Key Duplicating Software Using Digital Image Processing

by

Naeem Nematollahi

A Thesis

Presented to Lakehead University

in Partial Fulfillment of the Requirements for the Degree of

Master of Science

in

Electrical and Computer Engineering

Thunder Bay, Ontario, Canada

20 September 2011

Abstract

In order to cut a key duplicate, it is necessary to measure the original key very precisely. There is a great variety in the key measuring devices currently in use; however, they follow the common basic principle of using a known light source of some kind and measuring the features of the light reflected off the key. In some variations of these systems, only a small part of the key can be measured at one time, and therefore some part of the system needs to be physically mobile to measure the entire key; be it the key itself, the light source or the camera measuring the reflected light. Most of the systems also require the key to be positioned exactly straight in the device in order to be measured, and even a slight deviation means that the blank key will be cut wrong and will be unusable.

In this thesis, I propose a new key measuring system based on digital image processing methods. This system will be able to measure a key based on an image scanned using a regular desktop scanner, rather than an expensive and dedicated key measuring machine. A user will be able to place the key on the scanner in any angle; the system will detect the key's orientation from the image and account for it in the measures. Finding the correct key blank is one of the two important tasks in this project. I need to compare parts of the keys that are the same on all keys of the same model. Key heads of the same model, unlike the shanks, remain unchangeable and are good for comparison. By comparing the results in the database, I will find the closest model to the modeled original head image and retrieve the corresponding key blank and its necessary information like size and number.

After the system finds the correct key blank, it will measure the cuts and depth of dents on the original shank image. Using Canny Edge Detection technique, I will get a contour that only contains a connected shape of the edges of the shank with only one pixel width. Measuring the cut and dents in pixels is too jagged so using sub-pixel interpolation I will interpolate more precise coordinates for the points observed in the image as edge pixels. At the end, I will save the positions of each sub-pixel point in an array.

Acknowledgement

I would like to thank my supervisor Dr. Richard Khoury for his guidance and patience throughout while completing this thesis. Without his common-sense, knowledge and perceptiveness I would never have finished my thesis.

I am grateful to all my professors at Lakehead University specially Dr. Carlos Christoffersen, Dr. Hassan Naser, Dr. Maurice Benson and Dr. Rachid Benlamri for giving me helpful advice to my study and research.

I would also like to extend my thanks to my family. My parents have provided me with countless opportunities for which I am eternally grateful. My wife is my source of strength and without her support, encouragements and love, this thesis would never have started much less finished.

Table of Contents

Abstract ii
Acknowledgement iii
Table of Contents iv
List of Figures vi
List of Tables ix
1 Introduction 1
1.1 Background and Motivation 1
1.2 Definitions 1
1.3 Thesis Contribution 2
1.4 Thesis Outline 3
2 Past Works 4
2.1 Introduction 4
2.2 Backlight-based systems 6
2.3 Laser-based systems 10
3 Image Processing Algorithm 15
3.1 Introduction 15
3.2 filledBinary 16
3.2.1 Converting the color or grayscale image into a binary image 16
3.2.2 Hole and Noise Elimination 18
3.3 cutHead 19
3.3.1 Properties of the image 20
3.3.2 Adjusting the angle of the key image 22
3.3.3 Adding zero pad array to the image 22
3.3.4 Cutting head from the key image 24
3.4 findKey 29
3.4.1 Eliminating useless space of the image 29
3.4.2 Moment Invariants for modelling the head of the key image 29
3.4.3 Distance of moment invariants between two images 32
3.4.4 Database and finding correct key blank 32
3.5 Shank 40
3.5.1 Cutting the shank from the key image and detecting the edges and dents of the shank 40

3.5.2 Detecting edges and dents of the shank 42
3.5.3 Sub-pixel Interpolation 45
3.5.4 3.5.4 Determining the positions of cuts and converting them to physical measures 48
3.6 Another example of the software procedures on a sample key image 49
4 Experimental Results 61
4.1 Introduction 61
4.2 Identify the correct key blank 61
4.2.1 Sample Keys 63
4.2.2 Results on 21 Keys 67
4.2.3 Results with Database 69
4.3 Finding precise measures of the edges 74
4.3.1 Image to Key 74
4.3.2 Image to Image 78
4.3.3 Image to Cut Key to Original Key 79
5 Conclusions and Future Research 81
5.1 Contributions 81
5.2 Summary of the Results 81
5.3 Future work 82
Bibliography 83

List of Figures

Figure 1.1 The parts of a key 2
Figure 2.1 The backlight system using horizontal light generator [1] 6
Figure 2.2 Transparent section of the backlight system [1] 6
Figure 2.3 Intersection of the light beam line with the key surface [30] 7
Figure 2.4 Intersection of the light beam line with the key surface [30] 8
Figure 2.5 Source of the uniform light [3] 9
Figure 2.6 The lens of the second receiver is flush with the second surface [3] 9
Figure 2.7 Capturing two parts of the key using two receivers [3] 10
Figure 2.8 The measuring device (the laser and photo detector array) [29] 11
Figure 2.9 Determination of the position of the points on the key [29] 11
Figure 2.10 Measuring side of the key [29] 12
Figure 2.11 video measurement systems [29] 12
Figure 2.12 Image of the key when light rays are reflected back directly into a camera or scanner [32] . 13
Figure 2.13 A beam emitter emits a beam of light into a series of mirrors and beam splitter [24] 14
Figure 2.14 Two laser light sources symmetrical about the plane [25] 14
Figure 3.1 scanning the key using an open lid scanner 15
Figure 3.2 Pseudo-code of the algorithms 16
Figure 3.3 A sample color key image 17
Figure 3.4 The sample key converted to binary image 17
Figure 3.5 The inverted binary key image 17
Figure 3.6 Binary image filled with white pixels 18
Figure 3.7 Incomplete gaps after filling the image 18
Figure 3.8 The 4-connected filter for covering holes 18
Figure 3.9 Binary key image without any gaps and noise 19
Figure 3.10 Ellipse superimposed over a sample image (taken from the Matlab documentation) 21
Figure 3.11 Superimposed ellipse and the centroid over the image 22
Figure 3.12 Covered and uncovered parts of the key when the key is at the corner 23
Figure 3.13 Parts of the key image that cannot be detected and is not accessible from y-axis 23
Figure 3.14 Adding zero pad arrays to the image 24
Figure 3.15 Major axis line and superimposed ellipse over the key 25
Figure 3.16 Top, bottom and centre points of the key image 25
Figure 3.17 Start and end points for finding joint of the key 26
Figure 3.18 Image cut from 1/12th of the distance before and after the centre point of the key 26
Figure 3.19 Cut-line that cuts the head from the key 28
Figure 3.20 Head of the key cut from the key image 28
Figure 3.21 Head of the key after applying boundingbox function 29
Figure 3.22 (a) Original image. (b) Translated image. (c) Half-size image. (d) Mirrored image. (e) Image rotated 45°. (f) Image rotated 90° 31
Figure 3.23 (a) Key number 67. (b) Key number 77. They both have two joints inside their divisions. (c) and (d) are close-ups of (a) and (b), respectively 33

Figure 3.24 (a) Key head No.67 cut from the top blue line. (b) Key head No.77 cut from the bottom blue line. (c)Key head No.77 cut from the bottom blue line. (d) Key head No.77 cut from the top blue line. .. 34
Figure 3.25 Sets of images in database 36
Figure 3.26 A gap not covered by the filter 37
Figure 3.27 Filled and unfilled head images of two key types 38
Figure 3.28 $\mathbf{a1}$ is the original head image that is compared with $\mathbf{a2}$, $\mathbf{a3}$ and $\mathbf{a4}$. $\mathbf{b1}$ is filled original image that is compared with $\mathbf{b2}$, $\mathbf{b3}$ and $\mathbf{b4}$. 39
Figure 3.29 Obtained key blank with the information 40
Figure 3.30 (a) Original color image. (b) Binary image after using filledBinary function 41
Figure 3.31 The shank cut from the key image 42
Figure 3.32 Edge detector masks, (a) Image neighbourhood. (b) Sobel. (c) Prewitt. (d) Roberts. 43
Figure 3.33 Connected contour of the shank using canny edge detection 44
Figure 3.34 Finding the contour of the shank without the line that separates head and shank 45
Figure 3.35 (a) Edge pixel and its neighbourhood in binary contour of the shank. (b) Retrieved neighbourhood from the grayscale key image. 46
Figure 3.36 Vandermonde matrix for solving the polynomial equation 47
Figure 3.37 The interpolating polynomial of nine points 47
Figure 3.38 sub-pixel points representing more precise edge points than edge pixels 48
Figure 3.39 Original grayscale key (2nd key) 49
Figure 3.40 Inverted binary key image (2nd key) 50
Figure 3.41 Binary key image without gaps, holes and noise (2nd key) 50
Figure 3.42 Division of key image (2nd key) 51
Figure 3.43 The line that separates the head and shank of the key image (2nd key) 52
Figure 3.44 The head cut from the body of key image (2nd key) 52
Figure 3.45 Head of the key in bounding box (2nd key) 53
Figure 3.46 Filled Original key head (2nd key) 53
Figure 3.47 Three head images of key type number 77 54
Figure 3.48 Three head images of key type number 77 (second joint) 54
Figure 3.49 Three head images of key type number 8 54
Figure 3.50 Three head images of key type number 96 55
Figure 3.51 Three head images of key type number 69 55
Figure 3.52 Three head images of key type number 66 56
Figure 3.53 Three head images of key type number 67 56
Figure 3.54 Three head images of key type number 67 (second joint) 57
Figure 3.55 Three head images of key type number 68 57
Figure 3.56 Obtained key blank from database using the function findKey(2nd key) 58
Figure 3.57 Shank cut from the original key image (2nd key) 58
Figure 3.58 Connected contour of the shank (2nd key) 59
Figure 3.59 Sub-pixel points over the shank (2nd key) 59
Figure 3.60 Edge points superimposed over the edge pixels 60
Figure 4.1 Three sets of seven types of key 62
Figure 4.2 21 Binary key heads 62

Figure 4.3 Sub-pixel edge points of the key shanks 75
Figure 4.4 Selected points on the key shank (Figure 4.5(a)) 76
Figure 4.5 Selected green points on the key shank (Figure 4.5(b)) 77
Figure 4.6 Two images of a key. 78
Figure 4.7 Selected points on two scans of the key. 78

List of Tables

Table 2.1 Similarities and differences between selections of reviewed patents 5
Table 3.1 Number of pixels in each row of the division 27
Table 3.2 Seven moment invariants of the images in Figure 3.21 32
Table 3.3 Comparing moment invariants of two-joint key heads 35
Table 3.4 Distances of filled and unfilled head images of two key types 38
Table 3.5 Number of pixels in each row (2nd key) 51
Table 3.6 Distances of key type number 77 54
Table 3.7 Distances of key type number 77 (second joint) 54
Table 3.8 Distances of key type number 8 55
Table 3.9 Distances of key type number 96 55
Table 3.10 Distances of key type number 69 55
Table 3.11 Distances of key type number 66 56
Table 3.12 Distances of key type number 67 56
Table 3.13 Distances of key type number 67 (second joint) 57
Table 3.14 Distances of key type number 68 57
Table 4.1 Moment invariants of binary head images 63
Table 4.2 Distances between three test keys and the other head images, using unfilled heads 64
Table 4.3 Distances between three test keys and the other head images, using filled heads 65
Table 4.4 Average of filled and unfilled distances between three test keys and the other head images 66
Table 4.5 Average of distances using the database of key images 67
Table 4.6 Errors occurred in each step with all 21 sample keys 67
Table 4.7 Ratio of number of pixels in head hole to number of head's white pixels 69
Table 4.8 Distances of the moments between 21 sample keys and the database (unfilled key heads) 70
Table 4.9 Distances between 21 sample keys and the database (filled key heads) 71
Table 4.10 Average of distances of filled and unfilled key heads (first set) 71
Table 4.11 Average of distances of filled and unfilled key heads (second set) 72
Table 4.12 Average of distances between first and second sets 72
Table 4.13 Average of distances between three sets of the database 74
Table 4.14 Distances (in mm) of the green points in Figure 4.6 measured by software and calliper 76
Table 4.15 Distances in mm of the green points in Figure 4.7 measured by software and calliper 77
Table 4.16 Measures of the points on the two scans of the key by the software and calliper in mm 79
Table 4.17 measures of the points on the first original image in mm 79
Table 4.18 measures of the points on the second original image in mm 79
Table 4.19 measures of the points on the third original image in mm 80

1 Introduction

1.1 Background and Motivation

Key duplication is a profitable business, and the service is offered today in most hardware stores. Consequently, there exists a large variety of key duplication systems. Older generations of these systems had to physically touch the original key and transmit the measures mechanically to a blade that cuts the new blank key. The blade moves along the length of the blank key and cuts the notches to the depth of the corresponding location on the original key. In these devices there was no measuring system to measure the depth and form of the notches, aside from the current location being cut. In addition, an operator must pick the correct key blank among different types of keys. By contrast, in new patents of key duplication systems, a measuring system is one of the main parts of the apparatus.

Key measuring systems obtain the information needed about the shape and size of the original key. The cuts and indentations can be measured, transmitted, processed and saved in a computer and can be sent to a cutting machine to duplicate the original key. As a result, the original key is only needed for the first duplication. Furthermore, some key duplication systems are able to automatically determine the original key type and to find the correct key blank.

The newer generation of key measuring systems centres on the innovative use of diverse light sources. Some key measuring systems use backlights to generate an image of the object's shadows and obtain the outline of the key. Other systems employ laser technology and collimated lights to measure the depth of the key shank. Others still use uniform light sources to evenly illuminate the surface of the object. Some new developments make use of 3D cameras and high resolution cameras to capture the cuts and edges of the key as precisely as possible.

Another important issue in key measuring systems is the use of moving parts. Some measuring systems can only observe a single part of the key at one time. In order to observe and measure all parts of a key, these systems need to have some component that moves physically. In some cases the system moves the key in the device, in others the key is fixed while the light sources or the cameras move.

Adjusting the orientation of the key is another common problem in several kinds of key duplication systems. Without having information about the position of the key in the system, it is impossible to measure the details of the key precisely enough to duplicate it. In older designs the original key has to be attached in an appropriate location and angle, and even a slight error could lead to generating the wrong cuts and to a useless new key. Newer designs allow the system to automatically detect the position of the key, and to move it using a combination of fixation device, rotation platter and stepping motor to bring it to the correct orientation.

1.2 Definitions

It is worth formally defining the different parts of a key. As illustrated in Figure 1.1, there are three main parts of any key: the head, shoulder, and shank. The head is a more decorative part, in the sense that it is not needed to open a lock. It will often have key manufacturer information or directives such as "do not duplicate" engraved on it, as well as a hole to put the key on a keychain. The shank is the part of the key that is inserted in a lock, and which must match the lock in order to open it. There are two defining features to the shank. The grooves are the length-wise regular indentations in the middle of the shank.

These are identical for all keys of a same type, and are used to ensure that only keys of that model can be inserted into a lock. The notches, sometimes also called the biting, are the unique teeth pattern on either one or both edges of the shank. They must exactly match the pattern of the pin tumblers inside the lock in order to turn and unlock it. Finally, the shoulder of the key is a short connecter between the head and the shank. The joint of the key is the exact border between the shoulder and the shank. In this project I handle the head and the shoulder up to the joint as one piece.

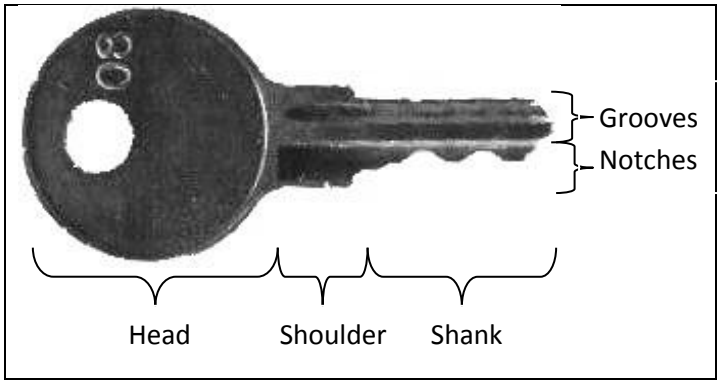


Figure 1.1 The parts of a key

A lock has a unique pattern of cuts and edges built into its cylinder. A key can slide into the cylinder if it has the right grooves in its shank, but it can only turn the cylinder to lock and unlock it if the notches match the pattern built into the tumbler of the cylinder. This is to say that any key of a given type can enter any lock of the same type, but not turn it. Consequently, key blanks of a given type come with grooves already carved in, and only the notches need to be cut.

1.3 Thesis Contribution

In this thesis, I introduce a new key measuring software that uses image processing techniques to achieve two main objectives: to identify the correct key blank and to find precise measures of the notches of the key shank. Given these two pieces of information, a cutting machine should be able to replicate the key. All the algorithms proposed in this thesis have been implemented and tested in Matlab.

One advantage of the system we are proposing is that it eliminates the need for dedicated key measuring devices and the use of sophisticated lighting systems and cameras. Instead, a common and commercially-available flatbed scanner is used to capture the image of the key. This image is then transmitted to a computer for processing. This furthermore eliminates the need for the key to be at the same place as the rest of the system: in our case, the key could be scanned at one location, the image emailed to a second location for processing, and the measures detected emailed to a third location for cutting.

A crucial challenge in our system will be to automatically discover the orientation of the key in the image. Indeed, our system will allow the original key to be located anywhere on the surface of the scanner with any angle and orientation. The software will have to detect and account for the orientation of the key during processing. This is one more advantage of the system: the operator does not need to take care to position the key properly, nor does the system need a setup to physically move the key to the correct place. Given this fact, and since the system can use any digital scanner of any brand and model to capture the image, it could be used by anybody.

1.4 Thesis Outline

The rest of this thesis is divided into four chapters. In Chapter 2 (Past Works), I describe the current state of patented technology for key measurement systems. For ease of presentation, I decided to categorize the patents into two families based on the light source they use to capture the image of the keys. Consequently, backlight-based systems are studied in section 2.2 and laser-based systems are presented in section 2.3.

My original image processing algorithm is detailed in Chapter 3. It is in that chapter that I give my main contributions, and that I explain the algorithms and methods I developed. After a brief overview in section 3.1, I present in section 3.2 the first processing steps needed to standardize the image and to eliminate the holes and noise it may contain. Next, I explain how the algorithm separates the head and the shank of the key precisely on the joint in section 3.3. In section 3.4 I show how the key head image is used to find the correct key blank in a database I designed. The procedure I use to find and measure the edges and notches of the key shank with subpixel accuracy are explained in section 3.5. First the shank is separated from the key image. Finally by combining the database information with the interpolated notch measures, I can convert the computed distances from pixels to millimetres.

The software has been thoroughly studied and tested, and the experimental results are analysed in Chapter 4. As I mentioned previously, the algorithm has two main objectives: to identify the correct blank, and to find precise measures of the notches of the key. The experimental results show conclusively that these goals have been achieved. In the first part, some tests are designed to show how the software processes the original image in order to find the correct match.distance. I tested 21 sample images of 7 different key types to expose problems I encountered and the procedures I used to solve them. In the second part of the experimental results, I show how precisely the software can find the edges of the key. I do so by comparing the depths of some selected points on a sample key with the same points on the image as measured by the software and on a professionally-made duplicate of the key.

Finally, I give some concluding remarks in Chapter 5. In this chapter I will recap the final results and show that we met the requirements presented in the contribution and goals of the project. Furthermore, I will propose directions for future work that can be considered to improve the quality of the results, and share some ideas for additions to this work that could expand the software.

2 Past Works

2.1 Introduction

There are three main parts of any key: the head, shoulder and shank. The head is a more decorative part, in the sense that it is not needed to open a lock. It will often have key manufacturer information or directives such as “do not duplicate” engraved on it. The shank is a part of the key that is inserted in a lock. There are two defining features to the shank. The grooves are the length-wise regular indentations in the middle of the shank. The notches, sometimes also called the biting, are the unique teeth pattern on either one or both edges of the shank.

While there is a great variety in key measuring devices, modern techniques follow a common basic principle, which consists in aiming a known light source of some kind at the key and measuring the light’s features after it hits the key using a camera. Some parts of the system need s to be mobile, the key, the light source or the camera, in order to measure the entire key. Moreover, the key must often be properly oriented in the system in order to get correct measurement, which means that a part of the system is dedicated to physically moving the key to the correct position.

The two main types of light sources that are used in key measuring systems are laser and backlight. Laser-based systems basically work by projecting a laser line on each side of the key and measuring the deflection angle to know the shape of the key at each point. Meanwhile, backlight systems get the outline of the key and measure the notches. We categorized the patents based on which of these two main light sources they use, in order to be able to compare them with each other.

Table 2.1 introduces an overview of the patented systems that I will be presenting.

Table 2.1 Similarities and differences between selections of reviewed patents

U.S. Patent Number	Number and type of light sources	Number of cameras	Moving part	Key orientation	Featured measured
5807042	1 backlight 1 collimated	2	Key and light stripe generator	Rotation platter	The shadows of key edges
6064747	1 LED backlight 1 Laser	1 CDD camera	Laser generator	Corrected by software	backlit and intersection of laser with side of the key
6175638	2 backlights	1	Key	Fixation device	Shadow image of cross sectional of the shank
6836553	2 uniform lights	1 or 2	Key	Corrected by software	Profile from backlit image and grooves from plain-light image
6152662	1 laser	1 movable or 2 fixed	Key and camera	Rotating holder or stepping motor	Detected light
6449381	1 direct light	1	Key	Corrected by software	Difference in brightness between the flat parts and the convoluted parts
6647308	1 laser	1 3D sensor	Key or system of mirrors	Rotation	Reflected laser beam
6895100	2 laser laminar beam	2	Key	Clamp move the key	Laser beam profile on the shank

2.2 Backlight-based systems

Patent number 5807042, “Method and apparatus for automatically making keys” in Figure 2.1 utilizes a horizontal light generator (38) and horizontal light receiver (39) positioned immediately above the pedestal (28) that allows light to be transmitted and received under the shank [1].

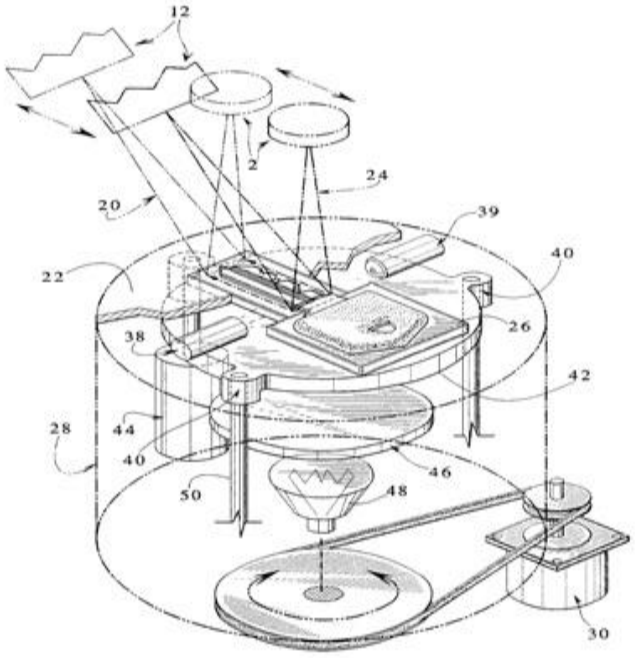


Figure 2.1 The backlight system using horizontal light generator [1]

If the shank of the object key is not positioned flatly on the blade section (25 in Figure 2.2), horizontal light above a minimum threshold will be received and the vertical elevation of the handle section (26 in Figure 2.2) of the transparent section will be incrementally lowered in relation to the output signal (Figure 2.2). Then the backlight (48 in Figure 2.1) is turned on and some pictures are taken. The number of pictures taken will obviously depend on the quality and capability of the camera used, but the system’s patent recommends five pictures be taken. The camera is moved in three-quarter-inch increments across a distance of about four inches and more pictures are taken at each location. These pictures are then digitalized and electronically merged together to generate one silhouette of the key and its corresponding output signal. Information about the shape, depth of cut, location of cuts, and location of the shoulder, are extracted and stored in memory [1].

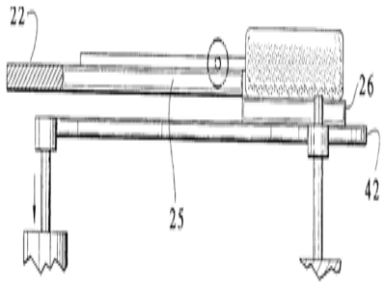


Figure 2.2 Transparent section of the backlight system [1]

Although there is no need to confine and fixture the key in a holder in this patent, it uses a source light and a receiver to mechanically adjust the key to an appropriate place.

In patent number 6064747 “Method and apparatus for using light to identify a key”, by using energized LEDs and backlighting the master key we can identify the type of blank key from a set of more than three hundred different types of blanks [30]. The key information is standardized to be comparable to other keys in memory. The light beam is projected at the key, and reflected and scattered from the surface. Referring to Figure 2.3, once the intersection of the light beam line with the key surface has been found, these positions must be mapped into depth of a cut out. The light beam (LB) is inclined at an angle θ. Less than 90 degrees, the point of intersection with the key surface will vary in the direction of the length of the key (Δ_l), which is a function of the depth (Δ_d) of the milling pattern at that point. The angle θ is constant and Δ_l is measured from the image. The depth is then calculated using the Equation (2.1). These calculations result in depth of milling as measured every one thousandth of an inch across the width of the key [2].

