;
        bla = blanks(c_codeT*(n+2*(n-1)));
        bla = zeros(r_codeT,length(bla));

    otherwise
        disp('n must be 5 or 6.')
end

PosVal = zeros(r_codeT,c_codeT);
DS = zeros(r_codeT-1,c_codeT);  %Primary Difference Sequence

switch n
    case 8
        for i = 1: c_codeT
            code8 = char(int2str(code(:,1+(i-1)*n:i*n)));
            [~,c8] = size(code8);

                bla(:,1+(i-1)*c8:i*c8) = code8;

        end

    case 6
        for i = 1: c_codeT
            code6 = char(int2str(code(:,1+(i-1)*n:i*n)));
            [~,c6] = size(code6);

                bla(:,1+(i-1)*c6:i*c6) = code6;

        end


    case 5
        for i = 1: r_codeT
            code5 = char(int2str(code(1+(i-1)*n:i*n,:)'));
            [r5,c5] = size(code5);
            for j = 1:r5
                bla(i, 1+(j-1)*c5:j*c5) = code5(j,:);
            end
        end

end
bla = char(bla);

for i = 1:r_codeT
    for j = 1:c_codeT
        for k = 1:length(BitSeq)
            if isequal(bla(i,1+(j-1)*n+2*(n-1)*(j-1):j*n+2*(n-1)*j),BitSeq{k})
                PosVal(i,j) = k-1;
            end
```

```
        end
    end
end

switch n
    case 8
        for i = 1:r_codeT-1
            for j = 1:c_codeT
                DS(i,j) = mod((PosVal(i+1,j)-PosVal(i,j)),63);
                %DS(i,j) = mod(abs((PosVal(i,j+1)-PosVal(i,j))),63);
            end
        end

    case 6
        for i = 1:r_codeT-1
            for j = 1:c_codeT
                DS(i,j) = mod((PosVal(i+1,j)-PosVal(i,j)),63);
                %DS(i,j) = mod(abs((PosVal(i,j+1)-PosVal(i,j))),63);
            end
        end
    otherwise
        DS = [];
end
```

Document 6

```
% Title: PDScoeff.m
% [a1,a2,a3,a4] = PDScoeff(PDS) converts the primary difference sequence PDS into
% sequences a1, a2, a3 and a4.
function [a1,a2,a3,a4] = PDScoeff(PDS)
[r,c] = size(PDS);
a1= zeros(r,c);a2= zeros(r,c);a3= zeros(r,c);a4= zeros(r,c);

for i = 1:r
    for j = 1:c
        if PDS(i,j)>=5 && PDS(i,j)<= 58%PDS(i,j)>=5 && PDS(i,j)<= 58
            if PDS(i,j)>=41  % 5+2*18
                a4(i,j) = 2;
            elseif PDS(i,j)>=23 % 5+1*18
                a4(i,j) = 1;
            else
                a4(i,j) = 0;
            end

            PDS(i,j) = PDS(i,j) - a4(i,j)*18 -5;
            if PDS(i,j)>= 9 % 1*9
                a3(i,j) = 1;
            else
                a3(i,j) = 0;
            end

            PDS(i,j) = PDS(i,j)-a3(i,j)*9;
            if PDS(i,j) >= 6 % 2*3
                a2(i,j) = 2;
            elseif PDS(i,j) >= 3 %1*3
                a2(i,j) = 1;
            else
                a2(i,j) = 0;
            end
```

```
            PDS(i,j)= PDS(i,j)-a2(i,j)*3;
            a1(i,j) = PDS(i,j);
        else      % which means the element of PDS does not belong to [5,58]
            a1(i,j) = 8;
            a2(i,j) = 8;
            a3(i,j) = 8;
            a4(i,j) = 8;
        end

    end
end
```

Document 7

```
% Title: DetCor.m
% X  = DetCor(A,B) determines the coordinates of the dots in columns and rows.
% A = (p1r,p2r,p3r,p4r) is the reference unique 4-number combination.
% B = (p1,p2,p3,p4) is the 4-number combination of the interested region.
function X  = DetCor(A,B)

a = zeros(4,1000);
% establish a table
for i = 1:1000
    a(1,i) = mod(A(1)+i-1,236);
    a(2,i) = mod(A(2)+i-1,233);
    a(3,i) = mod(A(3)+i-1,31);
    a(4,i) = mod(A(4)+i-1,241);

end


% output the result.

flag = 0;
for i=1:1000
    if a(4,i) == B(4) && a(3,i)== B(3) && a(2,i) == B(2)  &&  a(1,i) == B(1)
         X = i;
         flag =1;
    end
end
if flag == 0
     disp('no match found')
     X = 0;
end
```

Document 8

```
% Title: getcent.m
% [X,Y] = getcent find the centroid coordinates in pixels for the object.
function [X,Y] = getcent
clc; clear all;
disp('Running program, please wait...'); % Message sent to command window.