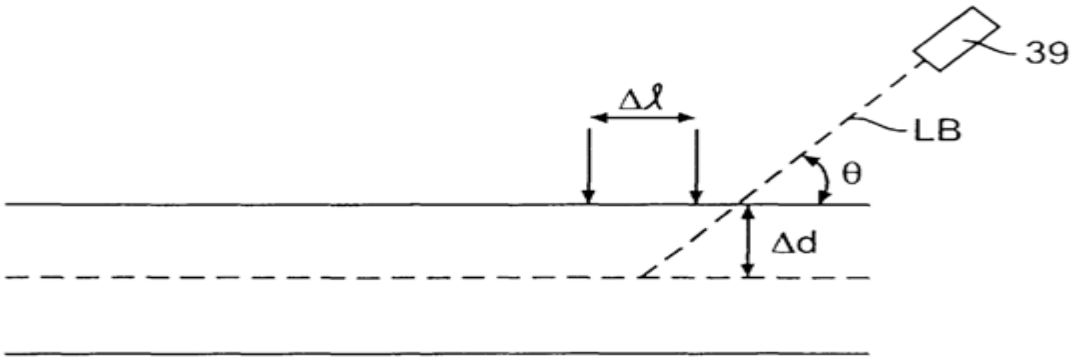


Figure 2.3 Intersection of the light beam line with the key surface [30]

$$\Delta_d = \Delta_l tan(\theta) \tag{2.1}$$

Another variation of this system uses the light beam system to record an entire three-dimensional surface image of the key which is shown in Figure 2.4.The light beam LB can be swept or moved across the length of the key to obtain multiple milling images so that the images can be averaged to improve the accuracy of the data. This embodiment would also provide a three-dimensional representation of the entire surface of the key [30].

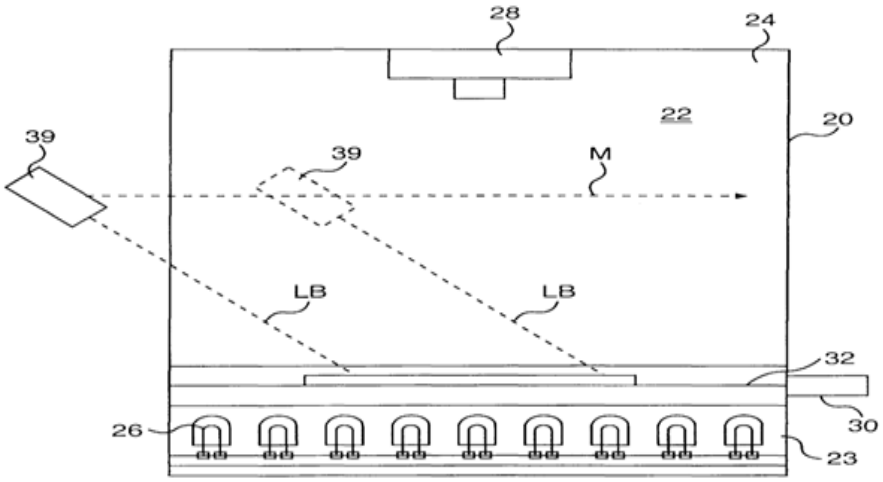


Figure 2.4 Intersection of the light beam line with the key surface [30]

In the two patents we reviewed, light is directed at the tip and it may produce glare or side glints. Patents 6175638 “Key imaging system and method” and 6406227 “Key measurement apparatus and method” avoid this problem by instead using a device which acquires a shadow image of a cross-section of a key by sending light rays along the groves and indentations of the key shank towards the screen. The cross-sectional image will be picked up by a digital scanner and stored for further identification or comparison. The computer matches the captured shadow images to the stored images [31], [28].

In one final example, in patent number 6836553 “Key identification system” which is close to the system we will be developing in this work [3], the system has two sources of uniform light are fixed relative to the housing and opposite of each other which can generate the image of the master key located between them which is shown in Figure 2.5. The sources of uniform light are capable of evenly illuminating the surface of an object. This even illumination can prevent the hot spots and glare that come from using a single light source such as laser. The master key can be positioned on the first luminous surface or a transparent support. When the power system is activated, the light emitted from the first luminous surface will backlight master key. Viewed from the opposite side of the master key from first surface, the backlighting of the master key outlines the profile of the master key (shape, size, head...). This backlighting process reveals the biting part of the key. When the power source is activated next, the second surface generates light that will illuminate the unsupported surface of the master key including any grooves and indentations. This system includes a receiver to capture the image of the master key that is generated by the first and second surfaces. The receiver can be a high resolution camera (PixelCam or Vitana) or a non-digital camera which is connected to a digitizer. The further the lens of the receiver moves away from the surface of the second luminous surface, the more the viewing angle of the receiver is restricted. Thus, the largest viewing area is obtained when the lens of the receiver is flush with the surface. The further the lens of the receiver moves away from the surface of the second luminous surface, the more the viewing angle of the receiver is restricted. Thus, the largest viewing area is obtained when the lens of the receiver is flush with the surface [3].

This final system is closest to the one I will be developing in this work. I use a digital desktop scanner which, much like the reviewed patent, has a transparent surface that the original key is placed on it and its light source is backlight. The source of light is a mobile uniform light that moves toward the length of the scanner’s surface and evenly backlights the object positioned on the surface step by step. A system in the scanner captures the objects on the scanner in line with the moves of the backlight system. The

data about the object like size, coordinates, color and shape are transmitted to a computer to process. In this patent the light sources are one or two uniform static lights that illuminate the surface of the key shank but in the project that I will explain, the light source of the scanner moves toward the length of the surface that the key lies on it and scan each part of the object step by step. So anything which is located on the surface of the scanner is illuminated and captured and the image is transmitted to a processor.

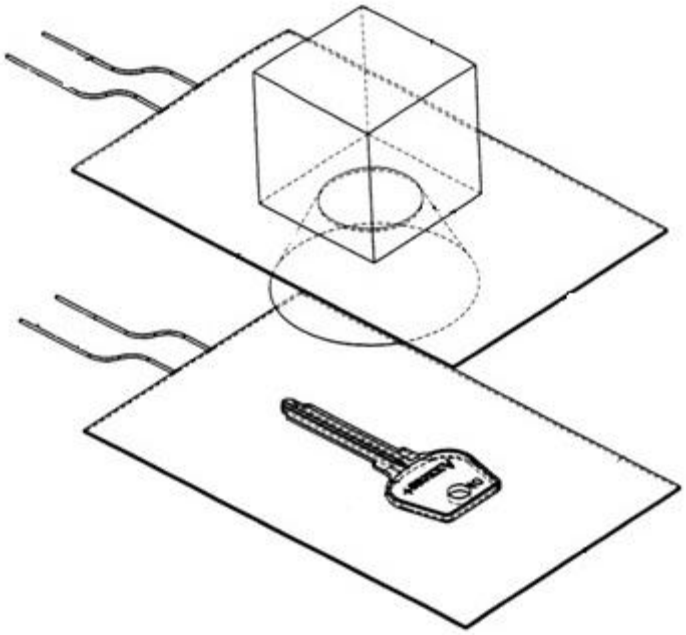


Figure 2.5 Source of the uniform light [3]

An alternative embodiment of this method is the one that comprises of second receiver which is aligned with a second opening in the second luminous surface shown in Figure 2.6.

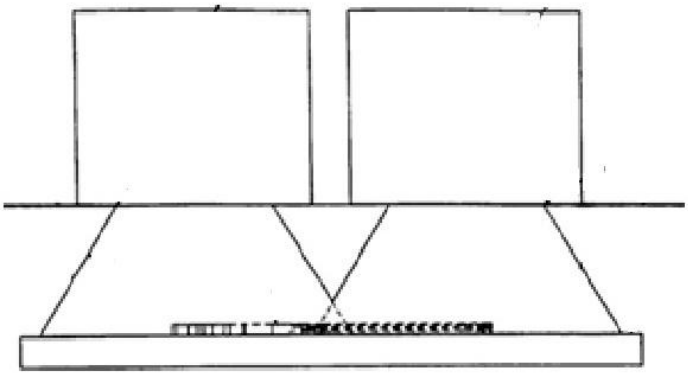


Figure 2.6 The lens of the second receiver is flush with the second surface [3]

The lens of the second receiver is also flush with the second surface. One receiver will capture an image of a first part of master key and the second receiver will capture the image of the second part of the master key (Figure 2.7). The control system analyses the image to extract identifying information from the original key.

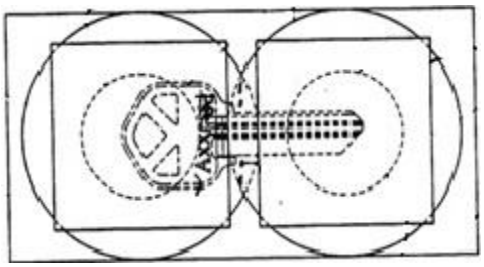


Figure 2.7 Capturing two parts of the key using two receivers [3]

There are two main downsides to this system. First, in a compact setup following Figure 2.5, the camera would be fairly close to the key and the key will not fit entirely in the camera's visible area. Consequently, we should make the key movable in the visible area, or make the camera and lights themselves movable. The second downside of this system is its duplication of the hardware and software: it uses two different lights and two different sets of algorithms to handle the front-lit and back-lit image, in order to perform a complete key measuring operation which means this device is not an efficient one.

2.3 Laser-based systems

Our first example of a system that employs laser as the light source is patent number 6152662, "Key duplication apparatus and Method". The measuring device employs a laser light beam which is collimated by collimating lens. The measurement device includes collection optics for receiving a light beam deflected by a target and a linear photo detector array that detects the light collected by the collection optic shown in Figure 2.8. The laser and photo detector array are both coupled to an electronic package that controls the output from the laser and receives the detected signal from the array. By detecting the position on the photo detector array of the deflected light from the target, the distance to the target may be estimated [29].

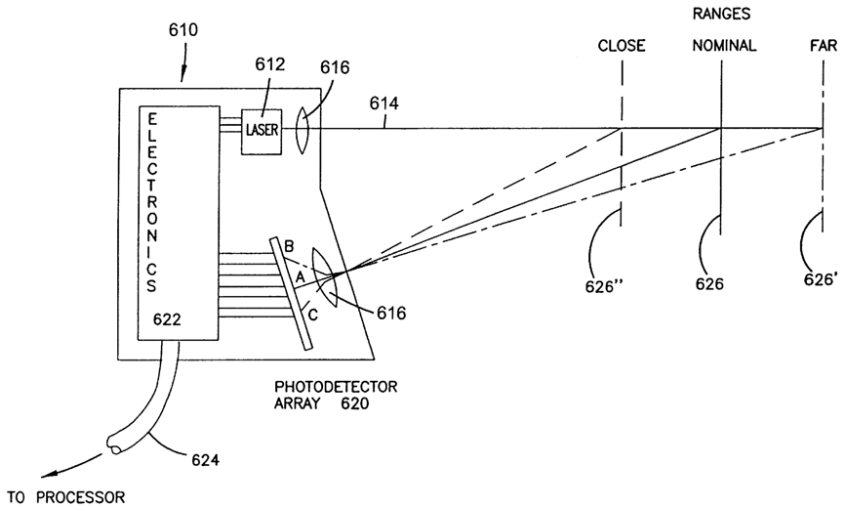


Figure 2.8 The measuring device (the laser and photo detector array) [29]

In this system, a light-based measuring device uses an optical interaction to produce an optical signal for determining the position of the point on the key. As it is shown in Figure 2.9 the portions along the side of key shank can be illuminated. The keyway is measured by positioning spots across the side of the key shank at the nominal distance from the measuring device (figure 2.10).

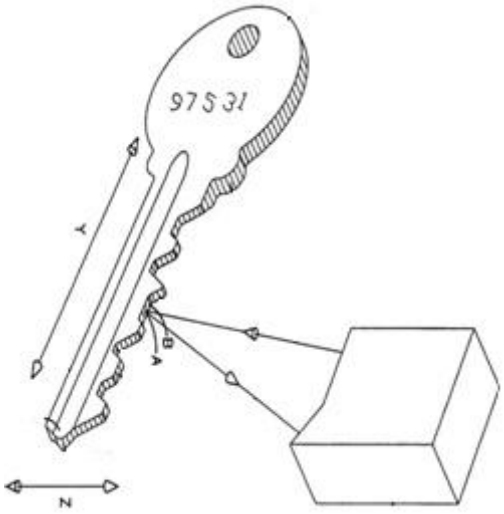


Figure 2.9 Determination of the position of the points on the key [29]

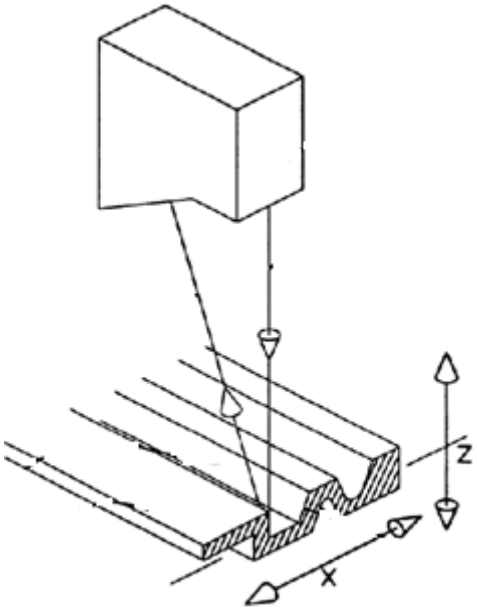


Figure 2.10 Measuring side of the key [29]

One image is taken from the side of the key and the other one is taken of the key tip in a direction along the axis of the key. The video measurement system requires the video camera and the key be movable relative to each other to permit side-on and end-on views to be taken (Figure 2.11).

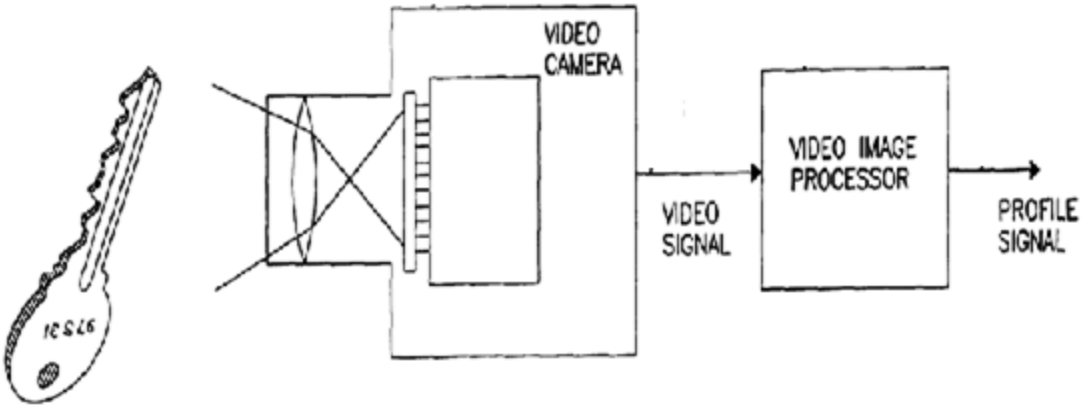


Figure 2.11 video measurement systems [29]

In patent number 6449381, "Key imaging system and method", the goal is to improve the accuracy of key identification by providing images of grooves and indentations as well as direct images of flat portions (parts 5 and 11 of Figure 2.12) of the shank [32]. The key is placed on a wide surface positioned facing up and toward light source which illuminates the key shank. The light rays will produce a direct image of key and shadow images of the grooves and indentations of the shank. The direct images are

formed when the rays are reflected back directly into a camera or scanner. The convoluted surface will not reflect the light rays directly into the camera as will a flat surface. The image will reveal a wide range of data relative to the key. The sharp distinction between the black convoluted surface of the grooves and indentations and the white flat surface of the key is distinguishable. The lateral surface of the shank of the key shown as number 4 in Figure 2.12 is not necessary for the actual operation of the key and therefore is not subjected to the wear and tear of regular use that may cause changes in the geometric shape of the grooves and indentations. Therefore, the geometrical features of the grooves and indentations on the shank which run throughout the whole length of the scanned lateral surfaces of the key allow for a precise automated identification. The color contrast between direct images (white) and those of the grooves and indentations (black) also provides for a precise automated identification. Finally, the evaluation of the direct images of the wide surfaces of the key will allow for the evaluation of additional characteristics of the key which were not previously available using prior art methods which rely on the examination of the front cross-section of the key.

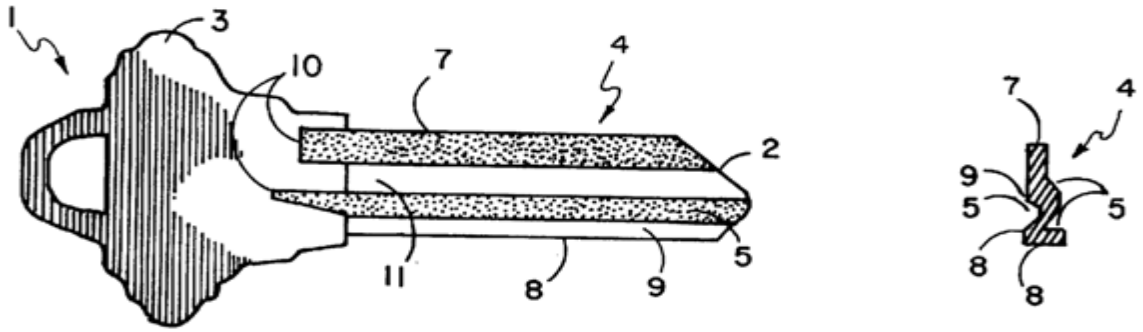


Figure 2.12 Image of the key when light rays are reflected back directly into a camera or scanner [32]

Providing a process using a database and CAD-Cam system for manufacturing original key is the goal of patent number 6647308, "Key manufacturing method". This system uses a holographic camera for capturing the surface of the key. In Figure 2.13, a beam emitter (14) emits a beam of light onto a series of mirrors and beam splitter (16). An incident beam of light is broken into two or more beams with at least one transmitted beam being used as an illuminating beam for the key (10) and the other beam being used as a reference beam. A signal beam is produced which is intercepted and read by the camera (12) and converted into electric data [24].

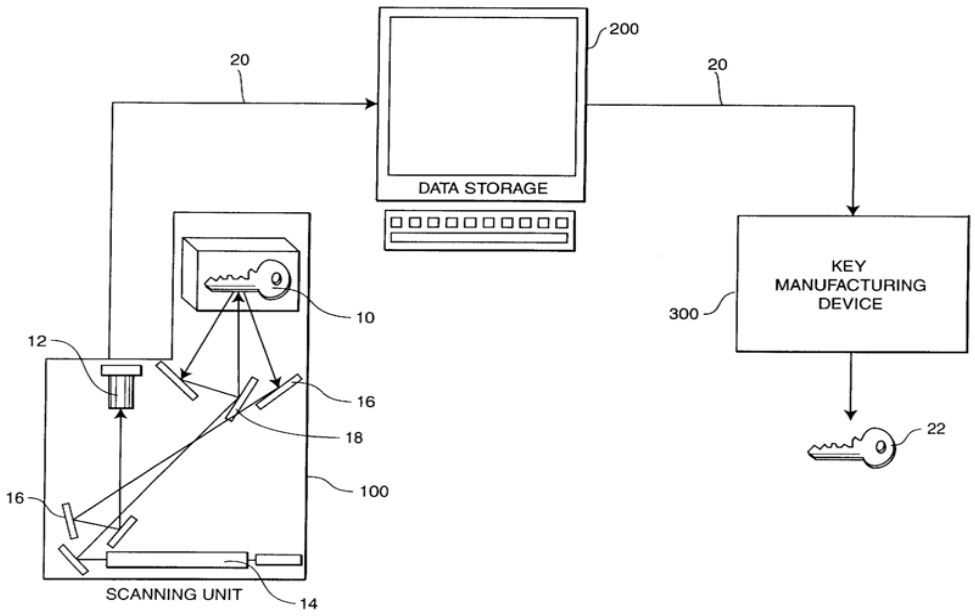


Figure 2.13 A beam emitter emits a beam of light into a series of mirrors and beam splitter [24]

Another variation of this technique with a 3-D scanner system allows for the highly accurate three-dimensional scanning of very detailed objects using "laser stripe scanning" techniques. The model-maker uses a scanning principle known as laser triangulation, in which a beam of laser light is projected as a stripe onto a three-dimensional object and viewed at an angle using a video camera. The image seen on the screen reveals the contour of the object where the laser light intersects the surface of the object [24].

Patent number 6895100, "Method to identify a key profile machine to implement the method and apparatus for the duplication of keys utilizing the machine" comprises two laser light sources (18 in Figure 2.14), which are facing each other and symmetrical about the plane in which the key is moved. These lasers can illuminate both sides of the key shank. Two video cameras (20) are fixed relative to the lasers and are inclined relative to the planes in which light beams lie. The illuminated profiles of the key can be ready by the two cameras and the data is then transmitted to processor computer [25].

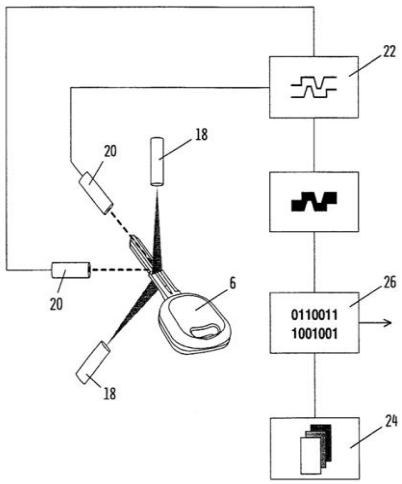


Figure 2.14 Two laser light sources symmetrical about the plane [25]

3 Image Processing Algorithm

3.1 Introduction

In this chapter I will introduce a function I designed that receives a scanned image of a key and finds the correct key blank in the defined database of key images. Then the function measures the cuts and dents of the original key shank. This information could then be sent to a cutting device to cut the measured indentations on the extracted key blank and duplicate the original key.

Figure 3.1 scanning the key using an open lid scanner

My algorithm receives as input an ordinary scanned image of a key, such as the one in Figure 3.1. It is worth noting that this image was scanned with the scanner's lid open, to avoid having shadows from the back of the lid that would blur the edges of the key in the picture. In the first step, I need to convert my image to binary format. This makes it easier to measure the positions of cuts and different parts of the key image. Moreover, it eliminates some problems, such as the local peaks in pixel intensities that are due to glare on the key image. Using a function filledBinary, that I implemented in MATLAB using MATLAB toolbox, I will have a solid binary image without any gaps and noise inside and outside the key body to find the real contour and perimeter of the key image.

Finding the correct key blank is one of the two important tasks in this project. I need to compare parts of the keys that are the same on all keys of the same model. Key heads of the same model, unlike the shanks, remain unchangeable and are good for comparison. I defined a function named cutHead in MATLAB that separates the head of a key from its body. This function identifies the line dividing the head and shank which is called the joint of the key and precisely cuts the head from the joint of the key body. I will get the seven moment invariants of the cut head to model the object in my image and calculate the distances between modelled head images. By comparing the results in the database, I will find the closest model to the modeled original head image and retrieve the corresponding key blank and its necessary information like size and number.

After the system finds the correct key blank, it will measure the cuts and depth of dents on the original shank image. In order to focus on the shank's dents I will apply the function that already identified the line dividing the head and shank and separate the shank of the key image. Using Canny Edge Detection technique, I will get a contour that only contains a connected shape of the edges of the shank with only one pixel width. Measuring the cut and dents in pixels is too jagged so using sub-pixel interpolation I will interpolate more precise coordinates for the points observed in the image as edge pixels. The sub-pixel

point will be the ones that will be converted to physical measures to know how to cut the key. At the end, I will save the positions of each sub-pixel points in an array. Figure 3.2 shows the pseudo-code of the algorithms developed in this project.

Input: Key Image, Key Head Database	
1.	Binary Image ← Convert Key Image to binary format
2.	Binary Image ← Fill Binary Image with white pixels
3.	Binary Image ← Eliminate noises and gaps from Binary Image
4.	Head Image, Shank Image ← Cut Binary Image
5.	Head Image Model ← Compute geometric invariants of Head Image
6.	Key Model, Key Measure ← Match Head Image Model in Key Head Database
7.	Shank Contour Image ← Obtain the connected contour of Shank Image
8.	Shank Contour Image ← Sub-pixel interpolation of Shank Contour Image
9.	Shank Measures ← Convert the positions of Shank Contour Image into millimeters using Key measure
Output: Key Model, Shank Measures	

Figure 3.2 Pseudo-code of the algorithms

The rest of this chapter will describe in details each of these functions. In 3.2 I will explain an algorithm which fills in the key image with white pixels, eliminates the noise and covers the gaps inside the body of key image. In section 3.3 I introduce a function that can cut the head of the key image precisely, and, in section 3.4 I will present the database of the key heads and the algorithm of finding the correct key blank in the database. Next, in Section 3.5 I will explain the algorithm to get the edges of the shank of the key and measure them precisely. Finally, while every part of the chapter will be illustrated by an example, in section 3.6 I will give a second complete example of the functioning of the system. Experimental results obtained using the system will be presented next in Chapter 4.

3.2 filledBinary

In this section I will explain the way I can convert the image's format to binary. The function filledBinary extracts the main object of the image and eliminates other small objects that are considered as noise. The advantage of using this function is that it covers all gaps and holes inside the key image but the real holes of the key head.

3.2.1 Converting the color or grayscale image into a binary image

The first step of the algorithm is to convert the grayscale or color image taken by the scanner into a binary image. In order to convert an image to binary, this function sets all pixels with intensity greater than a threshold to white and all other pixels to black. In this project, I found that even a pixel with a low intensity could be an edge of the shank and must therefore be considered as a part of the binary image. Consequently, I need to use a low threshold at this point. I experimented with different threshold values to convert the images to binary format and found that a threshold 0.9 of the intensity of a white pixel (i.e. 230 out of 255 in a grayscale image) generates a good image for the next steps of the algorithm [27]. To illustrate, Figure 3.3 shows one of the color images I received from the scanner, while Figure 3.4

shows the resulting binary image, and Figure 3.5 is the same image with the values inverted to be a white (1) key on a black (0) background. This inversion is made to simplify the rest of the algorithm.

Figure 3.3 A sample color key image

Figure 3.4 The sample key converted to binary image

Figure 3.5 The inverted binary key image

3.2.2 Hole and Noise Elimination

There are many small regions and individual pixels that are not connected to the main body of the key in the image. These tiny components are considered noise in the image. Moreover, as can be seen in Figure 3.5, there are a lot of gaps and holes inside the key. These are the result of lighter regions in the original scanned image, and they could lead to problems later in the algorithm. I deal with both of these problems in this step of the algorithm.

First, I need to eliminate all noise pixels and small regions that are apart from the key. Since the key is the single largest object in the picture, I can simply keep the biggest connected component found in the image and change the value of all other regions from 1 to 0 [14], [15].

Holes that are entirely surrounded by white pixels are assumed to be inside the key and are filled in. Applying this simple rule to the image of Figure 3.5 generates the image of Figure 3.6. It can be seen from a close-up of that image, shown in Figure 3.7 that although the key is more regular it is still not complete since my filling method cannot correct gaps that are connected to the background.

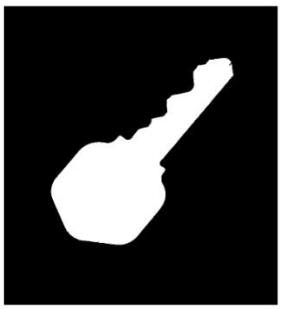


Figure 3.6 Binary image filled with white pixels

Figure 3.7 Incomplete gaps after filling the image

In order to fill in the gaps and keep the real holes inside the key head, I apply the 4-connected filter shown in Figure 3.8 to the entire Figure 3.5. This filter is designed to turn all four neighbours of every white pixel white. Although using this method leads to covering all gaps of the image, the key image will be one pixel wider than its original size. To restore the actual size of the key, I change the value of white pixels on the perimeter of the key image from 1 to 0, in essence eliminating the extra layer of white pixels added by the 4-connected filter.

0	1	0
1	1	1
0	1	0

Figure 3.8 The 4-connected filter for covering holes

Another drawback of using this method is that it fills in the holes inside the key head. The holes inside the head of the key will be important later in the algorithm, to compare the key with different types of key blanks. Consequently, at this step I need to restore the holes of the key head. I start with a simple assumption: that a real hole in the key's head will be large when compared to a hole due to an artefact

of the scanning and binary conversion. So I begin by finding the biggest hole in the key, and when I fill in all holes of the key image I skip the biggest hole that I just found. But moreover, in some types of keys like the one in Figure 3.5, there is more than one hole inside the key head. So after finding the biggest hole of the key, I get the number of pixels of the hole and retrieve other holes that are close in size to that largest one and I also exempt them from getting filled in. Figure 3.9 shows the result after I applied the algorithm on the Figure 3.5.

Figure 3.9 Binary key image without any gaps and noise

Using this method, I can get a binary image that has only one connected component and there is no gap and hole inside the key body. Another advantage of this method is that it keeps the holes of the key head. This feature helps me find the correct key blank more accurately. Totally, this image will help the system to have a better understanding from the positions of head, shank and edges of the key.

3.3 cutHead

In this project, I need to find a correct key blank for the original key in order to duplicate it. This will be done by comparing together the part of the keys which is identical on all keys of the same model; namely the head of the key, since the shank can have unique cuts on it. I need to tell apart the head from the body in the key image, and moreover the division between the two parts must be the same in all pictures of the same key model. The line dividing the head and shank is called the joint of the key. In this section I explain how I find it.

3.3.1 Properties of the image

An important challenge in this project is the ability to measure a key placed in any orientation. Indeed, in many existing key measuring systems, the key must be positioned exactly right to take the measures, and even a slight deviation results in the duplicate key having the wrong cuts and being unusable. This weakness of existing systems is one that our system will compensate for. A user of our system should be able to scan a key with any orientation, and our algorithm will automatically detect the orientation and adjust the measures accordingly.

I used the "regionprops" function from the standard Matlab library to compute the orientation [27]. The orientation of a region is the angle (in degrees) between the horizontal axis and the major axis of the ellipse that has the same second moments as the region [27], [14], [15], [9].

An image moment is the weighted average of the image pixels' intensities. A lot of useful information can be obtained from the moments of the objects. Simple properties of the image which are found via image moments include area (or total intensity), its centroid, and information about its axis of orientation [14], [15], [9].

0^{th} moment of the image gives us the area of the image.

$$A = \sum_x \sum_y I(x,y) \tag{3.1}$$

The centroid is simply the center of mass of the region as illustrated in Figure 3.10. The Centre of mass is given by the 1^{st} moments. Equation (3.2) and (3.3) give us the x and y coordinates of the centre mass, respectively.

$$\bar{x} = \frac{\sum_x \sum_y xI(x,y)}{\sum_x \sum_y I(x,y)} \tag{3.2}$$

$$y = \frac{\sum_x \sum_y yI(x,y)}{\sum_x \sum_y I(x,y)} \tag{3.3}$$

The second central moments are given in Equations (3.4), (3.5) and (3.6).

$$u_{xx} = \sum_x \sum_y x-\bar{x}^{\,2} I(x,y) \tag{3.4}$$

$$u_{xy} = \sum_x \sum_y x-\bar{x}\;\; y-\bar{y}\; I(x,y) \tag{3.5}$$

$$u_{yy} = \sum_x \sum_y y-\bar{y}^{\,2} I(x,y) \tag{3.6}$$

The orientation of an object is defined as the axis of the least second moment. The Orientation of the object is obtained using Equation (3.7).

$$\theta = tan^{-1}((u_{yy} - u_{xx} + \sqrt{u_{yy} - u_{xx}^{\ 2} + 4 * u_{xy}^{\ 2}})/2 * u_{xy}) \quad if\ u_{yy} > u_{xx}$$

$$\theta = tan^{-1}(2 * u_{xy}/(u_{xx} - u_{yy} + \sqrt{u_{xx} - u_{yy}^{\ 2} + 4 * u_{xy}^{\ 2}})) \quad if\ u_{xx} > u_{yy} \tag{3.7}$$

The relevant properties measured by the function I used are ”Orientation” (the angle in degrees ranging from -90 to 90 degrees between the x-axis and the major axis of the ellipse that has the same second-moments as the region), “Major Axis Length”, “Minor Axis Length”, ”Eccentricity” and ”Centroid” [27], [15]. These properties are illustrated with a sample image in Figure 3.10. The left side of the figure shows an image region and its corresponding ellipse. The right side shows the same ellipse, with features indicated graphically:

- The solid blue lines are the axes.
- The red dots are the foci.
- The orientation is the angle between the horizontal dotted line and the major axis.

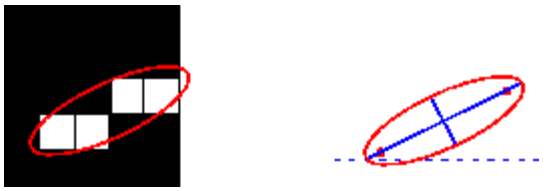


Figure 3.10 Ellipse superimposed over a sample image (taken from the Matlab documentation[1])

The Major Axis Length and the Minor Axis Length are the lengths (in pixels) of the major and minor axes, respectively, of the ellipse with the same normalized second central moments as the region. This is the ellipse shown in the example of Figure 3.10.

Figure 3.11 shows the ellipse that has the discussed properties of the sample image. This ellipse is superimposed over the key image and its centre is centre mass of the key [7].

[1] http://www.mathworks.com/help/toolbox/images/ref/regionprops.html#bqkf8ji

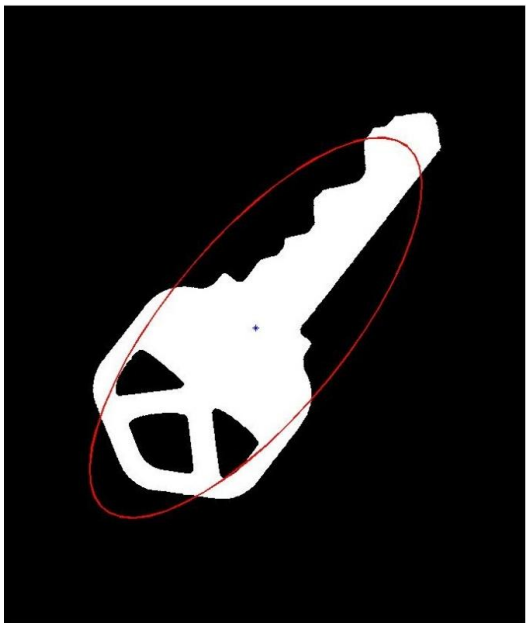


Figure 3.11 Superimposed ellipse and the centroid over the image

3.3.2 Adjusting the angle of the key image

In this project, in order to move in correct direction and measure key image pixel by pixel I need to know the orientation of the image. Next, the function changes the image orientation to a standard angle to find the top and bottom pixels of the image located on the major axis line. This approach is essential because in some angles and locations of the key image (for example exactly horizontal or vertical), it is hard and risky to find the correct top and bottom of the key. I will talk about the sensitive locations in next subsection.

This function rotates the orientation of the key image about the centre of the image to a defined angle. Rotating an image is one of the geometric transformations. Rotating or transforming an image changes the coordinate system of the original image and when we rotate it back to its original orientation, we usually lose some values and edges. In this chapter, the goal is to cut the head from the body of key image. I need to have the head of the key just for comparing among different key heads so it does not matter if I face some changes in the coordinates and values. In addition, as I will explain in section 3.4, I use geometric invariants to model the head of the key images so geometric transformations have no impact on the final outcome.

3.3.3 Adding zero pad array to the image

After adjusting the orientation of the key image, the function starts measuring the edges of the key pixel by pixel. It does so by moving along the Y-axis of the image in a direction perpendicular to the shank of the key or parallel with minor axis line of the key image. The objective is to cover the image of the key from top to bottom. Unfortunately, it is not always possible to do so this simply. In Figure 3.12 for example, only part of the image is accessible using the measuring algorithm, and only a partial edge is detected in red. Likewise, in Figure 3.13, most of the key shank, including the top end of the key, cannot be detected.

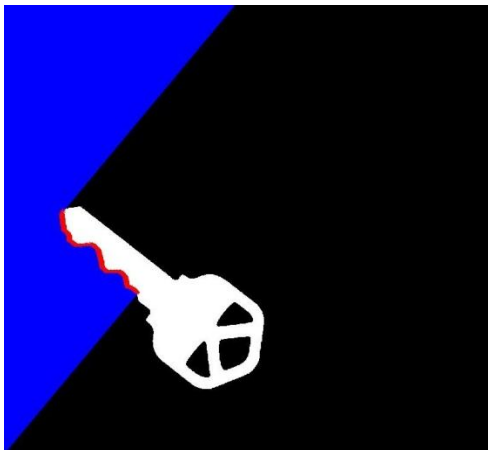


Figure 3.12 Covered and uncovered parts of the key when the key is at the corner

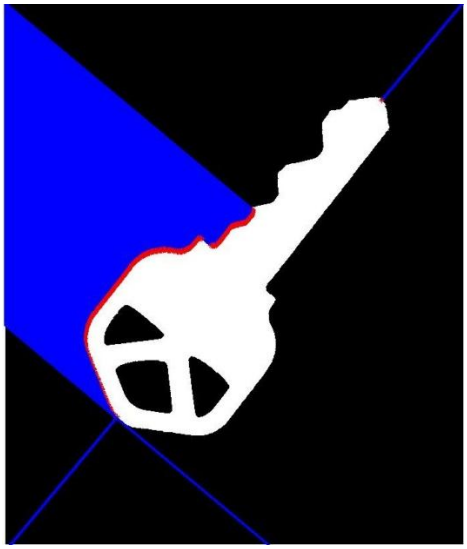


Figure 3.13 Parts of the key image that cannot be detected and is not accessible from y-axis

In order to prevent these problems from occurring and to be able to detect all parts of the key image, I first adjust the angle of the main object and change it to the standard orientation that I already defined for my program. After that I add zero pad arrays to all four sides of the image to have my main object at the centre of the image [15]. Then I double the length of the image using zero pad arrays. It is shown in Figure 3.14 that by using this technique and adjusting the angle of key image, the key will be detected completely by moving along the Y–axis and I can get the top and bottom of the key image. Finally, after cutting the head from the body of the key image, the function returns the key to its original orientation and dimensions.

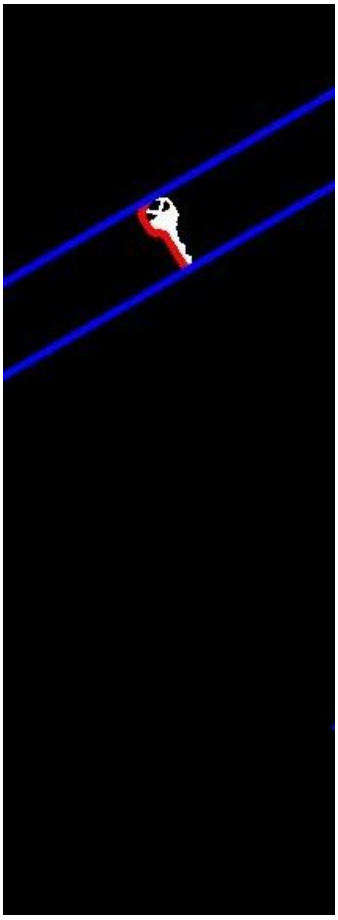


Figure 3.14 Adding zero pad arrays to the image

3.3.4 Cutting head from the key image

Since I have previously computed the key's properties, I can accurately find the line that cuts the head of the key. First, I need to identify the coordinates of the top and bottom of the key. These coordinates are somewhere on the major axis line, so it is simply a matter of following that line as I illustrate in Figure 3.15.

I already noted that after cutting the head from the body of the key image, the function returns the key to its original orientation and dimensions. But In order to illustrate next operations on the image clearly, in Figure 3.15 I present the points detected in Figure 3.14 superimposed on the key picture I've used in the previous examples.

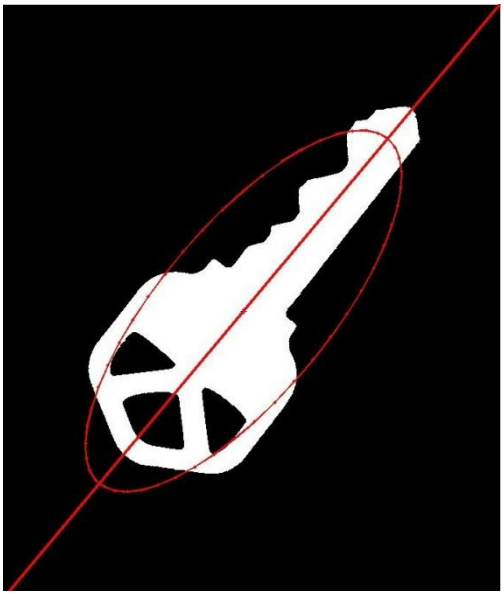


Figure 3.15 Major axis line and superimposed ellipse over the key

Following the processing of the previous section, I know the image only contains one connected white component. The top and bottom of the key are identified as the first and last white pixels on the major axis. These two coordinates in turn allow us to calculate the coordinates of the geometric centre and the length of the key image (Figure 3.16).

Figure 3.16 Top, bottom and centre points of the key image

I reviewed different types of keys and found that the joint of the keys are located very near the geometric centre. Specifically, in this project, I consider "very near" to be a neighbourhood of $1/6^{th}$ the key image's length. Since I have the top and bottom of the image I can calculate the length using the Euclidian distance between the points, as in Equation (3.8).

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 + y_1)^2} \tag{3.8}$$

I search a neighbourhood of 1/6th of *d* around the geometric center, or 1/12th before the centre point to 1/12th after the centre point following the major axis. In Figure 3.17 the start and end point of that interval are shown with dots, and Figure 3.18 gives a close-up view of that interval.

Figure 3.17 Start and end points for finding joint of the key

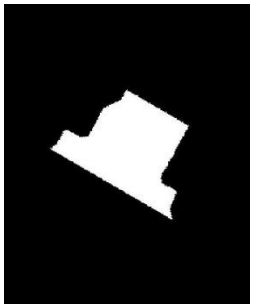


Figure 3.18 Image cut from 1/12th of the distance before and after the centre point of the key

Another observation, which is clearly evident on Figure 3.18, is that there is an important transition between the head and shank of the key, marked by an important change in the key's width. This division is what I will find in order to pinpoint the joint. Focusing on this particular neighbourhood of the key is useful to make the detection more accurate. There are variations of the key's width both in the shank (for the dents) and the head (for embellishments), but in this smaller neighbourhood the most important change will be the joint.

I start scanning this neighbourhood of the key row by row. Rows are defined as having the same orientation as the minor axis of the ellipse and are perpendicular to the shank (major axis) of the key. For each row I count number of white pixels using Equation (3.8) and save this number in the corresponding row of an array. This gives me an array where each element corresponds to the number of white pixels in a row of the cut shown in Figure 3.18. Next, I compute the difference between each two successive elements of the array. The highest difference value is the position of the joint I am looking for.

Table 3.1 shows the array corresponding to Figure 3.18. The length of the array is 118 pixels, the same as the distance between the start and end points. The 34^{th} and 35^{th} elements of the array (written in bold) show the sharpest difference between each two successive elements of the array. Consequently, the joint of the key is located between those two elements.

Table 3.1 Number of pixels in each row of the division

Column 1-24	Column 25-48	Column 49-72	Column 73-96	Column 97-118
140	130	82	84	76
140	128	83	84	76
138	128	84	84	74
138	129	83	84	74
137	129	83	85	73
135	129	83	84	73
135	127	83	83	72
134	127	83	83	72
132	126	83	83	71
132	**119**	84	83	72
131	**109**	83	83	72
131	101	83	82	72
131	92	84	82	72
130	88	83	81	71
129	88	83	81	72
130	84	84	81	72
130	84	83	79	72
129	85	83	80	73
129	83	84	80	72
130	83	83	78	72
130	84	83	78	72
128	82	83	77	72
129	83	84	77	
129	82	84	76	

To illustrate, the joint of the key in Figure 3.17 was found using this method and marked on Figure 3.19.

Figure 3.19 Cut-line that cuts the head from the key

A final observation is that the key is not balanced, meaning that the geometric center and the centroid are not at the same point. The head contains more of the mass of the key, and consequently the centroid will be further down the shank than the geometric center. I can use this fact to figure out on which side of the joint the head is: from the geometric center and following the major axis, it is on the side of the joint opposite to the centroid.

I can then finally separate the head and the shank. The head obtained for the sample key I have been using in this section is shown in Figure 3.20.

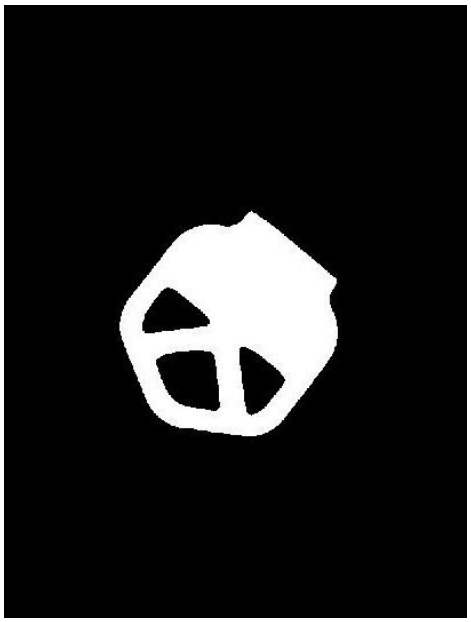


Figure 3.20 Head of the key cut from the key image

3.4 findKey

In this section, I will introduce a function which receives the image of the head cut from the original key and matches it to the correct key blank in a database. We can then retrieve information about the key from the database, including its physical size, key number and an image of the corresponding key blank. I use geometric invariants to model and compare the original key with other keys in the database. In this thesis, the database contains information from seven different sample keys.

This challenge is similar to that encountered in face-recognition systems, which try to match a person's face to a database of images of known individuals. Some typical approaches used in that area include Illumination invariant face recognition using Discrete Cosine Transform (DCT) and Principal Component Analysis (PCA) that uses low frequency components of DCT to normalize the illuminated image [26]. This approach uses PCA for recognition of images but the presented method in this thesis will use moment invariants to model the images. Another approach in face recognition pattern analysis area is Line Edge Map (LEM) that is generated for face coding and recognition [12]. This concept is robust to lighting condition changes and size variation which is similar to Hu's seven moment invariants which are invariant under size and scale.

3.4.1 Eliminating useless space of the image

As can be seen in the example of Figure 3.20, the images of the cut head are surrounded by large empty areas. These areas have no effect on the next steps of the algorithm, aside from increasing computation time, memory space and the database size. In order to have smaller images with the main objects, I used a function named Draw Bounding Box from MATLAB Central website and developed another function named boundingbox that eliminates the black area that surrounds the key head [27], [15], [21]. Figure 3.21 shows the result after applying boundingbox function over Figure 3.20.

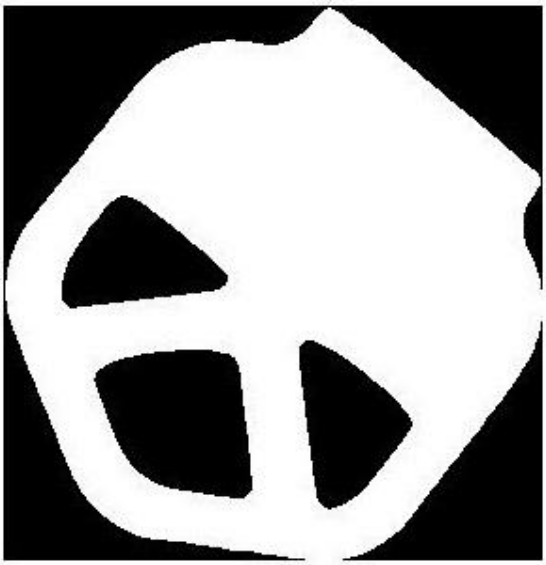


Figure 3.21 Head of the key after applying boundingbox function

3.4.2 Moment Invariants for modelling the head of the key image

Now that I can get the image of the head cut from the key, I need to extract some information and factors from the image in order to compare the head of the original key image with other heads cut from different key types. The extracted information should be unique for each type of key image

because I will look for the corresponding key blank in my database that has the same information as the original key.

I use image moment and moment invariants to model the image of the heads. Using the Hu set of moment invariants I can calculate moments which are invariant under translation, changes in scale, and also rotation [19].

If the size of my image is M * N, the 2-D moment of order (p + q) is defined as Equation (3.9) [14], [15], [19]:

$$m_{pq} = \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} x^p y^q f(x,y) \tag{3.9}$$

where p and q are integers.

The corresponding central moment of order (p +q) is defined as Equation (3.10):

$$u_{pq} = \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} (x-\overline{x})^p (y-\overline{y})^q f(x,y) \tag{3.10}$$

for p = 0, 1, 2, ... and q = 0, 1, 2, ..., where

$$\overline{x} = \frac{m_{10}}{m_{00}} \text{ and } \overline{y} = \frac{m_{01}}{m_{00}}$$

And the normalized central moment of order (p + q) is defined as Equation (3.11).

$$\eta_{pq} = \frac{\mu_{pq}}{\mu_{00}^{\gamma}} \tag{3.11}$$

where

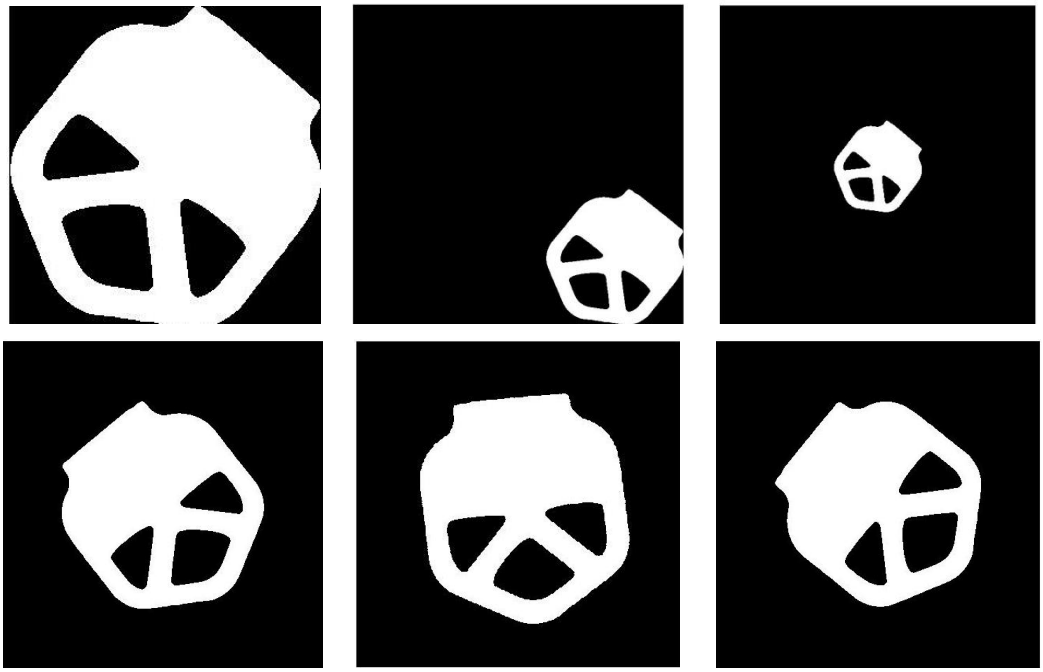
$$\gamma = \frac{p+q}{2} + 1$$

for p + q = 2, 3, ...