[FileName,PathName] = uigetfile({'*.bmp;*.tif;*.jpg;*.gif;*.png','Images
(*.bmp,*.tif,*.jpg,*.gif,*.png)';
   '*.bmp',  'BMP Image(*.bmp)'; ...
   '*.tif','Tiff Image(*.tif)'; ...
   '*.jpg','JPEG Image (*.jpg)'; ...
   '*.gif','GIF (*.gif)'; ...
```

```
   '*.png','PNG (*.png)'; ...
   '*.*',  'All Files (*.*)'}, ...
   'Select an Image');

if isequal(FileName,0)
   disp('User selected Cancel')
else
   FullFileName = fullfile(PathName, FileName);
end

OI3 = imread(FullFileName);
OI3 = flipud(OI3);
figure
[numofBlobs, centroids] = GetCentroids(OI3);

N = sqrt(numofBlobs);
   xt = zeros(N);
   yt = zeros(N);
   for n=1:N
      for m=1:N
         xt(m,n) = centroids(m+(n-1)*N,1);
         yt(m,n) = centroids(m+(n-1)*N,2);
      end
   end

   [temp,index]=sort(yt,'descend');
   % create y matrix
   y = temp;

   % create x matrix
   x = zeros(N);
   for m=1:N
      for n=1:N
         x(n,m)=xt(index(n,m),m);
      end
   end

   X = x;
   Y = y;
```

Document 9

```
% Title: dotDirec.m
%  [recX,recY,xSpace,ySpace] = dotDirec(x,y) analyzes the x-coordinates and
%  y-coordinates of the centroids and assign offset direction indicators recX and
%  recY to each dot. xSpace and ySpace are two arrays used to record the spacings
%  of the columns and rows respectively.

function [recX,recY,xSpace,ySpace] = dotDirec(x,y)
[N,M] = size(x);
recX = zeros([N,M]);
recY = recX;
xSpace = zeros(1,M);
ySpace = zeros(N,1);
xt1 = zeros(1,M);
xt2 = zeros(1,M);
yt1 = zeros(N,1);
yt2 = zeros(N,1);
```

```
for i = 1:M
        xt1(i) = min(x(:,i));
        xt2(i) = max(x(:,i));
        xSpace(i) = xt2(i)-xt1(i);
end


for i = 1:M

        for j = 1:N
            x_m1 = abs((x(j,i)/xt1(i))-1);
            x_m2 = abs((x(j,i)/xt2(i))-1);
            x_mm = min([x_m1,x_m2]);
            if xSpace(i)> 2/3* max(xSpace)

                    xt3 = (xt2(i)-xt1(i))/3+xt1(i);
                    xt4 = 2*(xt2(i)-xt1(i))/3+xt1(i);
                    if x(j,i)>=xt1(i) && x(j,i)< xt3
                        recX(j,i) = -1;
                    elseif x(j,i)>=xt3 && x(j,i)< xt4
                        recX(j,i) = 0;
                    elseif x(j,i)>=xt4 && x(j,i)<= xt2(i)
                        recX(j,i) = 1;
                    else
                        switch x_mm
                            case x_m1
                                recX(j,i)=-1;
                            case x_m2
                                recX(j,i)=1;
                        end
                    end
            elseif  xSpace(i)< 1/3* max(xSpace)
                recX(j,i) = 3;
            else
                    xt5 = 0.5*(xt1(i)+xt2(i));
                    if x(j,i)>=xt1(i) && x(j,i) < xt5
                        recX(j,i) = -2;
                    elseif x(j,i)>=xt5 && x(j,i) < xt2(i)
                        recX(j,i) = 2;
                    else
                        switch x_mm
                            case x_m1
                                recX(j,i)=-2;
                            case x_m2
                                recX(j,i)=2;
                        end
                    end


            end
        end
end

for j = 1:N
        yt1(j) = min(y(j,:));
        yt2(j) = max(y(j,:));
        ySpace(j) = yt2(j)-yt1(j);
end
```