The Hu set of seven 2-D moment invariants can be derived from the set of Equations (3.12) [14].

$$
\begin{aligned}
I_1 &= \eta_{20} + \eta_{02} \\
I_2 &= (\eta_{20} - \eta_{02})^2 + (2\eta_{11})^2 \\
I_3 &= (\eta_{30} - 3\eta_{12})^2 + (3\eta_{21} - \eta_{03})^2 \\
I_4 &= (\eta_{30} + \eta_{12})^2 + (\eta_{21} + \eta_{03})^2 \\
I_5 &= (\eta_{30} - 3\eta_{12})(\eta_{30} + \eta_{12})[(\eta_{30} + \eta_{12})^2 - 3(\eta_{21} + \eta_{03})^2] + \\
&\quad (3\eta_{21} - \eta_{03})(\eta_{21} + \eta_{03})[3(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2] \\
I_6 &= (\eta_{20} - \eta_{02})[(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2] + 4\eta_{11}(\eta_{30} + \eta_{12})(\eta_{21} + \eta_{03}) \\
I_7 &= (3\eta_{21} - \eta_{03})(\eta_{30} + \eta_{12})[(\eta_{30} + \eta_{12})^2 - 3(\eta_{21} + \eta_{03})^2] - \\
&\quad (\eta_{30} - 3\eta_{12})(\eta_{21} + \eta_{03})[3(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2].
\end{aligned}
\tag{3.12}
$$

I used a custom M-function called invmoments [15] that implements these seven Equations. This function gets an image as an input and the result is a seven-element row vector containing the moment invariants of the input image. These seven moment invariants are insensitive to translation, scale change, mirroring and rotation [27], [15], [19], [11]. To demonstrate, I applied these geometric changes to Figure 3.21 to obtain the new images of Figure 3.22, and I compared the seven 2-D moment invariants of each of these images in Table 3.2. The results are almost identical in all cases, aside from a sign difference.



a b c
d e f

Figure 3.22 (a) Original image. (b) Translated image. (c) Half-size image. (d) Mirrored image. (e) Image rotated 45°. (f) Image rotated 90°

Table 3.2 Seven moment invariants of the images in Figure 3.21

Moment Invariant	Original Image	Translated	Half Size	Mirrored	Rotated 45°	Rotated 90°
I_1	0.6832	0.6832	0.6833	0.6832	0.6833	0.6832
I_2	3.7572	3.7572	3.7608	3.7572	3.7509	3.7572
I_3	5.3004	5.3004	5.2968	5.3004	5.3024	5.3004
I_4	3.6459	3.6459	3.6468	3.6459	3.6470	3.6459
I_5	8.1330	8.1330	8.1341	8.1330	8.1360	8.1330
I_6	5.5247	5.5247	5.5274	5.5247	5.5226	5.5247
I_7	8.7231	8.7231	8.6998	-8.7231	8.7187	8.7231

3.4.3 Distance of moment invariants between two images

In order to compare the seven moment invariants of two images, I implemented a function called varmoments. In Equation (3.12), $I_1, I_2, I_3, I_4, I_5, I_6$ and I_7 show the seven 2-D moment invariants of an image. Using Equation (3.13) I can compute the distance between seven moment invariants in two images.

$$V = \sqrt{(I_{11} - I_{21})^2 + (I_{12} - I_{22})^2 + (I_{13} - I_{23})^2 + (I_{14} - I_{24})^2 + (I_{15} - I_{25})^2 + (I_{16} - I_{26})^2 + (I_{17} - I_{27})^2} \quad (3.13)$$

For example, the difference between the original key image in Figure 3.22(a) and the half-size key image in Figure 3.22(c) computed by my function is 0.0241

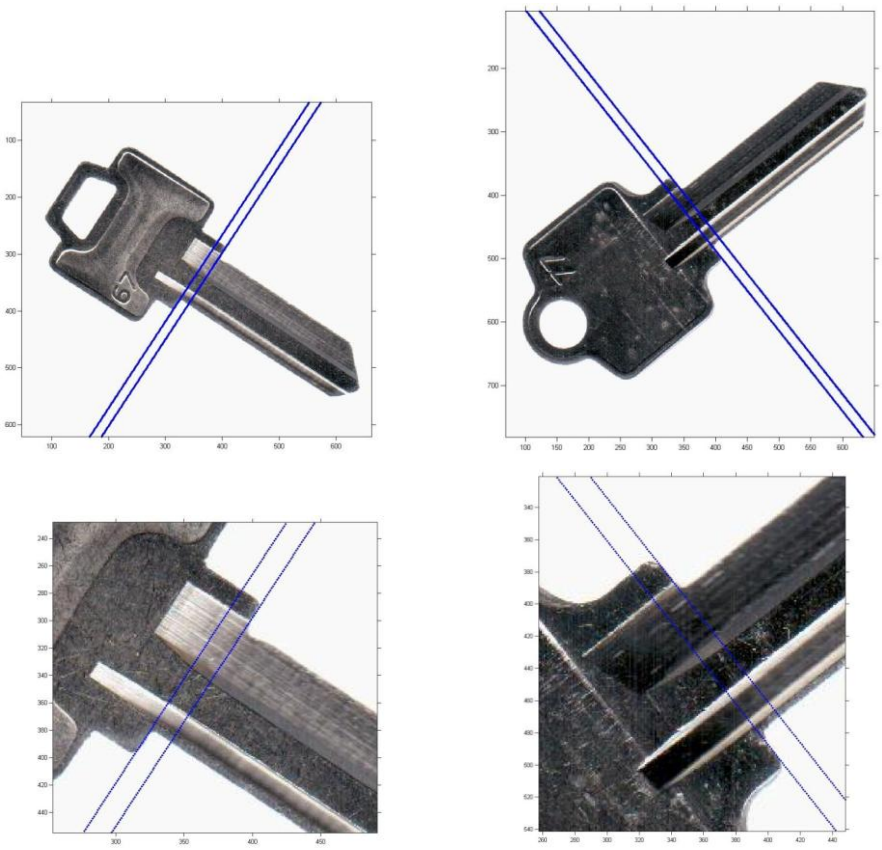
Now, I can compare the outputs from different comparisons. The smallest result shows the closest model to the modeled original image.

3.4.4 Database and finding correct key blank

I created a database of key images and information for my project. Initially, it contains the information from seven different keys I've worked with. I scan these seven keys three times each using a digital desktop scanner, and I applied the procedure of Sections 3.3 and 3.4 to extract the heads from the key images. The reason I make three different copies of the keys is that the glare and light reflection on each one is different, and these differences create variations in the moment invariants that confuse the identification of the correct blank. Indeed, when I computed the distances of their moment invariants, I found that, in some cases, the distance between two keys of different types is smaller than the distance of two pictures of the same key, and this would lead to recognition errors. To compensate for that problem, I decide to compare the original key's head with several different images of the blanks' head and get the average of the distances. The group that got the smallest average was the correct key blank. In general, the result is a lot better than by comparing with just one key. After testing and comparing key heads, I found that getting the average of two images of the head reduced the rate of misidentifications from 6.3% or 5.2% to 3.1%, and getting the average of three images eliminates almost

all errors, so that there are only minimal benefits to using four images or more. Later on in this section, I will explain the source of this last error and an additional algorithm I used to deal with the last remaining errors.

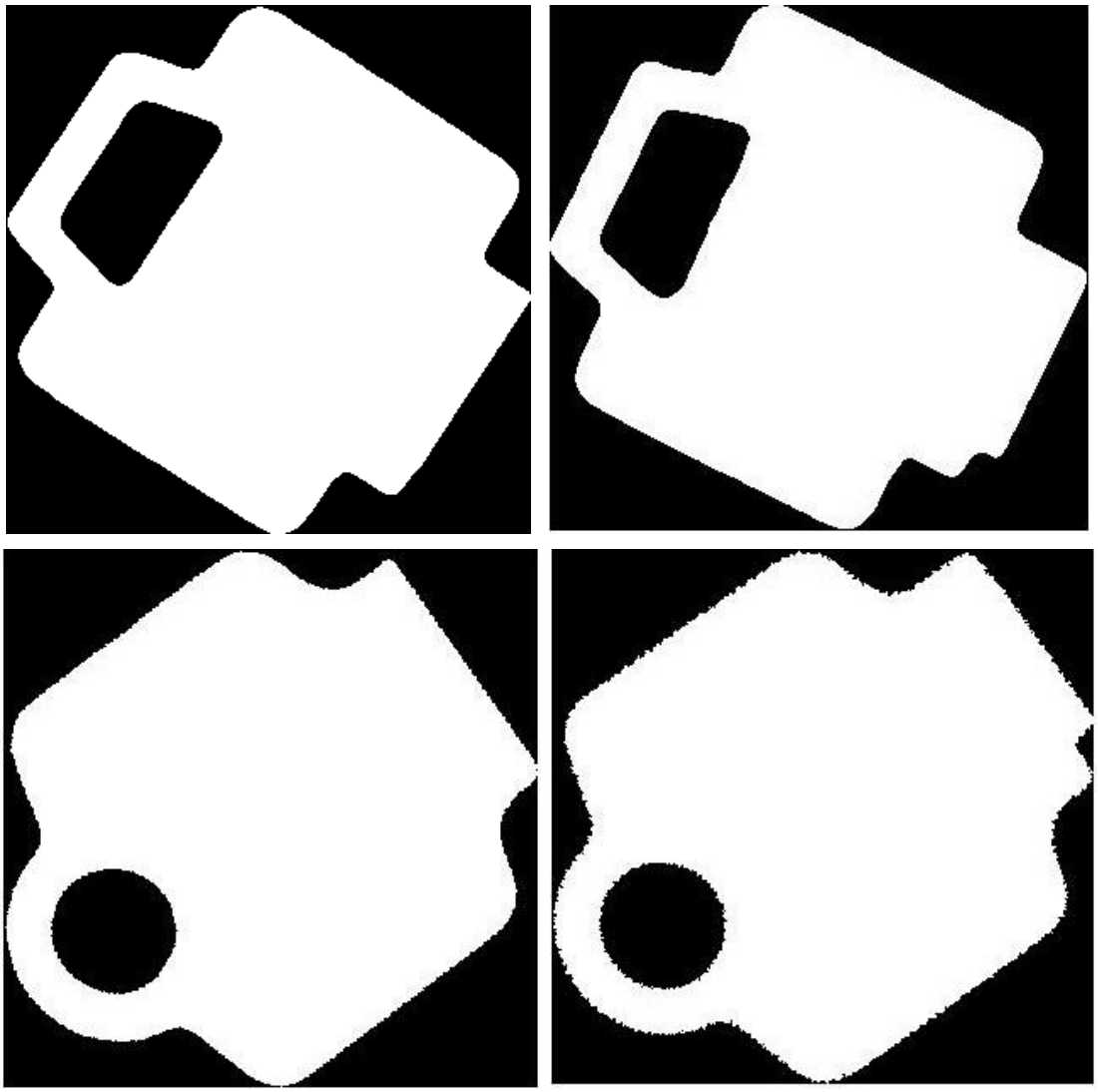
In Figure 3.18 I already illustrated the division which is a neighbourhood of $1/6^{th}$ of the length of the key image around the geometric center in order to pinpoint the joint. I consider the joint of a key as the point I can cut the head from the body of the key image. In some key types there are two joints in the division. Figure 3.23 shows two samples of these types of keys.



a b
c d

Figure 3.23 (a) Key number 67. (b) Key number 77. They both have two joints inside their divisions. (c) and (d) are close-ups of (a) and (b), respectively

I already explained that I find the joint by obtaining the maximum difference between two successive numbers of white pixels in each row perpendicular to the shank. However, small differences arising between different images of one key make it possible that the function finds the maximum on the first or second joint. To illustrate, I scanned each key of figure 3.23 twice and found the joint of each image. As figure 3.24 shows, two possible joints were identified for each key.



a b
c d

Figure 3.24 (a) Key head No.67 cut from the top blue line. (b) Key head No.77 cut from the bottom blue line. (c)Key head No.77 cut from the bottom blue line. (d) Key head No.77 cut from the top blue line.

Figure 3.24 shows the difference between two heads cut from the same key image. Now, I want to compute moment invariants of these four images and show their distances in a Table 3.3.

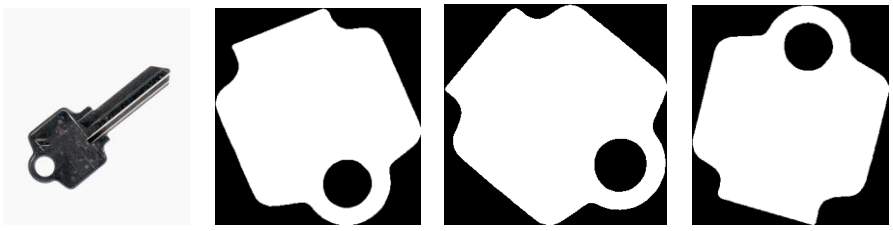
As it was explained in sub-section 3.4.3, in order to find the difference between two images, I use the function varmoments. I expect that the distance between the head images in Figure 3.24 (a) and (b) gives me the smallest distance because the head images are taken from the same key patent. Unfortunately the results shows that the distance between images in Figure 3.24 (b) and (d) are the smallest one. It means it is possible that the function matches the key in Figure 3.24(b) to Figure 3.24(d) which is absolutely incorrect. I have the same scenario for Figure 3.24 (c) and (d). The distance between Figure 3.24 (a) and (c) are smaller than the distance between Figure 3.24 (c) and (d).). It is clear, then, that the extra part of the joint has a major impact on the values of the invariants.

Table 3.3 Comparing moment invariants of two-joint key heads

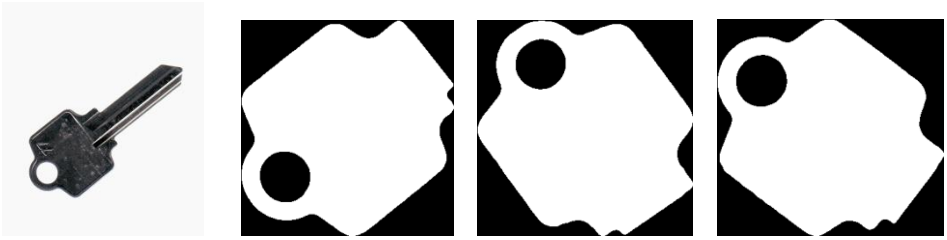
Moment Invariants	Figure 3.24 (a)	Figure 3.24 (b)	Figure 3.24 (c)	Figure 3.24 (d)	Varmom ((a), (b))	Varmom ((c), (d))	Varmom ((a), (c))	Varmom ((b), (d))
I_1	0.7504	0.7512	0.7680	0.7660	1.4404	2.5414	1.8250	1.3876
I_2	4.7026	4.5647	4.2290	3.5128				
I_3	5.3907	4.4814	6.6022	4.7584				
I_4	4.3326	4.3494	4.6487	4.7037				
I_5	-9.1947	-8.7707	10.2776	-9.4728				
I_6	-6.7009	6.6719	6.7695	6.4627				
I_7	10.5732	9.5494	11.1741	9.8319				

I explained that in the database I place three heads of the key images for each key type. In order to solve the problem of recognizing the wrong key blank because of a cut at the wrong joint, I created two sets of images for keys that have two joint divisions. One set has three head images taken from the top joint and the other one has three head images taken from the bottom joint. Both of these sets refer to one key blank in the database. The database of the project is shown in Figure 3.25 and the added sets are shown in last two sets in Figure 3.25.

D1



D2



D22

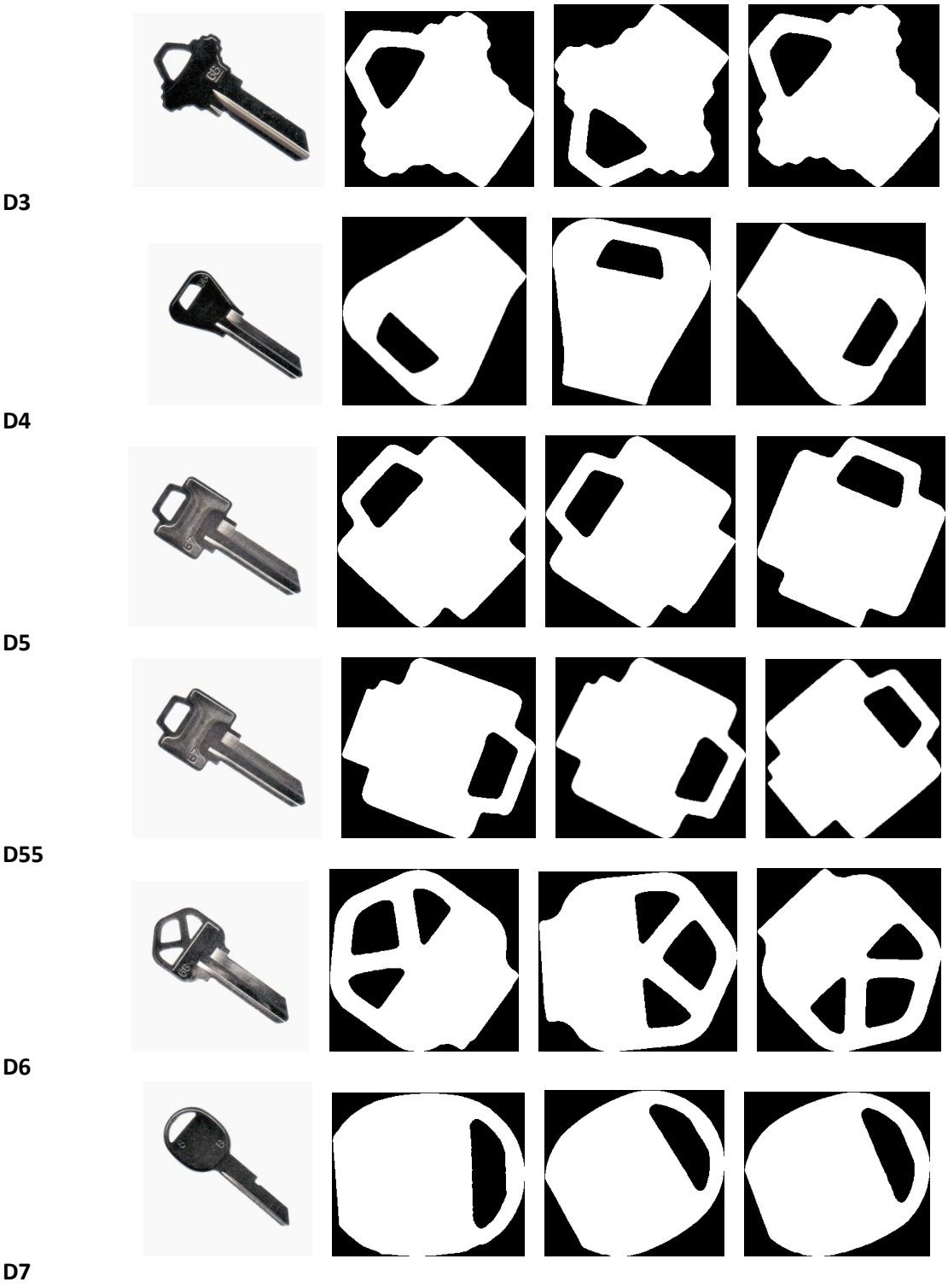


Figure 3.25 Sets of images in database

I explained earlier that there remains a small misidentification error after comparing the original key head with three blank key heads. In 3.2.2 I explained that in order to keep the holes of the key head, I do

not use the algorithm that covers all the holes of the image. Instead, I run the 4-connected filter over the image to fill in small holes and gaps while keeping the main holes of the key head. But in some cases, bright spots in the image due to glare and reflected scanner light create holes and gaps that are too large to be covered by the filter (Figure 3.26).

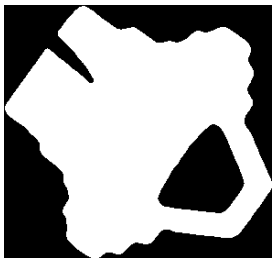
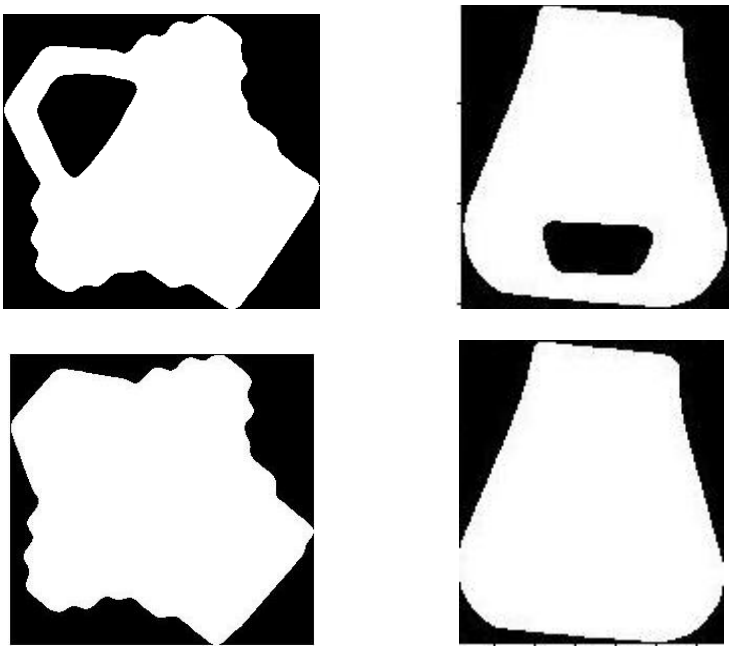


Figure 3.26 A gap not covered by the filter

The errors mentioned before are because of these false holes in the key heads that remain unfilled. They cause variations in the moment invariants of the head, and in some cases these variations lead to the system misrecognising the key. In order to deal with this problem, I completely fill in the three key heads in each key set and locate these filled key heads beside the first three unfilled key heads [27]. Now in each group I have six key heads of one key type that three of them are filled with white pixels.

The other reason that I compare the filled heads as well as unfilled heads is that I add new information to the first comparison method. In Figure 3.27 and Table 3.4 I show an example where the distances between two different key heads are close and the possibility of error exists. But when I compare the filled key heads, the distance is sharp enough to recognize that these are two different key heads.This example shows how the filled-head comparison can complement the previous test. By using the average of all six comparisons (the three not-filled heads and the three filled heads) I can increase the chance of picking the correct match in my database.

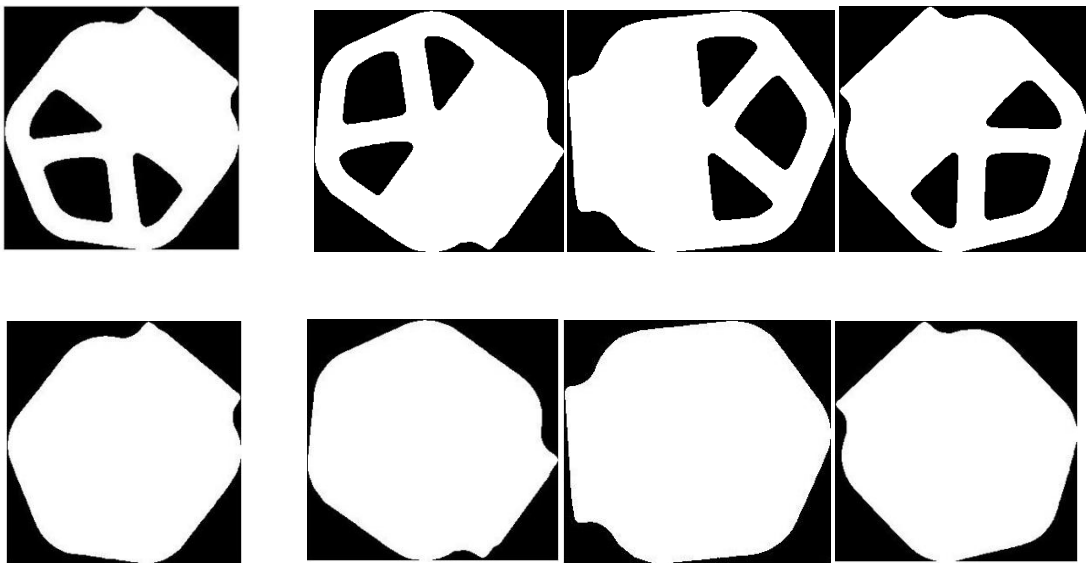


a b
c d
Figure 3.27 Filled and unfilled head images of two key types

Table 3.4 Distances of filled and unfilled head images of two key types

Var(Figure 3.27 (a), (b))	1.4208
Var(Figure 3.27 (c), (d))	4.3395

The function that I defined for the database picks the original key head image and one of the sets in the database (as shown in Figure 3.25, each set contains three head images of the same key type) and gets the distances of moment invariants between the original image and the three head images existing in each set of the database. Then, the function fills in the original head image and the head images in the set with white pixels. Again, the function gets the distances of moment invariants between the filled original head image and the filled head images in the set. Finally, there will be six distances for each set. Figure 3.28 shows a sample of original head image and a set of head images that will be compared with.



a_1 a_2 a_3 a_4
b_1 b_2 b_3 b_4
Figure 3.28 a_1 is the original head image that is compared with a_2, a_3 and a_4. b_1 is filled original image that is compared with b_2, b_3 and b_4.

The average of these six distances is preserved as a double-precision constant. Images in Figure 3.28 (a_1) and 3.28 (b_1) will compare to all other sets and the function will keep their average of distances, too. The reason that I compare the original key head with three different heads of the same type of key is to decrease the possibility of error in finding the correct key blank. The reason that I used filled heads three times (like the unfilled head images) is that, I want to keep the balance between the effect of filled and unfilled distances of moments in average and final result. Comparing the filled binary head images is an algorithm to compare the moment invariants of the complete patterns of the keys. Finally, the set which got the lowest average is considered as the closest set of images to the original head image. The corresponding color key blank image of the selected set, plus the physical length of the key blank (in millimetres) and the key's patent number and pixel length are all displayed on the screen as the outputs of the function. Pixel length is the ratio between the physical length of the key blank in millimetres and the length of the original key computed from top and bottom points of the key image in pixels. Using pixel length, I can get the real and physical size of each pixel in my original key image. Figure 3.29 shows the sample key blank image with the information I can get after finding the correct key blank.

Figure 3.29 Obtained key blank with the information

In addition to the moment invariants, I developed a second function that works as a supplement in order to enhance the accuracy of the key blank detection. That function counts the number of pixels in head's holes and number of head's white pixels. The output of the function is the ratio of these two numbers. The ratio is unique for each key head image within a margin of error, and can thus be used as another factor to compare different key heads and get the right key blank. I use this function to resolve ambiguous cases where the difference between moment invariants of two different keys is too small to be confident the algorithm picked the correct one.

3.5 Shank

In this section, I introduce the algorithms and procedures that allow me to find the edges and to measure the depths of the dents and indentations of the original key image in physical measures.

In order to find and measure the dents of the shank, I begin by focusing on the shank of the key image and detecting the edges of the image. Then I interpolate more precise sub-pixel coordinates for the points observed in the image as edges. At the end, I will convert the sub-pixel measures into physical measures.

3.5.1 Cutting the shank from the key image and detecting the edges and dents of the shank

I change my original color key image, to a binary image filled in with white pixels. Then I eliminate noise, holes and gaps that exist inside the key image. I explained the details of the algorithms in 3.2. Using the function filledBinary, I got a binary image of the key without any noise and holes inside the shank of the key. Figure 3.30 shows the original color image and the binary one.

a b
Figure 3.30 (a) Original color image. (b) Binary image after using filledBinary function

I illustrated the details about properties of the image in 3.3.1, adjusting the angle of the key image in 3.3.2 and adding zero pad arrays to the image in 3.3.3. I repeat all these algorithms and find the geometric centre, top and bottom of the key (Figure 3.17), and the division which is cut from $1/12^{th}$ of the distance before and after the geometric centre of the key (Figure 3.18). Using the technique I explained in 3.3.4, I find the joint of the key and the line that separates the key head and the shank (Figure 3.19).

Because the weights of the pixels are not the same at all parts the key image [27], [7]; the centroid (centre mass) is closer to the head of the key. I can use this fact to know that the head and the shank are located at which side of the cut line. I separate the shank from the body of key image and restore it in another image shown in figure 3.31.

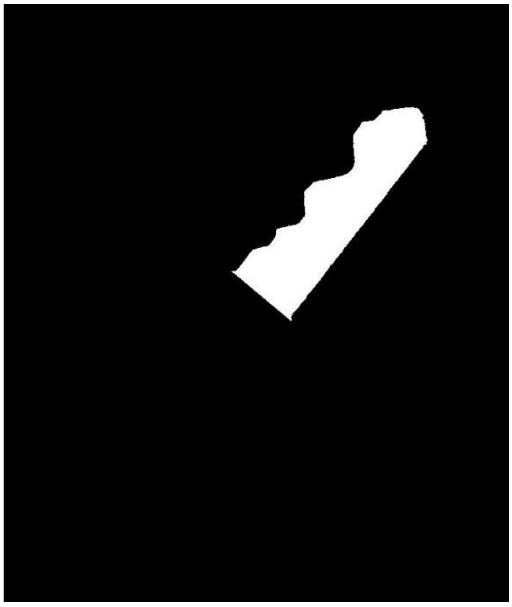


Figure 3.31 The shank cut from the key image

3.5.2 Detecting edges and dents of the shank

I need to have an image which only shows me the white pixels that are representative of the dents of the key. The goal is to obtain the connected contour of the image with one pixel width. Other white pixels that are inside the shank should be eliminated from the ideal contour.

Edge detectors are tools in image processing that identifies points in a digital image. These operators recognise the points in an image in which brightness changes sharply or has discontinuities [2]. The basic goal in image processing is to find places in an image where the intensity changes rapidly [14], [15], [2]. There are two general criteria:

1. The places where the first derivative of the intensity is greater in magnitude than a specified threshold.
2. The places where the second derivative of the intensity has a zero crossing.

In this project I use Canny Edge Detector which is one of the most powerful edge detectors [14], [18]. First, this detector employs Gaussian filter with a specified standard deviation to smooth the image and reduce the noise [4]. There are three techniques (Sobel, Prewitt, Roberts) shown in Figure 3.32 that can be used to compute the derivatives. Then the edge detector computes the local gradient, $[g_x^2 + g_y^2]^{\frac{1}{2}}$ and edge direction $\tan^{-1}(^{g_x}{}_{g_y})$ at each point of the image [14], [2]. The definition of an edge point is a point which locally has the maximum strength in the direction of the gradient [14], [4]. The edge points bring about the ridges in the gradient magnitude image. The algorithm tracks along the top of these ridges and change the value of all pixels that are not on the ridge top. Then based on two thresholds, *T1* and *T2* that *T1* < *T2*, ridge pixels are thresholded. Ridge pixels with values greater than T2 are said to be "strong" edge pixels and ridge pixels with values between *T1* and *T2* are said to be "weak" edge pixels. Finally, the algorithm performs edge linking by adding the weak pixels that are 8-connected to the strong pixels [4].

Canny Edge Detector results are superior to the results from other edge detectors. It detects the details of the image clearly and produces the cleanest edge map.

z_1	z_2	z_3
z_4	z_5	z_6
z_7	z_8	z_9

Image neighbourhood

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

$$g_x = z_7 + 2z_8 + z_9 - (z_1 + 2z_2 + z_3)$$

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

$$g_y = z_3 + 2z_6 + z_9 - (z_1 + 2z_4 + z_7)$$

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

$$g_x = z_7 + z_8 + z_9 - (z_1 + z_2 + z_3)$$

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

$$g_y = z_3 + z_6 + z_9 - (z_1 + z_4 + z_7)$$

-1	0
0	1

$$g_x = z_9 - z_5$$

0	-1
1	0

$$g_y = z_8 - z_6$$

a

b

c

d

Figure 3.32 Edge detector masks, (a) Image neighbourhood. (b) Sobel. (c) Prewitt. (d) Roberts.

Figure 3.33 shows the result after I apply canny edge detector on Figure 3.31. The width of the connected contour in the image is ideally one pixel.

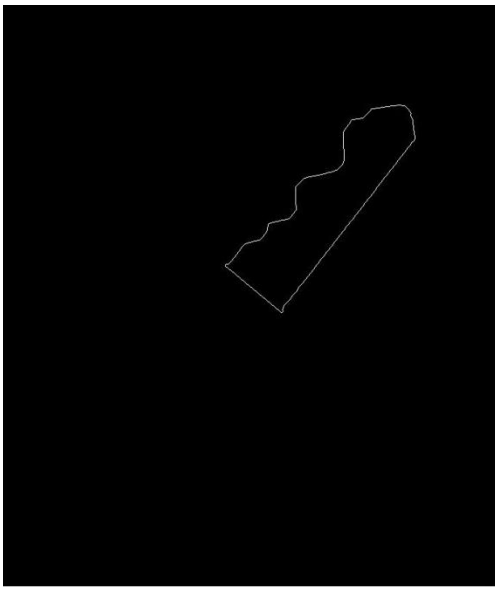
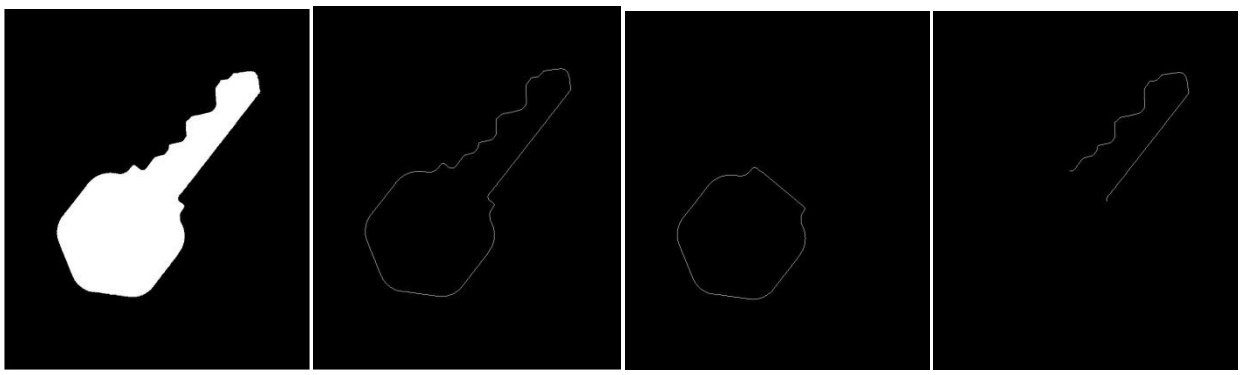


Figure 3.33 Connected contour of the shank using canny edge detection

In Figure 3.33 each pixel presents a point of the ridges that will be measured accurately for cutting a new key blank. In this image, some pixels are not part of the shank edges. The pixels of the line that cuts the border of head and shank of the key and connects two sides of the key shank are not actual elements of the edges that should be transferred to the system as the information for cutting device. This line appeared in Figure 3.33 because it was a part of the perimeter of the shank in Figure 3.31. In order to get rid of this line at the joint of the key and have an image with real dents of the key image, first I fill in the binary image of the key shown in Figure 3.30(b) with white pixels (Figure 3.34(a)) and using canny edge detector I get the connected contour of the whole key image (Figure 3.34(b)). Using cutHead function that I explained in 3.3 I have the head of the key image. I fill in its holes and obtain the connected contour of the head (Figure 3.34(c)). The difference between connected contours of the whole key image and head of the key image lead me to find the connected contour of the shank without the line that connects two sides of the key shank (Figure 3.34(d)).



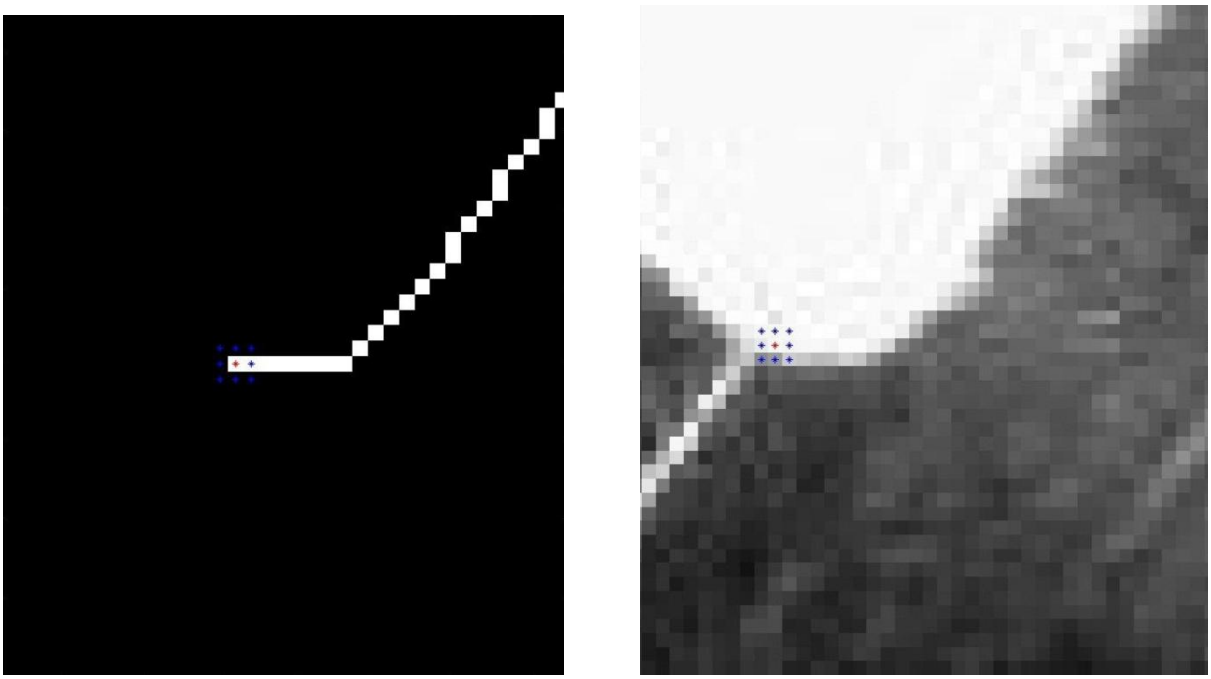
a b c d

Figure 3.34 Finding the contour of the shank without the line that separates head and shank

3.5.3 Sub-pixel Interpolation

In last part I found the way I can keep the pixels of the shank that present the edges and dents of the original key image. Each pixel in Figure 3.34(d) is representative of a position of cut but measuring the positions of cuts in pixels is too rough and is not accurate and appropriate enough for a cutting machine to know how to cut a new key blank. In this part the goal is to interpolate more precise coordinates for the points observed in the image as white pixels. The sub-pixel point will be the ones that will be converted to the suitable scale to know how to cut the key.

In this step I get an edge pixel of the shank and its coordinates from Figure 3.34(d) and find the coordinates of the eight pixels around the edge pixel. Now I got a 3 by 3 neighbourhood of the binary image that the middle element is the edge pixel. I retrieve the corresponding 3 by 3 neighbourhood from original grayscale key image with the same coordinates that I just found from binary image. In Figure 3.35(a) the middle star denotes the selected edge pixel and the stars around it show the 3 by 3 neighbourhood of pixels. In Figure 3.35(b) the retrieved 3 by 3 neighbourhood from the original grayscale key image is denoted. The star in the middle of the neighbourhood in Figure 3.35(b) shows the coordinates of selected edge pixel from Figure 3.35(a).



a b

Figure 3.35 (a) Edge pixel and its neighbourhood in binary contour of the shank. (b) Retrieved neighbourhood from the grayscale key image.

Now in the neighbourhood shown in grayscale image I need to find the coordinates of the point of the edge among these nine pixels as precisely as possible. I will use the Vandermonde method to interpolate multivariate real-valued functions [17].

I have nine pixels in a neghbourhood. Each pixel has its coordinates and intensity in gray scale measures. In another words, I have a polynomial which has nine terms and two variables so it is a two degree polynomial. Equation (3.14) shows my polynomial which passes through all of the considered points.

$$p\ x,y\ = c_0 + c_1x + c_2y + c_3xy + c_4x^2 + c_5y^2 + c_6x^2y + c_7xy^2 + c_8x^2y^2 \tag{3.14}$$

I will simply write down the problem in the form $I\ C = M$ that
M is the vector of values that keeps the intensities of the pixels in the 3 by 3 neighbourhood. C is the vector of coefficients and I is the Vandermonde matrix [17].

The Vandermonde matrix is an $n \times n$ matrix where the first row is the first point evaluated at each of the n monomials, the second row is the second point evaluated at each of the n monomials, and so on. Figure 3.36 shows the Vandermonde matrix I used to solve the Equation [17].

	1	x	y	xy	x^2	y^2	x^2y	xy^2	x^2y^2
$x=0\ y=0$	1	0	0	0	0	0	0	0	0
$x=1\ y=0$	1	1	0	0	1	0	0	0	0
$x=2\ y=0$	1	2	0	0	4	0	0	0	0
$x=0\ y=1$	1	0	1	0	0	1	0	0	0
$x=1\ y=1$	1	1	1	1	1	1	1	1	1
$x=2\ y=1$	1	2	1	2	4	1	4	2	4
$x=0\ y=2$	1	0	2	0	0	4	0	0	0
$x=1\ y=2$	1	1	2	2	1	4	2	4	4
$x=2\ y=2$	1	2	2	4	4	4	8	8	16

Figure 3.36 Vandermonde matrix for solving the polynomial equation

In the problem $I\ C = M, C$ is the vector of coefficients and is unknown so by dividing M by I, I can simply find the coefficients of the equation.

Now I got a polynomial that passes through all the nine points in the neighbourhood. The goal is to find a more precise edge in this neighbourhood so I have to obtain the coordinates of a point in the polynomial that gives me the darkest point in the grid because the darkest point is the real position of edge located in that 3 by 3 neighbourhood. A sample grid that covers nine points is shown in Figure 3.37. The darkest point of the grid has the minimum point value in the polynomial [17].

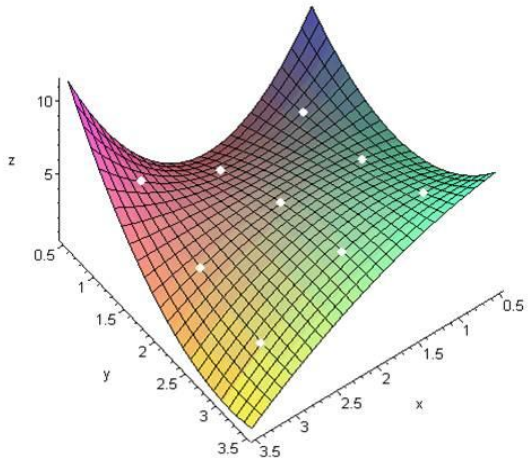


Figure 3.37 The interpolating polynomial of nine points[2]

[2] http://www.mathworks.com/help/toolbox/optim/ug/fmincon.html

In order to find the coordinates of the more precise edge in the grid, I will find the coordinates of the point that gives me the minimum result in the polynomial Equation (3.12). I used a function called fmincon[3] from MATLAB optimization toolbox to compute the polymonial function based on Sequential quadratic programming (SQP) and find the coordinates of the real edge [15]. I defined a set of lower and upper bounds (*lb* = [-0.5,-0.5] and *ub = [+2.5,+2.5]*) on the design variables so the solution is always in the range *lb ≤ x & y ≤ ub.*

After I got the coordinates of the minimum value in the polynomial, I determined the coordinates of the real edge point in the binary image of the shank (3.31(d)). The binary image of the shank denoted with the coordinates of more precise edge points are shown in Figure 3.38. The blue line is the major axis line of the key image.

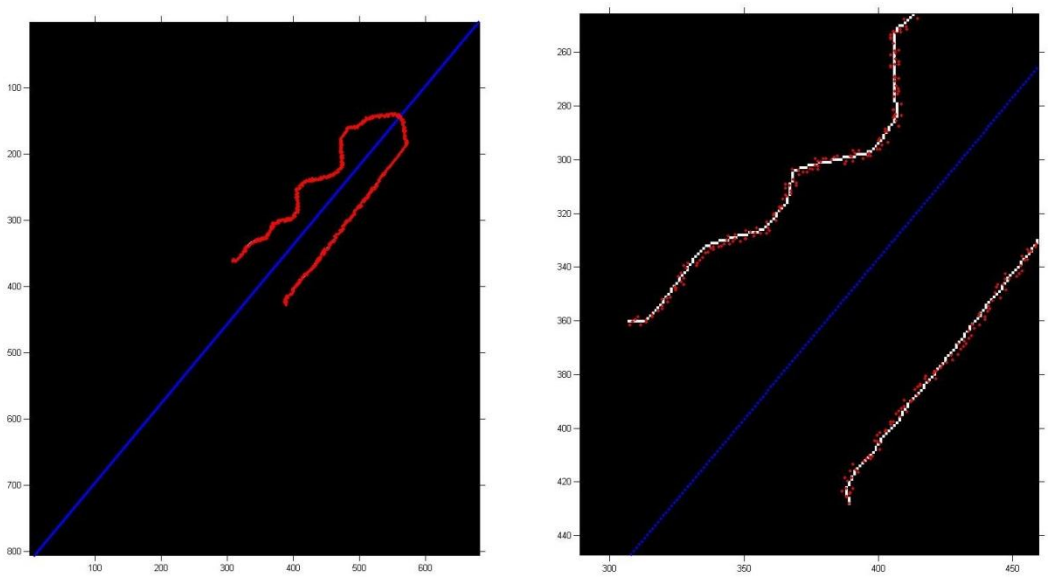


Figure 3.38 sub-pixel points representing more precise edge points than edge pixels

3.5.4 3.5.4 Determining the positions of cuts and converting them to physical measures

Sub-pixel points are the real dents that the blade of the cutting machine will cut a new key blank. In order to find and store the locations of each sub-pixel points I compute the shortest distance between each edge point and the major axis line of the key image shown in Figure 3.37 [13]. In order to be able to compute the shortest distance of two sided keys' edges, I do not measure the distance between each edge point and total width of the key.

The distance between the point (x_1, y_1) and the line $y = mx + b$ is given by the formula shown in Equation (3.15) [13].

$$D = \frac{y_1 - mx_1 - b}{\sqrt{m^2 + 1}} \tag{3.15}$$

[3] http://www.mathworks.com/help/toolbox/optim/ug/fmincon.html

The calculated distance in my image is in pixels. In 3.4.4 I explained the way I can find the correct key blank and determine the pixel length in physical measures (millimetres) for the corresponding original key image. Using that technique I can convert the distance between the edge points and the major axis line to millimetres and store the distances in an array. The array keeps all the coordinates of the edge points and their distance to the major axis line in millimetre.

3.6 Another example of the software procedures on a sample key image

In this section I illustrate the algorithms and their performance on a sample key and show the results step by step. Figure 3.39 shows the grayscale key image that we want to work on it.

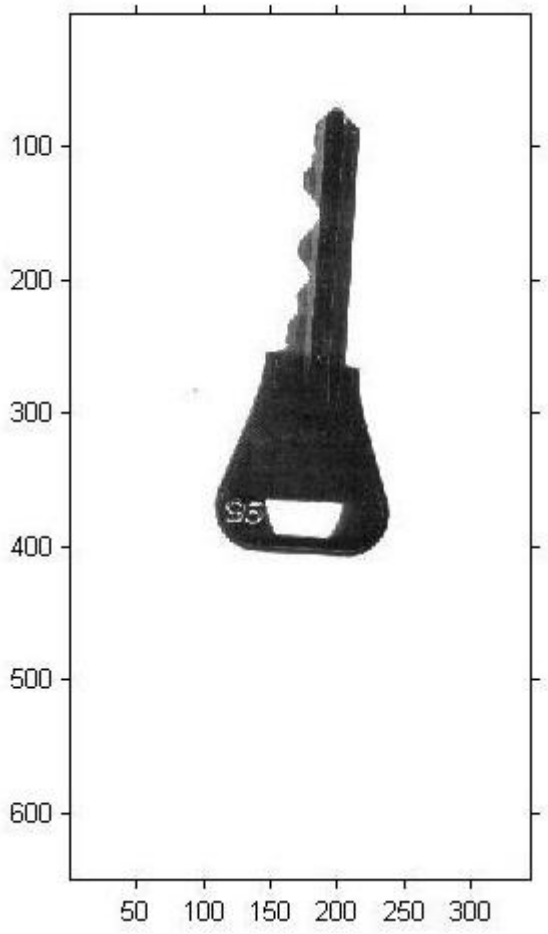


Figure 3.39 Original grayscale key (2nd key)

The first function is filledBinary (explained in 3.2) that has two steps. The first step is converting the grayscale image into a binary image. Figure 3.40 shows the inverted binary key image.

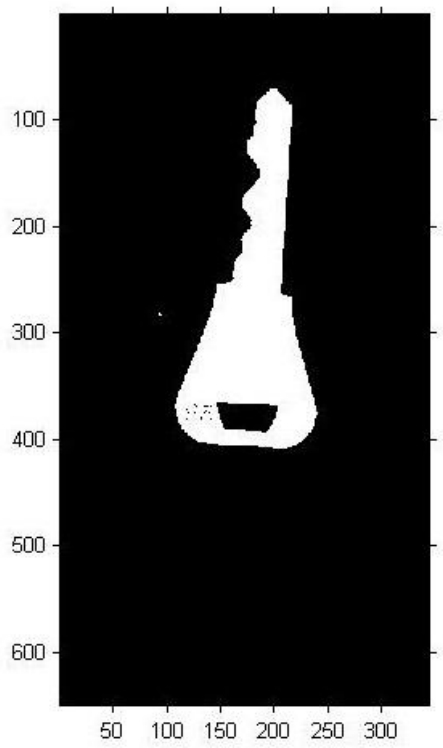


Figure 3.40 Inverted binary key image (2nd key)

Then I eliminate gaps, holes and noise of the binary image shown in Figure 3.41.

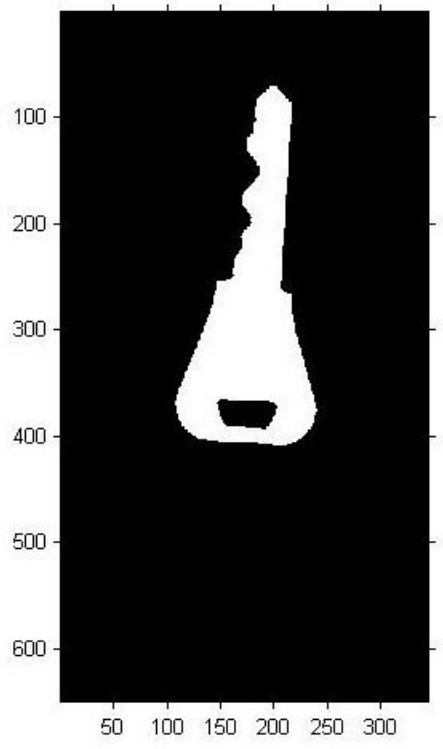


Figure 3.41 Binary key image without gaps, holes and noise (2nd key)

According to section 3.3, the image is ready to cut the head from the body of the key image. The first step of the function cutHead is to obtain the division of the key that is the neighbourhood of 1/6th around the geometric center shown in Figure 3.42.

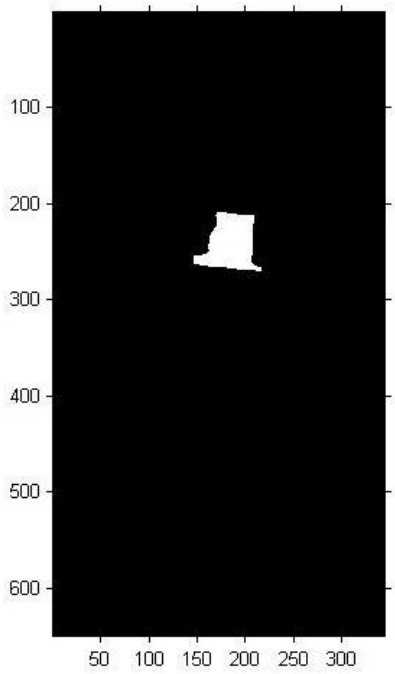


Figure 3.42 Division of key image (2nd key)

Next I start counting number of white pixels in each row of the division and save the numbers in elements of an array shown in Table 3.5. The sharpest difference in each two successive elements represent s the position of joint and the place I can cut the head from the shank of the key image.