```
for j = 1:N
    for i = 1:M
        y_m1 = abs((y(j,i)/yt1(j))-1);
        y_m2 = abs((y(j,i)/yt2(j))-1);
        y_mm = min([y_m1,y_m2]);

             if ySpace(j)> 2/3* max(ySpace)

                   yt3 = (yt2(j)-yt1(j))/3+yt1(j);
                   yt4 = 2*(yt2(j)-yt1(j))/3+yt1(j);
                   if y(j,i)>=yt1(j) && y(j,i)< yt3
                       recY(j,i) = -1;
                   elseif y(j,i)>=yt3 && y(j,i)< yt4
                       recY(j,i) = 0;
                   elseif y(j,i)>=yt4 && y(j,i)<= yt2(j)
                       recY(j,i) = 1;
                   else
                       switch y_mm
                          case y_m1
                              recY(j,i)=-1;
                          case y_m2
                              recY(j,i)=1;
                       end
                   end
             elseif  ySpace(i)< 1/3* max(ySpace)
                recY(j,i) = 3;

             else
                    yt5 = 0.5*(yt1(j)+yt2(j));
                    if y(j,i)>=yt1(j) && y(j,i) < yt5
                        recY(j,i) = -2;
                    elseif y(j,i)>=yt5 && x(j,i) < yt2(j)
                        recY(j,i) = 2;
                    else

                       switch y_mm
                          case y_m1
                              recY(j,i)=-2;
                          case y_m2
                              recY(j,i)=2;
                       end
                    end
             end
    end
end


xismatched = zeros(N,M);
wrongcolumn = zeros(1,M);
x_replace = 225;
k = 1;

for i = 1:M
    if isempty(find(recX(:,i) ==0))
        wrongcolumn(k)= i;
        k = k+1;
    end
end
```

```
for i = 1:M
    if wrongcolumn(i) ~=0
        t = wrongcolumn(i);

        for j = 1:N
            if (recY(j,t)== -1) || (recY(j,t) == 1)
                x_replace = recX(j,t);
                xismatched(j,t) = 1;
            else
                xismatched(j,t) = 0;
            end


            if recX(j,t) == x_replace
                recX(j,t) = 0;
            elseif recX(j,t)>0
                recX(j,t) = 1;
            else
                recX(j,t) = -1;
            end
        end




    end
end


yismatched = zeros(N,M);
wrongrow = zeros(N,1);
y_replace = 225;
l = 1;

for j = 1:N
    if isempty(find(recY(j,:) ==0))
        wrongrow(l)= j;
        l = l+1;
    end
end

for j = 1:N
    if wrongrow(j) ~=0
        s = wrongrow(j);

        for i = 1:M
            if (recX(s,i)== -1) || (recX(s,i) == 1)
                y_replace = recY(s,i);
                yismatched(s,i) = 1;
            else
                yismatched(s,i) = 0;
            end


            if recY(s,i) == y_replace
                recY(s,i) = 0;
            elseif recY(s,i)>0
                recY(s,i) = 1;
```

```
            else
                recY(s,i) = -1;
            end
        end



    end
end
```

Document 10

```
% Title: Gridestab.m
% [averx,avery,~,~] = Gridestab(recX,recY,x,y)reconstructs the virtual raster lines
%  and returns the coordinates averx and avery of the raster lines.
function [averx,avery,deltax,deltay] = Gridestab(recX,recY,x,y)

plot(x(:,:),y(:,:),'r*');axis square
hh=axis;
[M,N] = size(x);
averx = zeros(1,M);
deltax = zeros(1,M-1);
hold on;

for i = 1:M
      count = 0;
sum = 0;

    for j = 1:N

        if recX(j,i) == 0
            sum = sum + x(j,i);
            count = count+1;

        end

    end
    averx(i) = sum/count;
    plot([averx(i),averx(i)],[hh(3),hh(4)],'b');axis square
end

for i = 1:M-1
    deltax(i) = averx(i+1)-averx(i);
end


avery = zeros(1,N);
deltay = zeros(1,N-1);

for j = 1:N
      count = 0;
sum = 0;

    for i = 1:M

        if recY(j,i) == 0
            sum = sum + y(j,i);
            count = count+1;
```

```
        end

    end
    avery(j) = sum/count;
    plot([hh(1),hh(2)],[avery(j),avery(j)],'b');axis square
end

for j = 1:N-1
    deltay(j) = avery(j+1)-avery(j);
end

hold off
```

Document 11

```
% Title: RotateCorrect.m
% Purpose: Do incline-correction.
clc;
clear all;
[FileName,PathName] = uigetfile({'*.bmp;*.tif;*.jpg;*.gif;*.png','Images
(*.bmp,*.tif,*.jpg,*.gif,*.png)';
   '*.bmp',  'BMP Image(*.bmp)'; ...
   '*.tif','Tiff Image(*.tif)'; ...
   '*.jpg','JPEG Image (*.jpg)'; ...
   '*.gif','GIF (*.gif)'; ...
   '*.png','PNG (*.png)'; ...
   '*.*',  'All Files (*.*)'}, ...
   'Select an Image');

if isequal(FileName,0)
   disp('User selected Cancel')
else
   FullFileName = fullfile(PathName, FileName);
end

OI = imread(FullFileName);

imshow(OI);

[numofBlobs, centroids] = GetCentroids(OI);

xy = [centroids(round(numofBlobs/2),1), centroids(round(numofBlobs/2),2)];
P = FindPoint(centroids,xy);


alphaSum = 0;
for i=1:4
   temp = P{i}-xy;
   d = sqrt(temp(1)^2+temp(2)^2);
   if temp(1)/d ~=0 && temp(2)/d ~=0
   if mod(i,2)==0
      asin(abs(temp(2))/d)
      alpha = asin(abs(temp(2))/d);
   else
      asin(abs(temp(2))/d)
      alpha = pi/2-asin(abs(temp(2))/d);
   end
   alphaSum = alphaSum + alpha;
   else
```

```
      alphaSum = 0;
   end
end
alphaM = 0.25*alphaSum*180/pi;
sprintf('the rotated angle is: %.2f degrees',alphaM)

[FileName,PathName] = uigetfile({'*.bmp;*.tif;*.jpg;*.gif;*.png','Images
(*.bmp,*.tif,*.jpg,*.gif,*.png)';
   '*.bmp',  'BMP Image(*.bmp)'; ...
   '*.tif','Tiff Image(*.tif)'; ...
   '*.jpg','JPEG Image (*.jpg)'; ...
   '*.gif','GIF (*.gif)'; ...
   '*.png','PNG (*.png)'; ...
   '*.*',  'All Files (*.*)'}, ...
   'Select an Image');

if isequal(FileName,0)
   disp('User selected Cancel')
else
   FullFileName = fullfile(PathName, FileName);
end
OI2 = imread(FullFileName);
RI = imrotate(OI2,-alphaM);
figure
imshow(RI);
```

Document 12[28]

```
% Title: FindPoint.m
% This function is not written by the author of this thesis. It is found on a
% Matlab learner's forum on the internet.
% Point = FindPoint(xy0, xy)Find nearest points of one given point in 4 quadrants.
function  Point = FindPoint(xy0, xy)
Point = cell(1,4);
x = xy0(:,1) - xy(1);
y = xy0(:,2) - xy(2);
% 1st quadrant
id1 = (x>0 & y>=0);
Point{1} = subfun(id1,x,y,xy0);
% 2nd quadrant
id2 = (x<=0 & y>0);
Point{2} = subfun(id2,x,y,xy0);
% 3rd quadrant
id3 = (x<0 & y<=0);
Point{3} = subfun(id3,x,y,xy0);
% 4th quadrant
id4 = (x>=0 & y<0);
Point{4} = subfun(id4,x,y,xy0);

function xyPoint = subfun(id,x,y,xy0)  % find the nearest point
xyPoint = [];
x1 = x(id);
if ~isempty(x1)
    y1 = y(id);
    distance = x1.^2 + y1.^2;
    xy0_1 = xy0(id,:);
    xyPoint = xy0_1(find(distance == min(distance)),:);
end
```