Table 3.5 Number of pixels in each row (2nd key)

Column 1-10	Column 11-20	Column 21-30	Column 31-40	Column 41-50	Column 51-60	Column 61-64
63	49	40	39	38	33	35
63	46	39	39	37	33	35
62	43	40	40	36	33	34
63	41	40	39	36	33	34
59	40	40	39	36	34	
57	40	41	39	35	33	
55	40	39	40	34	34	
53	40	39	39	33	34	
53	40	40	39	33	35	
53	40	39	38	33	36	

The 4th and 5th elements of the array (in bold) show the sharpest difference and lead me to find the coordinates of the joint. In Figure 3.43 I show the line that pass the joint of the key and separates head and shank of the key image.

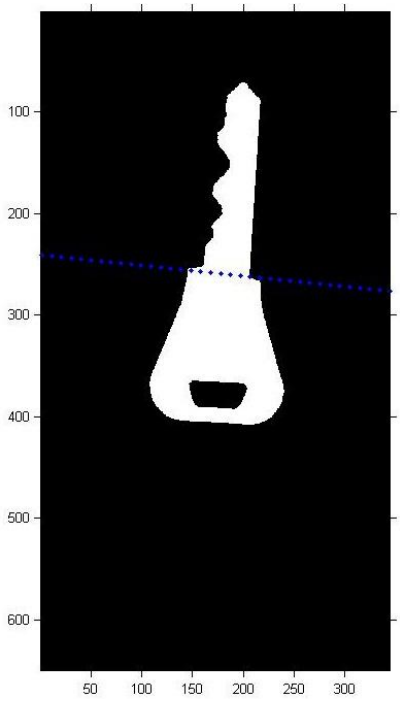


Figure 3.43 The line that separates the head and shank of the key image (2nd key)

In last part of the function cutHead I restore the head of the key image illustrated in Figure 3.44.

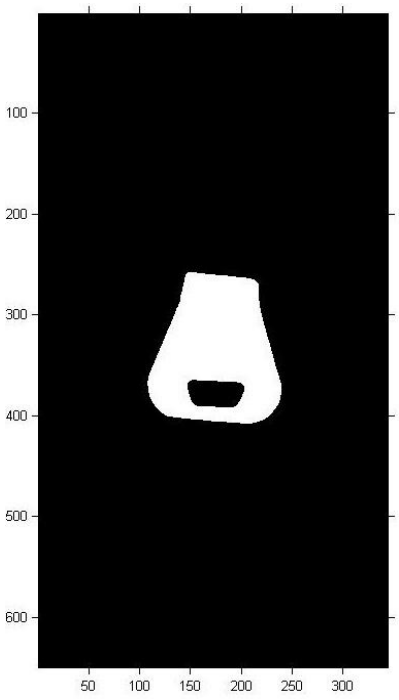


Figure 3.44 The head cut from the body of key image (2nd key)

According to the function findKey (section 3.4), the image of the head in Figure 3.44 is surrounded by large empty areas and using the function boundingbox, I eliminate the black areas around the main object (Figure 3.45).

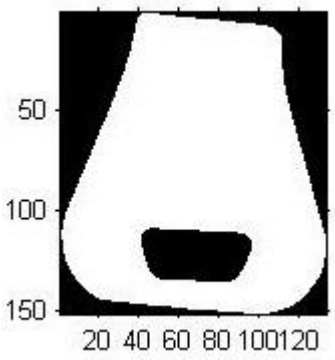


Figure 3.45 Head of the key in bounding box (2nd key)

In Figure 3.46 I show the filled original head images and using the Equations (3.12) and (3.13) I compute the distance of moment invariants between the filled and unfilled head images shown in Figures 3.45 and 3.46.

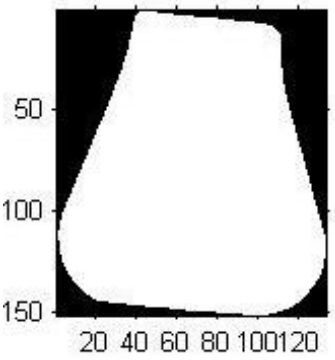


Figure 3.46 Filled Original key head (2nd key)

The result shows the distance between moment invariants of filled and unfilled original key head images is 2.1564 which is a reasonable difference between their moment invariants. This distinction between these two images helps me use the filled original key head image (Figure 3.46) as another feature for comparison between filled head images.

In Figures 3.47 – 3.55, I showed the keys in each set and in Tables 3.6 – 3.14 I computed the result of distances between moment invariants of original head image and the key head images in each set.

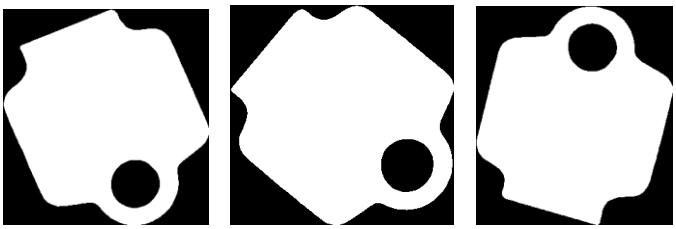


Figure 3.47 Three head images of key type number 77

Table 3.6 Distances of key type number 77

	Unfilled head images	Filled head images	Average
Var(4.7, 4.9(a))	4.9506	4.4813	4.7372
Var(4.7, 4.9(b))	4.6538	4.8143	
Var(4.7, 4.9(c))	4.9439	4.5797	

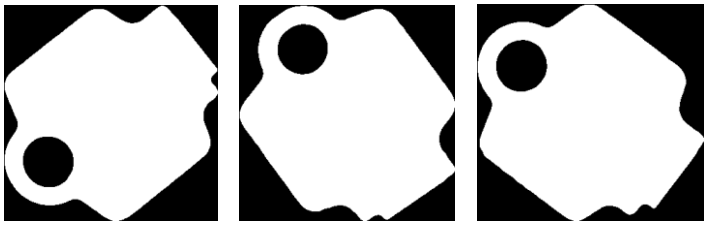


Figure 3.48 Three head images of key type number 77 (second joint)

Table 3.7 Distances of key type number 77 (second joint)

	Unfilled head images	Filled head images	Average
Var(4.7, 4.10(a))	2.4622	6.8755	4.6655
Var(4.7, 4.10(b))	2.7309	6.9442	
Var(4.7, 4.10(c))	2.6066	6.3739	

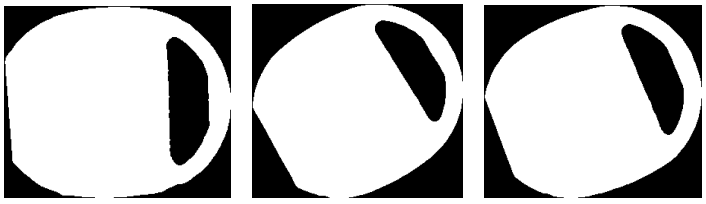


Figure 3.49 Three head images of key type number 8

Table 3.8 Distances of key type number 8

	Unfilled head images	Filled head images	Average
Var(4.7, 4.11(a))	2.3892	8.5499	7.1800
Var(4.7, 4.11(b))	2.5146	14.2171	
Var(4.7, 4.11(c))	3.0144	12.3949	

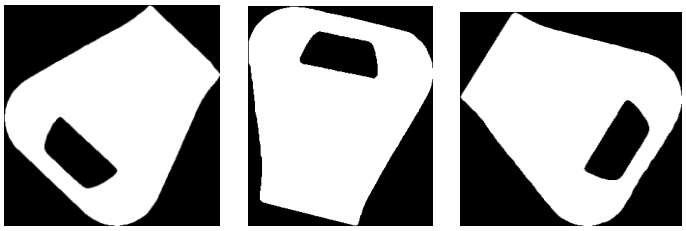


Figure 3.50 Three head images of key type number 96

Table 3.9 Distances of key type number 96

	Unfilled head images	Filled head images	Average
Var(4.7, 4.12(a))	0.4673	0.4241	0.4750
Var(4.7, 4.12(b))	0.1895	0.4062	
Var(4.7, 4.12(c))	0.6456	0.7174	

Figure 3.51 Three head images of key type number 69

Table 3.10 Distances of key type number 69

	Unfilled head images	Filled head images	Average
Var(4.7, 4.13(a))	1.5527	1.8618	1.6081
Var(4.7, 4.13(b))	1.5815	1.5506	
Var(4.7, 4.13(c))	1.6248	1.4774	

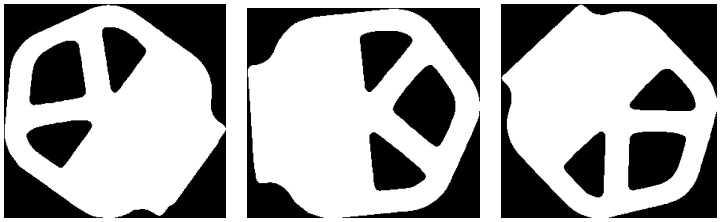


Figure 3.52 Three head images of key type number 66

Table 3.11 Distances of key type number 66

	Unfilled head images	Filled head images	Average
Var(4.7, 4.14(a))	2.7897	9.2944	6.7555
Var(4.7, 4.14(b))	2.4611	10.0186	
Var(4.7, 4.14(c))	3.1982	12.7712	

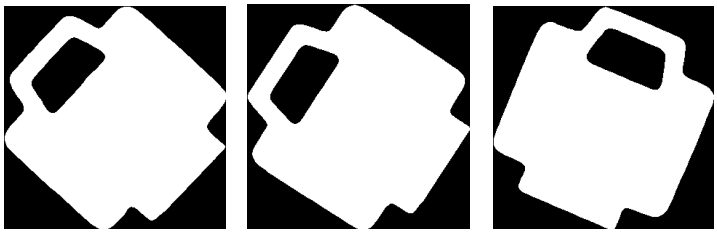


Figure 3.53 Three head images of key type number 67

Table 3.12 Distances of key type number 67

	Unfilled head images	Filled head images	Average
Var(4.7, 4.15(a))	3.7834	9.9060	6.9732
Var(4.7, 4.15(b))	3.6274	10.2183	
Var(4.7, 4.15(c))	3.3000	11.0042	

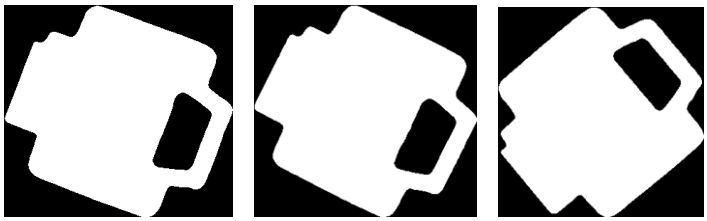


Figure 3.54 Three head images of key type number 67 (second joint)

Table 3.13 Distances of key type number 67 (second joint)

	Unfilled head images	Filled head images	Average
Var(4.7, 4.16(a))	2.5001	7.8831	5.0631
Var(4.7, 4.16(b))	2.4524	7.3097	
Var(4.7, 4.16(c))	2.4943	7.7394	

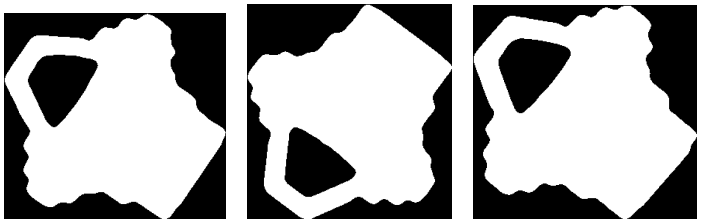


Figure 3.55 Three head images of key type number 68

Table 3.14 Distances of key type number 68

	Unfilled head images	Filled head images	Average
Var(4.7, 4.17(a))	1.4208	4.3395	2.7449
Var(4.7, 4.17(b))	1.4575	4.1395	
Var(4.7, 4.17(c))	1.3828	3.7297	

Table 3.9 has the lowest average of distances among moment invariants so it introduces the information about the key type for further uses. Besides, Figure 3.56 is popped up on the screen to show us the key blank that is found by the function findKey.

Figure 3.56 Obtained key blank from database using the function findKey(2nd key)

After finding the correct key match in the database, I have the information about the physical length, key number and pixel size of the original key. In part 3.5.1 I explained the way I cut the shank from the key image. Figure 3.57 shows the shank of the original key image.

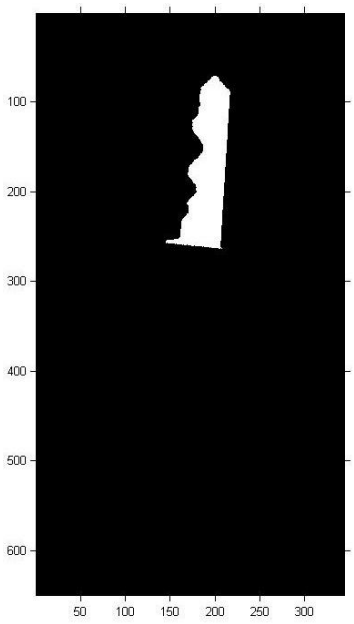


Figure 3.57 Shank cut from the original key image (2nd key)

Using canny edge detection and the method that I explained in 3.5.2 I obtain Figure 3.58 that is the connected contour of the shank which has the pixels showing the position of cuts and dents. I showed the major axis line in blue.

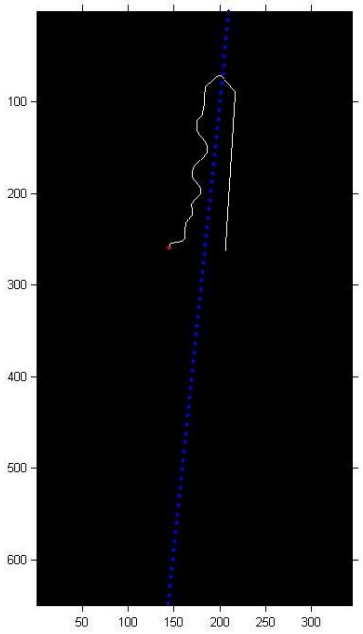


Figure 3.58 Connected contour of the shank (2nd key)

Sub-section 3.5.3 talks about the method I use to measure the position of cuts and indentations in physical measures not in pixels. Vandermone sub-pixel interpolation is the method I use to find more precise edges in the connected contour of the shank. Figure 3.59 shows the connected contour denoted with sub-pixel points representing more precise edge point s than the edge pixels.

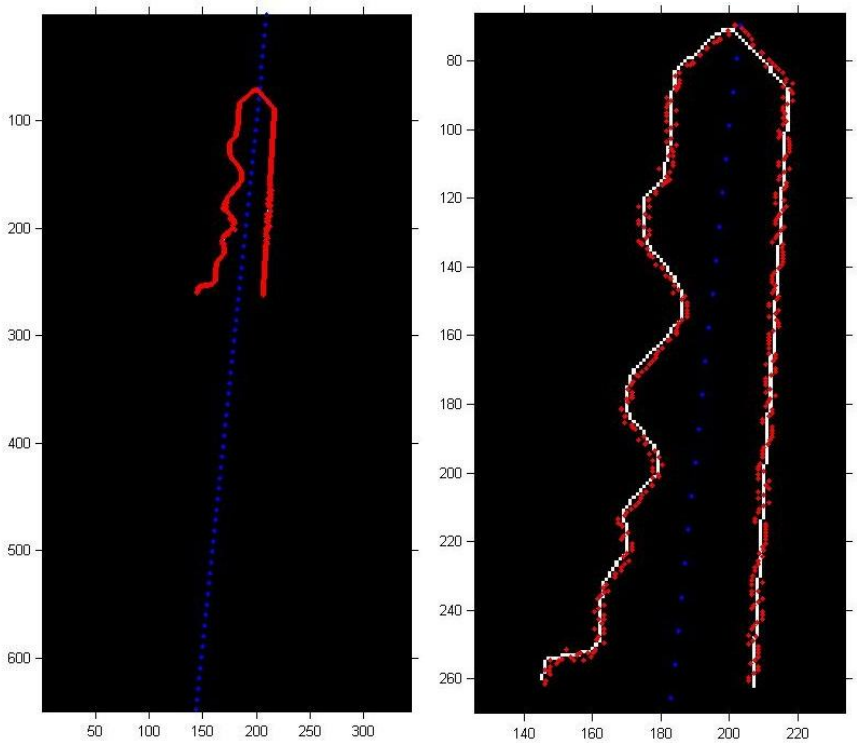


Figure 3.59 Sub-pixel points over the shank (2nd key)

According to sub-section 3.5.4 and Equation (3.15) in Table 4.32 I illustrate the real position of cuts in Figure 3.60.

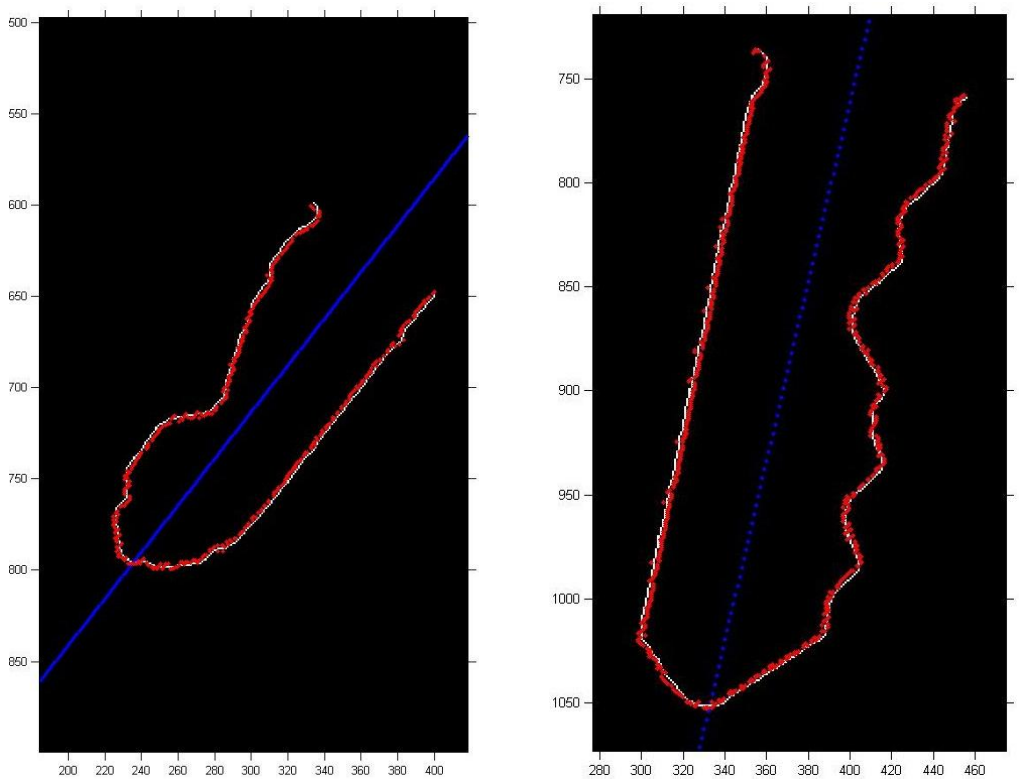


Figure 3.60 Edge points superimposed over the edge pixels

The coordinates of sub-pixel points and their distance to the major axis line (in pixels) are saved in an array. The information in the array alone with the key number will be transferred to a cutting machine to pick the correct key blank and cut the dents and edges on it.

4 Experimental Results

4.1 Introduction

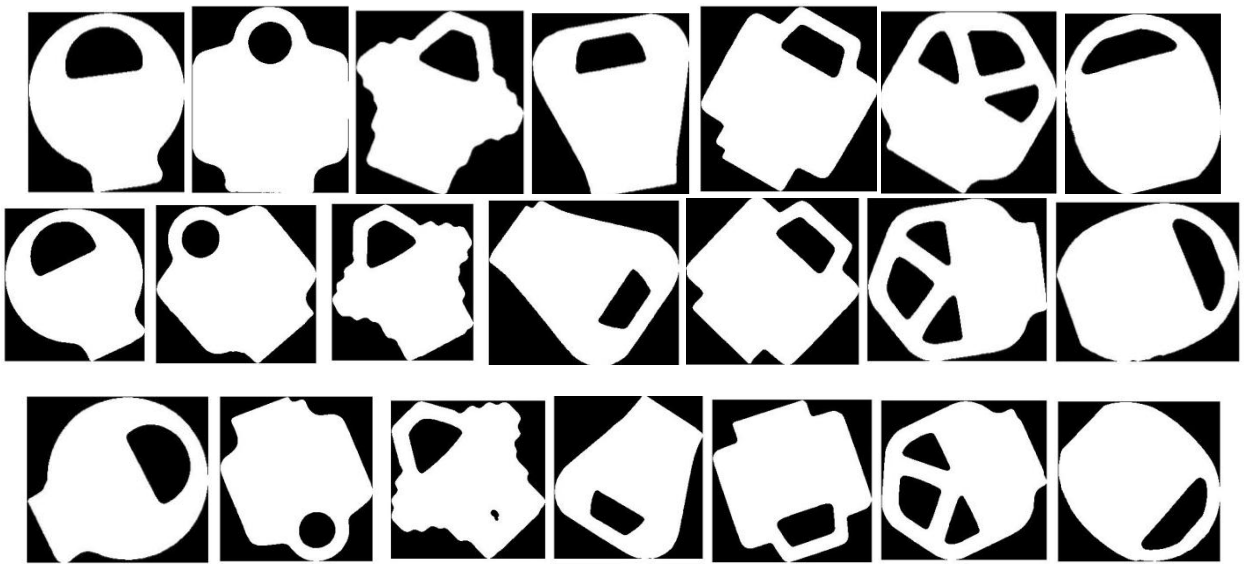
In this chapter I will present and analyse the experimental results I obtained with my key measuring system. I will test the two major goals of the project, which are finding the correct match for the original key image and measuring the depth of the cuts and edges of the original key shank. In the first set of tests in Section 4.2, I will study the performance of the procedures I used to find the correct key blank in the database and will discuss about the errors and how I deal with them. The second set of tests in Section 4.3 is designed to study the measures of the key dents. The relative errors in finding the position of cuts will be computed and I will explain the factors that can affect them.

4.2 Identify the correct key blank

In Chapter 3, I explained that I compare a key's head to six images in the database, three images of the head with its holes intact and three with the holes filled up. My first experiment is meant to demonstrate the necessity of doing this number of comparisons. I will study 21 different images of my test keys, presented in Figure 4.1. Their heads were extracted using the "cutHead" function of section 3.3 and shown in Figure 4.2.

a1 a2 a3 a4 a5 a6 a7
b1 b2 b3 b4 b5 b6 b7
c1 c2 c3 c4 c5 c6 c7

Figure 4.1 Three sets of seven types of key



A1 A2 A3 A4 A5 A6 A7
B1 B2 B3 B4 B5 B6 B7
C1 C2 C3 C4 C5 C6 C7

Figure 4.2 21 Binary key heads

Next, I computed the seven moment invariants of each binary key head image in Figure 4.2 using Equation (3.12). The results are presented in Table 4.1.

Table 4.1 Moment invariants of binary head images

	$I1$	$I2$	$I3$	$I4$	$I5$	$I6$	$I7$
A1	0.7059	3.6129	3.5105	3.8507	7.5449	5.7185	8.1389
B1	0.7069	3.6779	3.4909	3.8458	-7.5185	7.7555	8.3607
C1	0.7066	3.7279	3.5681	3.8376	-7.5543	5.7403	8.1454
A2	0.7674	4.2222	6.2130	4.6118	10.0261	6.7238	11.0535
B2	0.7665	4.0093	5.8692	4.6024	-9.9674	6.6083	-10.0123
C2	0.7665	3.5745	4.9163	4.6991	-9.5664	6.4972	9.8165
A3	0.7199	3.8415	3.7438	4.1048	-8.0292	6.0261	-9.6492
B3	0.7188	3.8609	3.6606	4.1131	-8.0000	6.0455	-9.6104
C3	0.7163	3.8394	3.6788	4.1285	-8.0325	6.0512	-9.3779
A4	0.7137	2.9742	3.0025	4.2865	-7.9364	5.7743	-8.7334
B4	0.7127	2.7319	2.9537	4.5874	-8.3608	5.9534	-9.2992
C4	0.7169	2.9923	2.9882	4.4405	-8.1554	5.9367	-9.4909
A5	0.7509	4.5142	4.4405	4.3482	-8.7492	6.6330	9.5033
B5	0.7503	4.7816	5.5811	4.3293	-9.3759	-6.7221	9.5165
C5	0.7481	4.6421	5.6490	4.2922	-9.3194	-6.6159	9.5822
A6	0.6871	3.8747	5.0117	3.6703	8.0113	5.6092	9.8860
B6	0.6877	3.6804	5.2733	3.6756	8.1992	5.5164	8.4971
C6	0.6879	3.6490	5.4409	3.6734	8.2679	5.4980	8.6313
A7	0.7525	3.7572	5.4073	4.1865	8.9840	6.0723	10.2819
B7	0.7486	3.7254	5.4913	4.1071	8.9561	5.9709	9.2501
C7	0.7501	3.7428	5.8372	4.1172	9.8402	5.9896	-9.1016

4.2.1 Sample Keys

When comparing the moment invariants, I would ideally obtain the minimum distances between keys of the same type. To check this in practice, I will begin by computing the distance of moment invariants between some of the head images using Equation (3.13). I have selected three sample key heads to explore this question: A5, A2 and A7. The distance results are presented in Table 4.2.

Table 4.2 Distances between three test keys and the other head images, using unfilled heads

	A5	A2	A7
A1	2.4648	4.8909	3.2439
B1	2.3336	4.7721	3.1203
C1	2.3896	4.8349	3.1978
A2	2.7090	0	1.7755
B2	2.0260	1.1245	1.3331
C2	1.4203	1.9752	1.1286
A3	1.3807	3.5983	2.0239
B3	1.4175	3.6802	2.1185
C3	1.3974	3.7469	2.1730
A4	2.5368	4.7566	3.1610
B4	2.4698	4.3923	2.9329
C4	2.2962	4.3017	2.7988
A5	0	2.7090	1.5842
B5	1.3318	1.8914	1.5039
C5	1.3459	1.8090	1.3229
A6	1.7141	3.0202	1.3261
B6	2.0992	3.6592	2.0975
C6	2.1187	3.5032	1.9614
A7	1.5842	1.7755	0
B7	1.5263	2.4468	1.0440
C7	2.0892	2.2360	1.5241

The results with head A5 does show the minimum distance when compared to B5 and C5, the two other keys of the same type. However, some of the other distances are very close to that minimum value as well; model 3 is a good example of this problem. This creates the potential for confusion in the system. The test with A2 further illustrates this problem. While B2 does give the minimal distance, three other keys, B5, C5 and A7, incorrectly show a lower distance than C2. The test with A7 gives even worse results: after the correct match with B7, five heads show a lower distance than C7. This demonstrates that the distance between these pairs of key heads is not sufficient to pinpoint the correct match. A system that used C2 and C7 as its database head, for example, would not correctly recognise keys A2 and A7.

As I explained in Chapter 3, I deal with this problem by also computing the moment invariants and the distance of filled versions of the key heads. To illustrate the impact of this change, the same set of comparisons of Table 4.2 was repeated using filled heads, and the results are shown in Table 4.3. We can see from this table that there are still errors present – but they are different from those in Table 4.2.

Previously, A2 was misclassified to B5, C5 and A7 before C2. In these new results, those three keys give a much higher distance, but C3 gives a new error. Likewise, A7 had falsely low distances with B2, C2, B5, C5 and A6 in Table 4.2. Now, four of these five have higher distance; only B5 remains an error, along with B6 And while A5 had the minimum distances when compared to keys B5 and C5, in Table 4.3 four

new errors come up: C2, B6, C6 and C7 have lower distances than B5 and C5. However, the three keys of model 3, which were nearly matches in Table 4.2, show a much higher distance now.

Table 4.3 Distances between three test keys and the other head images, using filled heads

	A5	**A2**	**A7**
A1	5.9733	3.9479	9.2432
B1	5.7708	3.6977	9.0026
C1	5.6605	3.7018	8.9675
A2	3.0470	0	5.4493
B2	2.8008	1.4078	6.1385
C2	1.4494	1.9058	4.7843
A3	3.7288	2.4278	7.2117
B3	3.8065	2.1936	7.1373
C3	3.5142	1.5503	6.5974
A4	7.1180	5.1856	10.3886
B4	7.8020	5.6390	10.9205
C4	6.8113	4.6303	9.8869
A5	0	3.0470	4.0654
B5	2.4089	4.9842	2.7318
C5	2.4560	5.1465	3.0383
A6	3.8461	6.2251	3.4573
B6	1.5740	4.1904	2.9139
C6	1.4408	4.1194	3.0365
A7	4.0654	5.4493	0
B7	4.0183	6.6276	3.0034
C7	2.0344	4.5969	2.5890

Clearly, neither the comparison between unfilled heads nor filled heads is sufficient by itself to pair the heads without errors. However, as we mentioned in Section 3.4, the errors are mostly different between the two comparisons. Table 4.4 shows the results of averaging out the distance values of tables 4.2 and 4.3. As can be seen, the low values of the correct matches in each of these tables average out to a similarly low value, while the erroneous matches that had a low value in one table and a high value in the other average out to a higher distance than the correct matches. Indeed, no erroneous matches remain in Table 4.4.

Of particular interest is B5, which was an erroneous match compared to A7 in both tests. Using unfilled heads, it showed a distance of 1.5039 compared to A7, lower than 1.5241 for C7 but higher than 1.0440 for B7. Using filled heads, it had a distance of 2.7318, this time lower than B7 at 3.0034 but higher than C7 at 2.5890. That is to say, B5 was always an erroneous low-distance match, but compared to a different one of the two correct keys. When averaging out, B5 gets a distance of 2.1178 compared to A7, higher than both B7 at 2.0237 and C7 at 2.0565.

Table 4.4 Average of filled and unfilled distances between three test keys and the other head images

	A5	A2	A7
A1	4.2190	4.4194	6.2435
B1	4.0522	4.2349	6.0614
C1	4.02505	4.2683	6.0826
A2	2.8780	0	3.6124
B2	2.4134	1.2661	3.7358
C2	1.4348	1.9405	2.9564
A3	2.5547	3.0130	4.6178
B3	2.6120	2.9369	4.6279
C3	2.4558	2.6486	4.3852
A4	4.8274	4.9711	6.7748
B4	5.1359	5.0156	6.9267
C4	4.5537	4.4660	6.3428
A5	0	2.878	2.8248
B5	1.8703	3.4378	2.1178
C5	1.9009	3.4777	2.1806
A6	2.7801	4.6226	2.3917
B6	1.8366	3.9248	2.5057
C6	1.7797	3.8113	2.4989
A7	2.8248	3.6124	0
B7	2.7723	4.5372	2.0237
C7	2.0618	3.4164	2.0565

Computing the average of a filled and unfilled version of the same image yields the desire results for A2 and A7, but in some cases the between A5 and the two other keys of its model are not the minimum. Indeed, C2, B6 and C6 have the lowest averages with A5. Moreover, in some correctly-classified cases I find that the average distance of a wrong key is not sharply higher than that of the same key type. In fact, the difference can be as low as 0.06 (or 3% of the distance value), in the case of the A7-C7 match compared to A7-B5. Clearly the potential for errors still exists.

As I explained in 3.4.4, in order to increase the chance of finding the correct match, I compute the distances between the original head image and three different images of a key head of each type stored in the database and shown in Figure 3.25. As we can see in Table 4.5, averaging the distance of three images gives results that are much more robust and resilient to noise. In all three sample cases, the test keys show the lowest average distance with keys of the same type. In addition, the difference between the distance of a correct and incorrect match becomes more significant. In the specific case of our previous example, the difference between A7-tpye 7 and A7-type 5 is five times higher, at 0.3 or 15% of the distance value.

Table 4.5 Average of distances using the database of key images

	A5	A2	A7
Key type No. 1	4.0987	4.3075	6.1291
Key type No. 2	2.2420	2.2364	3.4348
Key type No. 3	2.5408	2.8661	4.5436
Key type No. 4	4.8390	4.8175	6.6814
Key type No. 5	1.8856	3.2645	2.3744
Key type No. 6	2.1321	4.1195	2.4654
Key type No. 7	2.5529	3.8553	2.0401

4.2.2 Results on 21 Keys

The experiment described in the previous subsection was run with all 21 keys, not just the three presented previously. The results, shown in Table 4.6, confirm that the comparison becomes more accurate when we average more sample images together.

Table 4.6 Errors occurred in each step with all 21 sample keys

	Errors compared to one unfilled key	Errors compared to one filled key	Errors when averaging the filled and unfilled key	Errors when averaging two keys	Errors when averaging three keys
A1	0	0	0	0	0
B1	0	0	0	0	0
C1	0	0	0	0	0
A2	3	1	0	0	0
B2	1	3	0	0	0
C2	6	1	1	0	0
A3	0	0	0	0	0
B3	0	0	0	0	0
C3	0	0	0	0	0
A4	0	0	0	0	0
B4	0	0	0	0	0
C4	0	0	0	0	0
A5	0	4	3	1	0
B5	1	4	4	0	0
C5	4	4	4	0	0
A6	3	0	0	0	0
B6	1	4	4	0	0
C6	1	4	4	0	0
A7	5	2	0	0	0
B7	0	5	4	1	0
C7	4	6	4	4	2
System Average	**29 errors 10 keys failed**	**38 errors 11 keys failed**	**28 errors 8 keys failed**	**6 errors 3 keys failed**	**2 errors 1 key failed**

The results of Table 4.6 are obtained by comparing each of the 21 keys of Figure 4.2 with the other 20 keys. This gives a total of 420 comparisons. When using one unfilled head image only, 10 keys are recognized incorrectly. Moreover, there are 29 distances that are wrongly higher than the correct key distance, which is to say that 6.9% of distances are erroneous. Using only one filled head image gives even worse results: 11 keys are misclassified, and 38 of the distances, or 9%, are erroneous.

Averaging the filled and unfilled key head images gives a small improvement in the results. 28 of the distance averages (6.7%) are erroneous and 8 out of 21 keys are not identified correctly. Although the improvement is small, it confirms that averaging both images is more accurate than using either image separately. Next, I average the distances of two different key heads for each key. With three different heads per key model, there are three different pairs of key heads or 19comparisons for each of the 21 keys and 378 average distances in total. As we can see in Table 4.6, the number of errors drops sharply in this case: only 3 of the 21 keys are misclassified, and only 6 of the distances, or 1.5%, are erroneous.

Finally, I tested by taking the average of the distances between the original key and the three keys in each set (two keys in the set of the same key type). The results are now nearly perfect: only one key is misclassified. In this experiment I computed 7 average distances for each of the 21 keys, and I find that only 2 of these averages, or 1.4%, are erroneous.

This test shows why I need to have three keys of the same key type in each key set. Only one misclassified key remains after averaging three images, and the proportion of erroneous distance averages has remained roughly the same as with two averages. This seems to be the best result we can get by averaging images of the head. But errors do remain, and looking at Table 4.6 shows that key C7 is the single most difficult key to classify using comparisons.

I tested different key head images of this type and found out there is only one specific key type that might be selected incorrectly by the system as the match for original keys from key type number 7 and that is key type number 5. I reviewed the keys and tried to define another factor for comparison in addition to the moment invariants. Looking at the images of Figure 4.2, one clear difference between those two models and other key models as well, is the size of the main hole in the key's head. I can take that parameter into account easily enough, by computing the ratio of white and black pixels inside the key head. As we can see in Table 4.7, the difference between the ratios of these two types is a good enough to identify the correct key blank among key type number 5 and number 7. This factor is a kind of supplement only for identifying key type number 7. When the system classifies a key as type number 5 using the distances, this second test checks specifically the ratio of number of pixels in the hole to number of pixels of the head. If the ratio is closer to the average of the ratios in key type number 5, this key type is the correct key blank; otherwise, key type number 7 is the match for the original key image. As we can see in Table 4.7, there is a clear difference in the hole-to-head ratio of those two key models. The average ratio for model 7 is 0.1254, while for model 5 it is 0.0983. This feature is not useful for all other classifications and cannot be used as a reliable factor among all key types to identify the correct key blank. There are some overlaps among different ratios of other key types. Indeed, while the ratios between key type number 5 and 7 have a reasonable difference, the ratio of model 7 is close to that of model 3 and the ratio of model 5 is close to model 4. This method would thus not be able to tell apart these keys.

Table 4.7 Ratio of number of pixels in head hole to number of head's white pixels

	No. of hole black pixels	No. of head white pixels	Ratio (hole pix/head pix)
A1	7395	42648	0.1734
B1	7518	43062	0.1746
C1	7438	43066	0.1727
A2	4381	66414	0.0660
B2	4409	66375	0.0664
C2	4312	68409	0.0630
A3	5652	48107	0.1175
B3	5687	48605	0.1170
C3	5872	48202	0.1218
A4	5632	53738	0.1048
B4	5150	56677	0.0909
C4	5137	54789	0.0938
A5	5623	58748	0.0957
B5	5583	57094	0.0978
C5	5775	57002	0.1013
A6	11380	49424	0.2303
B6	11304	49701	0.2274
C6	11314	49823	0.2271
A7	5876	49528	0.1186
B7	6200	48258	0.1285
C7	6252	48397	0.1292

When I studied the first two tests of Table 4.6 further, I found that out of the 67 erroneous distances in the set of 840 distances of filled and unfilled key heads, almost half (32 errors) are related to key type number 2 and 5. Looking at Figure 4.2, we can see that these are the two key models for which there are two possible joints that my software can find. I discussed this problem in Section 3.4.4, and I explained that in these cases, the moment invariants will be different depending on whether the software cut the head from the first or second joint of the key. The resulting high error rate that I've shown here is why I decided it was necessary to have two sets of three images in the database for these keys, one set with each possible joint division.

4.2.3 Results with Database

For a final test, I decided to study the classification results using the 21 sample keys of Figure 4.2 and the 27 keys from seven types in my database[4] shown in Figure 3.24 I will first demonstrate the impact of having three key images of each key type in the database by comparing with different scenarios, and I will check that the results are comparable to those I obtained in the previous subsection. Finally I will compare the achieved results and check to see if the software can meet the requirements.

I began by computing the distance in moments invariants the 21 keys of Figure 4.2 compared with one unfilled key head image of each type in the database (plus a duplicate for types 2 and 5). Of the 189 distances presented in Table 4.8, 13 are erroneous. That gives an erroneous rate of 6.8%, not far off the

[4] Two extra sets are related to the key types which have 2 joints on their shoulders.

6.9% obtained in the same test in Table 4.8. However, these errors all occur with only two keys, B6 and C6.

Table 4.8 Distances of the moments between 21 sample keys and the database (unfilled key heads)

	D1	D2	D22	D3	D4	D5	D55	D6	D7
A1	0.0286	5.3258	3.3187	1.6186	1.4107	3.8171	2.5497	1.7471	2.4466
B1	0.2298	5.2142	3.2204	1.4173	1.3447	3.6738	2.4174	1.7289	2.3625
C1	0.1127	5.2693	3.2865	1.5957	1.4814	3.7452	2.4732	1.6877	2.3884
A2	4.8698	0.4824	1.8537	3.6807	4.5511	1.3812	2.6506	3.6706	2.5259
B2	4.0359	1.4351	1.1045	3.0781	3.7786	1.3049	1.9872	2.8512	1.7050
C2	3.1768	2.3866	0.1710	2.1528	2.6892	1.5442	1.4238	2.3646	1.2385
A3	1.6531	4.0683	2.1608	0.1060	1.4321	2.4848	1.4508	1.8157	1.6429
B3	1.6084	4.1514	2.2310	0.0867	1.3838	2.5563	1.4887	1.8659	1.7217
C3	1.4135	4.2135	2.2405	0.2517	1.2607	2.6306	1.4764	1.7387	1.6704
A4	1.1747	5.2052	3.0234	1.4024	0.3631	3.8106	2.6248	2.3099	2.5897
B4	1.9234	4.8388	2.6387	1.4338	0.5573	3.5729	2.5491	2.6527	2.5759
C4	1.8031	4.7609	2.6190	1.1084	0.6286	3.3978	2.3736	2.5345	2.4653
A5	2.4723	2.8188	1.4562	1.3805	2.5624	1.1862	0.1314	2.2554	1.7327
B5	3.4869	1.7640	1.5697	2.6084	3.6829	0.4283	1.3773	2.6239	1.4111
C5	3.4494	1.6923	1.4793	2.5639	3.6373	0.2945	1.4066	2.4736	1.2305
A6	2.3955	3.1816	2.0609	1.5644	2.6296	1.8863	1.7886	1.1351	1.5156
B6	1.9390	3.5612	2.4210	1.9989	2.4438	2.2316	2.1576	2.5524	1.4668
C6	2.1460	3.4000	2.3479	2.1003	2.6039	2.1384	2.1857	2.4521	1.3088
A7	3.2674	1.9631	1.0914	2.1995	3.2410	1.0867	1.7025	1.4585	0.9821
B7	2.7157	2.3590	1.3237	2.0842	2.8654	1.1678	1.6326	2.1410	0.2995
C7	3.4429	1.9458	1.4994	2.8772	3.5311	1.3636	2.1888	2.8339	0.8098

Next, I computed the distance of the moments using the filled version of the head image I used in Table 4.8. The results are shown in Table 4.9. There are 14 erroneous distances, or 7.4%, a small increase compared to Table 4.8. More importantly, we see the errors changed somewhat: five problem cases in Table 4.8 are correct in Table 4.9, while six new problem comparisons arose. A total of four keys are misclassified, including B6 and C6.

The advantage of defining two sets for the key types which have two joints is clear in Table 4.8 and Table 4.9. We do not see any error in key type number 2 and 5 in either of these tables, while they accounted for half the errors in the previous subsection.

Next, I compute the average of the filled and unfilled distances (the average of Table 4.8 and Table 4.9). The results are shown in Table 4.10. In the experiment of Table 4.6, this average led to a modest improvement of the results. Here, the results are not much improved: there are again 12 erroneous distances and 4 misclassified keys, the same result as with the filled head images.

Table 4.9 Distances between 21 sample keys and the database (filled key heads)

	D1	D2	D22	D3	D4	D5	D55	D6	D7
A1	0.1245	3.0375	5.5188	2.3665	1.3320	8.8413	6.4870	8.5927	12.9331
B1	0.2849	2.8196	5.2977	2.1363	1.5491	8.6340	6.2835	8.3731	12.7225
C1	0.2471	2.7367	5.2285	2.0823	1.5899	8.5289	6.1744	8.2923	12.6292
A2	3.9329	1.4972	2.1879	1.8265	5.1763	5.5854	3.4618	5.0568	9.5787
B2	3.2631	0.4009	2.2261	1.2277	4.5671	5.6849	3.3018	5.2895	9.8501
C2	4.8104	1.8875	0.7413	2.6627	6.0852	4.2682	1.8886	3.7658	8.3937
A3	2.2092	1.0872	3.4822	0.7715	3.3957	6.5966	4.2408	6.4441	10.6874
B3	2.1412	1.0152	3.4735	0.4246	3.3467	6.6555	4.3163	6.4662	10.7303
C3	2.6113	0.7770	3.0459	0.3027	3.8219	6.3168	4.0110	6.0494	10.3571
A4	1.3580	4.2729	6.7516	3.5381	0.0600	9.9597	7.6287	9.8039	13.9646
B4	1.9673	4.8867	7.3261	4.1319	0.8819	10.6534	8.3126	10.4174	14.6437
C4	1.1375	3.9004	6.3522	3.1129	0.7733	9.6351	7.3194	9.4185	13.6215
A5	5.9947	2.9784	1.5138	3.3852	6.9958	3.9772	0.0810	5.8260	5.3729
B5	8.2642	5.1244	3.7876	5.4981	9.2286	1.8293	2.4493	4.6712	3.3033
C5	8.3114	5.2442	3.8845	5.6136	9.2591	2.0442	2.4936	4.7368	3.4429
A6	9.5960	6.3992	4.6648	6.9357	10.6761	2.8131	3.9226	2.0678	3.4857
B6	7.5220	4.3154	2.7487	4.7639	8.5478	2.5736	1.6413	4.4470	3.9414
C6	7.4037	4.2230	2.6404	4.6678	8.4241	2.7133	1.5089	4.5507	4.0644
A7	9.3166	5.9063	4.8278	6.2503	10.3344	1.7301	4.1136	4.5667	2.3394
B7	9.8587	6.7830	5.2700	7.1160	10.7648	2.2491	4.0697	4.2905	2.2216
C7	7.9127	4.7431	3.1949	5.1220	8.8881	2.3943	2.0949	4.4948	3.4228

Table 4.10 Average of distances of filled and unfilled key heads (first set)

	D1	D2	D22	D3	D4	D5	D55	D6	D7
A1	0.07655	4.18165	4.41875	1.99255	1.37135	6.3292	4.51835	5.1699	7.68985
B1	0.25735	4.0169	4.25905	1.7768	1.4469	6.1539	4.35045	5.051	7.5425
C1	0.1799	4.003	4.2575	1.839	1.53565	6.13705	4.3238	4.99	7.5088
A2	4.40135	0.9898	2.0208	2.7536	4.8637	3.4833	3.0562	4.3637	6.0523
B2	3.6495	0.918	1.6653	2.1529	4.17285	3.4949	2.6445	4.07035	5.77755
C2	3.9936	2.13705	0.45615	2.40775	4.3872	2.9062	1.6562	3.0652	4.8161
A3	1.93115	2.57775	2.8215	0.43875	2.4139	4.5407	2.8458	4.1299	6.16515
B3	1.8748	2.5833	2.85225	0.25565	2.36525	4.6059	2.9025	4.16605	6.226
C3	2.0124	2.49525	2.6432	0.2772	2.5413	4.4737	2.7437	3.89405	6.01375
A4	1.26635	4.73905	4.8875	2.47025	0.21155	6.88515	5.12675	6.0569	8.27715
B4	1.94535	4.86275	4.9824	2.78285	0.7196	7.11315	5.43085	6.53505	8.6098
C4	1.4703	4.33065	4.4856	2.11065	0.70095	6.51645	4.8465	5.9765	8.0434
A5	4.2335	2.8986	1.485	2.38285	4.7791	2.5817	0.1062	4.0407	3.5528
B5	5.87555	3.4442	2.67865	4.05325	6.45575	1.1288	1.9133	3.64755	2.3572
C5	5.8804	3.46825	2.6819	4.08875	6.4482	1.16935	1.9501	3.6052	2.3367
A6	5.99575	4.7904	3.36285	4.25005	6.65285	2.3497	2.8556	1.60145	2.50065
B6	4.7305	3.9383	2.58485	3.3814	5.4958	2.4026	1.89945	3.4997	2.7041
C6	4.77485	3.8115	2.49415	3.38405	5.514	2.42585	1.8473	3.5014	2.6866
A7	6.292	3.9347	2.9596	4.2249	6.7877	1.4084	2.90805	3.0126	1.66075
B7	6.2872	4.571	3.29685	4.6001	6.8151	1.70845	2.85115	3.21575	1.26055
C7	5.6778	3.34445	2.34715	3.9996	6.2096	1.87895	2.14185	3.66435	2.1163

I double-check the results; I computed the average distances of the 21 keys with a second set of filled and unfilled key heads from my database. The results are shown in Table 4.11. In this case three keys are misclassified, two of which, A7 and C7, were misclassified in Table 4.10 as well. 10 distances (5.2%) are erroneous. This is comparable to the results in Table 4.10.

Table 4.11 Average of distances of filled and unfilled key heads (second set)

	D1	D2	D22	D3	D4	D5	D55	D6	D7
A1	0.1160	4.1701	4.0702	2.2270	1.1799	6.4917	4.1876	7.3111	7.0321
B1	0.3293	4.0041	3.9176	2.0046	1.2830	6.3273	4.0203	7.1164	6.8899
C1	0.2279	4.0041	3.9074	2.0769	1.3352	6.3055	3.9916	7.1453	6.8595
A2	4.4779	0.7088	2.0097	2.4418	5.0062	3.8771	2.9018	5.3252	4.8663
B2	3.7163	0.8355	1.4352	2.1083	4.2587	3.7418	2.4376	5.3787	4.6591
C2	4.0620	1.9152	0.1925	2.2488	4.4900	3.1289	1.4848	4.4813	3.8746
A3	1.9970	2.7656	2.4751	0.8570	2.4337	4.8953	2.5387	5.6250	5.5824
B3	1.9462	2.7082	2.5265	0.6836	2.3998	4.9239	2.5926	5.6923	5.6225
C3	2.0890	2.4381	2.3561	0.2901	2.5791	4.6984	2.4372	5.5791	5.3746
A4	1.2455	4.7724	4.5303	2.7877	0.2593	7.1194	4.8083	7.9117	7.6469
B4	1.9331	4.8831	4.6430	3.1230	1.0899	7.3617	5.1287	8.1304	7.9228
C4	1.4827	4.3506	4.1591	2.4560	1.0171	6.7793	4.5425	7.5222	7.3766
A5	4.1739	3.0820	1.4389	2.5978	4.7475	2.2278	0.3031	2.4019	4.1565
B5	5.8155	3.7566	2.4365	4.3036	6.4341	0.8753	1.6079	2.0499	3.1396
C5	5.8206	3.7570	2.4632	4.3154	6.4324	0.8336	1.6416	2.0922	3.0669
A6	5.9309	5.0017	3.1491	4.4428	6.6849	2.2464	2.5726	1.3840	3.1254
B6	4.6706	4.2417	2.3979	3.6467	5.6142	2.2353	1.6206	0.7920	3.5223
C6	4.7142	4.1030	2.3172	3.6324	5.6200	2.2308	1.5578	0.9412	3.5342
A7	6.2148	4.2720	2.6268	4.4979	6.7040	1.9971	2.6403	2.1908	2.8539
B7	6.2224	4.8661	3.1104	4.8189	6.8307	1.8105	2.5412	1.9114	1.8948
C7	5.6133	3.7012	2.1257	4.2303	6.2028	1.7910	1.8274	1.7634	3.2634

The next step of the experiment is to compute the average distance between each of the 21 sample keys and two sets of key heads in the database. The results of that experiment are shown in Table 4.12. We find that the rate of erroneous distances decreases noticeably, from 6.3% or 5.2% to 3.1% (6 erroneous distances). This shows that the average of distances computed here is more reliable than the individual distances, and is the same observation we made in the results of Table 4.6. However, despite the fact that the number of errors in distances decreased, there are still four misclassified keys.

It seems discouraging that the number of misclassified keys is not decreasing. However, we should note first that this number is holding constant at 4 keys, while over the same tests in Table 4.6 it decreased from 11 to 3 keys. In other words, it is holding constant near the best result of Table 4.6. Moreover, most of the errors in Table 4.12 are due to only one of the key types in the database. That is type 5 (D5 and D55), which is responsible for 5 of the 6 erroneous distances. This means that we are dealing with a problem related to some specific key types, which hopefully we can deal with by adding one more average or by using an extra test like the head-to-hole ratio we used before.

Table 4.12 Average of distances between first and second sets

	D1	D2	D22	D3	D4	D5	D55	D6	D7
A1	0.09627	4.17587	4.24447	2.109775	1.27562	6.4104	4.352975	6.2405	7.360975
B1	0.29332	4.0105	4.08832	1.8907	1.36495	6.2406	4.185375	6.0837	7.2162
C1	0.2039	4.00355	4.08245	1.95795	1.43542	6.22127	4.1577	6.06765	7.18415
A2	4.43962	0.8493	2.01525	2.5977	4.93495	3.6802	2.979	4.84445	5.4593
B2	3.6829	0.87675	1.55025	2.1306	4.21577	3.6183	2.54105	4.724525	5.218325
C2	4.0278	2.02612	0.32432	2.328275	4.4386	3.0175	1.5705	3.77325	4.34535
A3	1.96407	2.67167	2.6483	0.647875	2.4238	4.718	2.69225	4.87745	5.873775
B3	1.9105	2.64575	2.68937	0.469625	2.38252	4.7649	2.74755	4.929175	5.92425
C3	2.0507	2.46667	2.49965	0.28365	2.5602	4.5860	2.59045	4.736575	5.694175
A4	1.25592	4.75572	4.7089	2.628975	0.23542	7.0022	4.967525	6.9843	7.962025
B4	1.93922	4.87292	4.8127	2.952925	0.90475	7.23742	5.279775	7.332725	8.2663
C4	1.4765	4.34062	4.32235	2.283325	0.85902	6.64787	4.6945	6.74935	7.71
A5	4.2037	2.9903	1.46195	2.490325	4.7633	2.40475	0.20465	3.2213	3.85465
B5	5.84552	3.6004	2.55757	4.178425	6.44492	1.00205	1.7606	2.848725	2.7484
C5	5.8505	3.61262	2.57255	4.202075	6.4403	1.00147	1.79585	2.8487	2.7018
A6	5.96332	4.8960	3.25597	4.346425	6.66887	2.29805	2.7141	1.492725	2.813025
B6	4.70055	4.09	2.49137	3.51405	5.555	2.31895	1.760025	2.14585	3.1132
C6	4.74452	3.9572	2.40567	3.508225	5.567	2.32832	1.70255	2.2213	3.1104
A7	6.2534	4.10335	2.7932	4.3614	6.74585	1.70275	2.774175	2.6017	2.257325
B7	6.2548	4.71855	3.20362	4.7095	6.8229	1.75947	2.696175	2.563575	1.577675
C7	5.64555	3.52282	2.23642	4.11495	6.2062	1.83497	1.984625	2.713875	2.68985

The final test is to compute the distances of the 21 keys with all three sample keys in the database. Table 4.13 shows the results of this comparison. The error rate is halved compared to using only two key images: two keys are misclassified instead of four, and 3 distances are erroneous instead of 6 (1.5% instead of 3.1%). Both this decrease and the final values are in line with the results from the experiment of Table 4.6.

In Table 4.13, A7 and C7 are matched incorrectly to D5 and D55. That is the same misclassification that remained in Table 4.6. Again, using the technique that computes the ratio between white pixels of the head and black pixels of the hole of the key head, we will see that eventually, A7 and C7 go to D7 and all sample keys in this test will be linked to the correct key blank information.

Table 4.13 Average of distances between three sets of the database

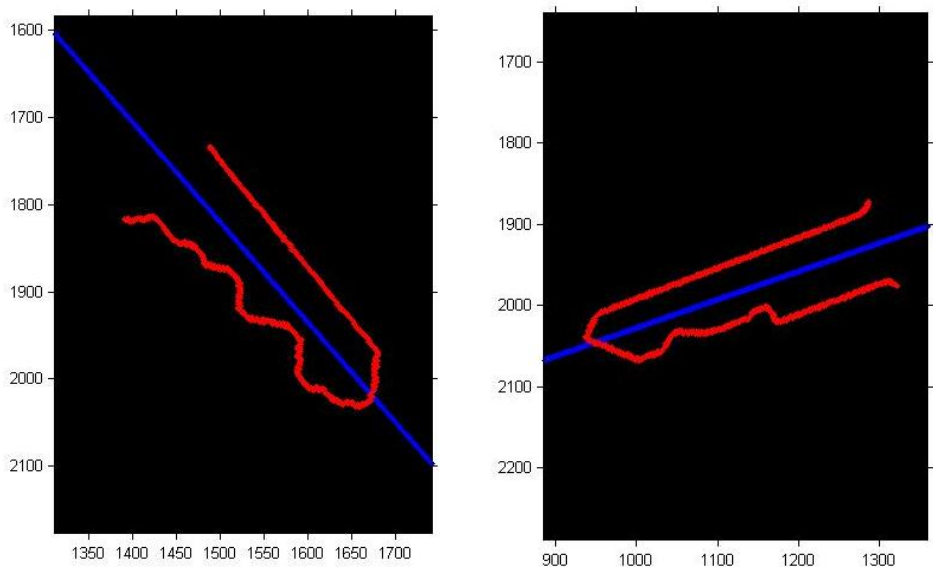
	D1	D2	D22	D3	D4	D5	D55	D6	D7
A1	0.164083	4.17885	4.24985	2.122417	1.26685	6.323	4.3764	5.8646	6.518483
B1	0.242217	4.018067	4.09435	1.904067	1.3529	6.1576	4.2116	5.7136	6.368067
C1	0.2151	4.004733	4.087667	1.967533	1.430683	6.1308	4.1807	5.6889	6.341267
A2	4.346783	0.954933	2.087667	2.598667	4.9379	3.6138	3.0167	4.5046	4.745333
B2	3.609633	0.8399	1.617	2.095567	4.226917	3.5136	2.5637	4.3249	4.475183
C2	3.9554	2.036117	0.348283	2.30015	4.4516	2.9537	1.5967	3.3846	3.5688
A3	1.89275	2.67445	2.650433	0.630817	2.422733	4.6520	2.7284	4.5168	4.977217
B3	1.828633	2.656867	2.69375	0.460717	2.379217	4.7012	2.7860	4.5722	5.035367
C3	1.958867	2.489617	2.5053	0.2386	2.555967	4.5235	2.6296	4.3733	4.82575
A4	1.314683	4.76375	4.7093	2.650983	0.167983	6.9315	4.9928	6.6196	7.104117
B4	1.981083	4.890483	4.816567	2.983717	0.8858	7.1818	5.3093	6.9891	7.408233
C4	1.492167	4.36435	4.326867	2.315017	0.83335	6.5958	4.7280	6.4094	6.844433
A5	4.135	3.009533	1.4032	2.44855	4.7777	2.2974	0.2358	2.8007	3.023767
B5	5.77765	3.612767	2.528783	4.142283	6.461017	0.7852	1.7339	2.4614	2.343667
C5	5.784033	3.623217	2.5434	4.165183	6.456067	0.8279	1.7648	2.4610	2.304233
A6	5.882917	4.925367	3.206517	4.31805	6.66415	2.3556	2.7093	1.4777	2.574583
B6	4.6348	4.0906	2.433217	3.4815	5.5582	2.2241	1.7449	1.6274	2.508367
C6	4.67815	3.955967	2.349317	3.47545	5.570133	2.2314	1.6870	1.6725	2.482567
A7	6.163633	4.153667	2.781167	4.3455	6.7469	1.8379	2.7863	2.4443	2.150583
B7	6.187867	4.7217	3.15885	4.678567	6.8332	1.7822	2.6693	2.295	1.617583
C7	5.5803	3.501717	2.20375	4.084733	6.219667	1.7319	1.9626	2.2504	2.2035

4.3 Finding precise measures of the edges

4.3.1 Image to Key

After finding the minimum distance of the seven moment invarients and the correct key blank, I can get the relevant information about the length of the key. This will help me convert the measures from pixels into millimetre. In this second test I will show how precisely the software can find the edges of the key. I will compare the results obtained by the software and the results I measure by a calliper with respect to the major axis line. Then I will show the error rate of the distances computed between the edges and major axis line of the key. The keys I will use for the experiments in this section are B2 and B7 from Figure 4.2.

The images are converted into binary format and the shanks are separated from the key images. The contour is found using canny edge detection and then refined using sub-pixel interpolation, as I explained in 3.5.3. Figure 4.5 shows the sub-pixel interpolation points of the shanks.



a b

Figure 4.3 Sub-pixel edge points of the key shanks

One of the outputs of the operation that is done on an image by the software is an array that contains the coordinates of the all sub-pixel edge points (red dots) and their distance (in millimetre) to the major axis line which is the blue line shown in Figure 4.5. The distance between the points and the line can be a good criterion to see if the measured edge points in the key image are close to real measurement of the key using a calliper.

Figure 4.6 shows five different edge points that are selected on the key shank image (Figure 4.5(a)) for the test. The points are shown in circles. The measured distances computed by the software and the calliper are illustrated in Table 4.14.

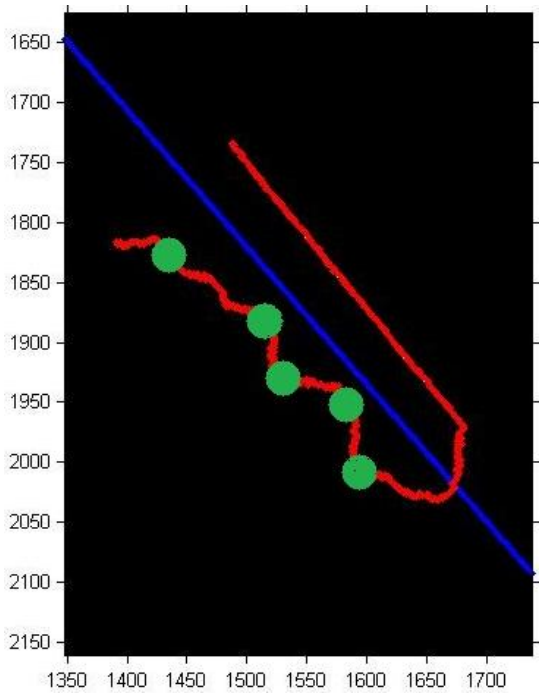


Figure 4.4 Selected points on the key shank (Figure 4.5(a))

Table 4.14 Distances (in mm) of the green points in Figure 4.6 measured by software and calliper

Coordinates	Distance (software)	Distance (calliper)	Relative error %
2.0035 1.5955	4.1	4.06	0.99
1.9475 1.5845	1.7	1.77	-3.95
1.9265 1.5285	4.1	4.19	-2.14
1.8805 1.5156	2.4	2.54	-5.51
1.8235 1.4336	4.4	4.57	-3.72

The relative error in that table was computed using Equation (4-1). The average relative error of the five sample points in this example is 3.2%.

(4-1)

$$Error = \frac{approximation - value}{value} \times 100$$

The same experiment was conducted with the second selected image. The five edge points are shown in green in Figure 4.7. Table 4.15 shows the distances of the points measured by software and the calliper.

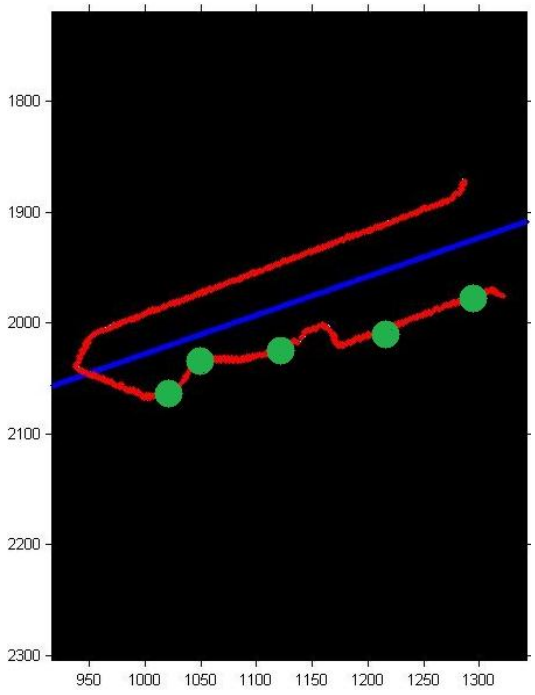


Figure 4.5 Selected green points on the key shank (Figure 4.5(b))

Table 4.15 Distances in mm of the green points in Figure 4.7 measured by software and calliper

Coordinates	Distance (software)	Distance (calliper)	Relative error %
2061.3 1018.5	3.2	3.30	-3.03
2032.5 1047.6	1.7	1.77	-3.95
2023.0 1119.5	3.0	3.04	-1.31
2005.6 1214.5	4.3	4.31	-0.23
1976.4 1291.5	4.1	4.31	-4.87

The average relative error of the five sample points in this example is 2.6%.

The errors in the edge points of Figure 4.6 and Figure 4.7 could be the result of several issues. One of the problems could be the resolution of the picture I am working on. A high resolution scanner can take better pictures so I could identify the edges and cuts more precisely. A better interpolation method would also yield more accurate subpixel coordinates, and improve the final results. On the non-technical side, the calliper I used to measure the length of the key for the database, the measure that was used as a constant to convert my subpixel coordinates to millimetres, has a maximum accuracy of 1 millimetre or 0.01 inch[5]. In this thesis I measured the distances with inch scale and converted them into millimetre. The accuracy of the computed major axis line can be a source of error. The experimental results show that these errors are negligible.

[5] The calliper I used in this test has two scales.

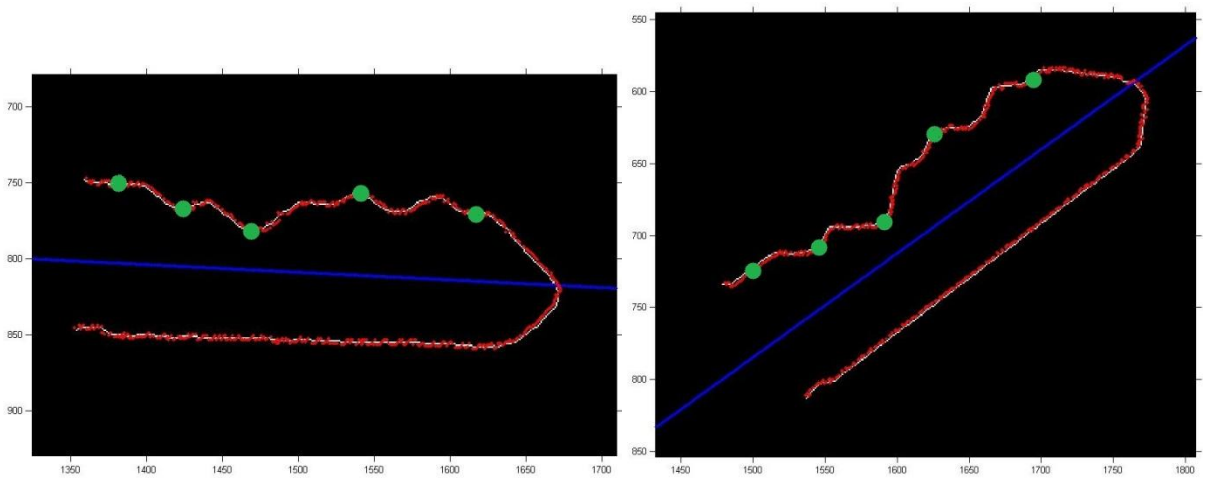
4.3.2 Image to Image

The purpose of this next test is to verify that two executions of my software using two scans of the same key gives similar results.

For this test, I will use the two images of the same key shown in Figure 4.8. I selected five points on the grooves of the keys and measured the distance between the points and the major axis line using software. In Figure 4.9 I illustrate the key shanks along with the subpixel points and the selected points to measure. Table 4.16 shows the results and error obtained between these two keys using Equation (4-1), along with the results from the calliper for comparison.

a b

Figure 4.6 Two images of a key.



a b

Figure 4.7 Selected points on two scans of the key.

Table 4.16 Measures of the points on the two scans of the key by the software and calliper in mm

Coordinates key a	Software key a	Coordinates Key b	Software key b	Error a to b %	Calliper	Error calliper to a %	Error calliper to b %
769.8 1616.5	3.7	590.5 1695.5	3.6	-2.8	3.30	12.1	9.1
755.5 1540.7	4.6	628.5 1625.3	4.5	-2.2	4.31	6.7	4.4
782.5 1469.5	2.1	689.5 1591.5	2.0	-4.8	1.90	10.5	5.2
765.6 1424.5	3.3	707.5 1546.5	3.0	-9.1	3.04	8.5	1.3
749.5 1380.2	4.4	724.5 1499.5	4.1	-6.8	4.31	2.0	-4.9

As Table 4.16 shows, the average of error between two scans of the same key is 6.4% and is always below 10%. By comparison, the error between calliper measures of the original key and those images is on average 6.5%, and at most 12%. It thus seems that the results of my software are very accurate given two different scans of the same key. The error rate between the two scans is comparable to the error rate from the original key to each image.

4.3.3 Image to Cut Key to Original Key

In this test, I take the measures of both the original and duplicated keys shown in Figure 4.8, and compare the error of five points between the original and duplicate key to the error between the original and my software's measures. This will tell us how precise the image processing measures are compared to a traditional mechanical system. I performed this test three times using three scans of the original key to generate reliable results.

Table 4.17 measures of the points on the first original image in mm

Calliper Original	Calliper Duplicated	Coordinates	Software Original	Error Original to Duplicated %	Error Original to Software %
3.30	3.17	769.8 1616.5	3.7	-3.9	12.1
4.31	4.06	755.5 1540.7	4.6	-5.8	6.7
1.90	1.77	782.5 1469.5	2.1	-6.8	10.5
3.04	2.79	765.6 1424.5	3.3	-8.2	8.5
4.31	4.31	749.5 1380.2	4.4	0.000	2.0

Table 4.18 measures of the points on the second original image in mm

Calliper Original	Calliper Duplicated	Coordinates	Software Original	Error Original to Duplicated	Error Original to Software %
3.30	3.17	1401.5 0584.5	3.5	-3.9	6.0
4.31	4.06	1431.5 0592.5	4.4	-5.8	2.0
1.90	1.77	1507.5 0683.2	1.9	-6.8	0
3.04	2.79	1550.0 0694.1	3.2	-8.2	5.2
4.31	4.31	1599.5 0710.7	4.4	0.000	2.0

Table 4.19 measures of the points on the third original image in mm

Calliper Original	Calliper Duplicated	Coordinates	Software Original	Error Original to Duplicated %	Error Original to Software %
3.30	3.17	1756.5 0856.5	3.3	-3.9	0
4.31	4.06	1899.5 0931.7	4.2	-5.8	-2.5
1.90	1.77	1973.5 1048.9	2.0	-6.8	5.2
3.04	2.79	2066.5 1081.5	3.1	-8.2	1.9
4.31	4.31	2158.5 1112.5	4.3	0.000	-0.2

On average, a traditional physical duplication of a key gives an average error of 4.9%. My system, on the other hand, gives an average error of 7.9% with the first image, of 3% with the second image, and of 1.9% with the third image.

On average between these three original images the percentage of error is 4.2% which is comparable to the traditional physical duplication of a key. The results of individual images show that it is possible that the measures computed by the system can be up to 3% better or worse than the traditional measurements, which in real terms corresponds to a negligible error of a few tenths of a millimetre only. The main reason for this variation in the results is the quality of the scanned image, which is crucial to measure the edges accurately.

Finally, we can consider the sign of the error throughout the tests of this section. The sign tells us in whether the software detected the edge too far outside the key (if it is positive) or too far inside the key (if it is negative). Considering tables 4.14 to 4.19, we find that there is not one dominant orientation for the error, meaning that the edge detection and interpolation step is not biased in one orientation. However, individual tables do show a bias, and can have mostly positive errors (such as for Table 4.17) or mostly negative errors (Table 4.15). So the edges detected for an individual key can be entirely too far inside or outside the key, and both types of error occur for different key images. Consequently, if it were possible to detect which situation an individual image is in, it could be possible to correct the entire set of edge points. This can be the focus of future development.

5 Conclusions and Future Research

5.1 Contributions

The contribution of this thesis has been the development of a new key-measuring algorithm instead of using sophisticated cameras like existing systems; I only used a commercially-available flatbed scanner. The software uses image processing techniques to compensate for the lack of accuracy in the key images taken by the scanner. Given an image of an original key, this algorithm serves two main functions:

1. To find the corresponding key blank type in a key model database;
2. To identify the coordinates of the edge points of the key's dents and measure the depth of the cuts of the original key in millimetres.

One important requirement was to have a software which can receive the key image with any orientation of the key on the surface of the scanner and on any positions, to make the system easy to use for a human operator. This has been accomplished by having the software find the largest object of the image, the key image, and then compute the orientation of the key. All following steps are accomplished taking into account the computed angle of the key.

The development of the key model database was also an important part of this work. This database stores pictures, measures and information about different keys. In order to find the correct match of the original key, the software compares the head of the original key with images of other heads in the database.

Once the software knows the orientation and position of the key image, it separates the shank of the key at the joint and keeps the pixels that are located at the edges of the cuts. An interpolation method is employed by the software to find the sub-pixel coordinates of the points. Using the measures retrieved from the database, the software can then convert these sub-pixel coordinates to actual physical measures to duplicate a new key.

5.2 Summary of the Results

The results generated focused on thoroughly testing the realization of the two main objectives of the algorithm, as highlighted above.

The results showed that the algorithm can accurately find the blank key corresponding to an original scanned key. The results further confirmed that the optimal comparison uses three different heads images of each key model and the average of filled and unfilled key heads. At less than three images we still find errors. On the other hand, only one ambiguous case remains when using three images, and I can deal with it by using some extra statistics, namely the size of the hole in the key head.

For the measures of the edges, I find on average 2% to 7% relative error in the depth of the cuts. This corresponds only to a fraction of a millimetre. Using the traditional key duplication system, there is 4.9% relative error on average between the original key and the duplicated one. By the new algorithm proposed using the image of the original key and the proposed image processing techniques, there is on average, 4.2% relative error which is 0.7% less than the error that traditional mechanical devices make

which is comparable to the traditional physical duplication of a key. This shows the quality of the scanned image has crucial effects to measure the edges accurately.

5.3 Future work

The database I built in this thesis only defines 7 different key models in the database. They were selected to be a representative sample, and included both single-sided and two-sided keys, keys with double-joints, and some similar-looking key models. However, there are more than 200 key types on the market today. One direction for future expansion of the software will be building a more complete database. For sure, covering more key types has a lot more different challenges to solve. There are some key types which are not symmetrical, which our software will need to deal with by adding a horizontal offset to the major axis line. Some different key types have the same head shapes so moment invariants alone will not be able to differentiate these types and extra information will be needed to disambiguate these cases, such as the hole-to-head ratio I experimented with in this project. Another practical expansion of my system will be connecting it to a means to actually duplicating keys. This could be a key-cutting device modified to accept my software's measures as input, or a 3D printer that would create an entire new key.

In this thesis I used the seven moment invariants to model and compare the head of the keys, as suggested by Hu [19]. While my experiments show that this measure is good and can allow me to find matching head pictures, it is not the only possibility for that purpose. For example, J. Flusser and T. Suk [11], [10] showed that the traditional Hu's invariant set is neither independent nor complete because on the original Hu's set is missing the third order independent moment invariant. Consequently, future work should explore the use of other invariant measures, such as the one proposed by Flusser and Suk. It is possible that these new measures will simplify the system by requiring fewer database images to average.

For the second objective of measuring the edges of the key, a crucial step is obtaining the boundary of the contour of the key shank. Indeed, that step is the start of the process to measure the indentations of the original key shank precisely. In this thesis I defined a constant threshold percentage to convert the grayscale image into binary format. In the future, the threshold should be computed dynamically by the software based on the illumination of the image; this would improve the accuracy of finding the edge pixels on the contour.

Finally, using a more precise interpolation method will lead to measuring the sub-pixel points more accurately, and thus reducing the error on the depth measures. A more traditional sub-pixel interpolation method commonly used in image processing software is bicubic interpolation [22], which should be the first improvement to try. More advanced methods have been proposed for applications that require high accuracy [5], [6]. The linear kernel interpolation was recently discovered by Grevera and Udapa is one of the most frequently mentioned methods in publications during recent years [16]. In general, large kernel sizes were found to be superior to small interpolation masks. For example, using quadratic 3×3 instead of cubic 2×2 interpolations, the interpolation error is decreased further [22], [20], [8]. Another interpolation method that could be used is the B-spline interpolator presents one of the best results in terms of similarity to the original image, and in comparison to other methods, it runs the fastest [23].

Bibliography

[1] R. Almblad, J. Blin, P. Jurczak, "Method and apparatus for automatically making keys", U.S. Patent 5 807 042, September 1998.

[2] T. Bose, "Image Processing Fundamentals," in Digital *Signal and Image Processing*, Wiley, Inc., Danvers, MA (2004).

[3] J. Campbell, G. Heredia, M. A. Mueller, "Key Identification System", U.S. Patent 6 836 553, December 2004.

[4] J. Canny, "A Computational Approach to Edge Detection," IEEE Trans*, Pattern Analysis and Machine Intelligence,* vol. 8, no. 6, pp. 679-697, Nov. 1986.

[5] P. E. Danielsson and M. Hammerin, "High accuracy rotation of images", Department of Electrical Engineering, Link¨oping University, Sweden, Tech. Rep. LiTH-ISY-I-11521990, 1992.

[6] P. E. Danielsson and M. Hammerin, "Note: High accuracy rotation of images," *CVGIP: Graph. Models Image Processing*, vol. 54, no. 4, pp. 340–344, 1992.

[7] S. Eddins. (2010). *Visualizing regionprops ellipse measurements*, [online], Available: http://blogs.mathworks.com/steve/page/5/

[8] O. D. Evans and Y. Kim, "Efficient implementation of image warping on a multimedia processor," *Real Time Image*, vol. 4, pp. 417–428, 1998.

[9] L. Fletcher. (2010). *Binary Image Analysis* [online], Available: http://users.cecs.anu.edu.au/~luke/cvcourse_files/online_notes/lectures_2D_2_binary_morph_6up.pdf

[10] J. Flusser, "On the Independence of Rotation Moment Invariants", Pattern Recognition, vol. 33, pp. 1405 1410, 2000.

[11] J. Flusser, T.Suk, "Rotation Moment Invariants for Recognition of Symmetric Objects", Inst. of Inf. Theor. Autom., Acad. of Sci. of the Czech Republic, pp. 3784 – 3790, December 2006.

[12] Y. Gao, M. K.H Leung, "Face Recognition Using Line Edge Map", Pattern Analysis and Machine Intelligence, pp. 764 – 779, August 2002.

[13] W. Garner. (2011). *Shortest Distance from a Point to a Line,* [online]. Available: http://www.math.ucsd.edu/~wgarner/math4c/index.htm

[14] R. C. Gonzalez, R. E. Woods, "*Digital Image Processing*", Prentice Hall, Inc., 2nd ed. New Jersey (2002).

[15] R. C. Gonzalez, R. E. Woods, S. L. Eddins, "*Digital Image Processing Using MATLAB*," Tata McGraw Hill Education Private Limited, New Delhi, India (2010).

[16] G. J. Grevera and J. K. Udapa, "An objective comparison of 3-D image interpolation methods," *IEEE Trans. Med. Imag.*, vol. 17, no. 4, pp. 642–652, 1998.

[17] D. W. Harder, R. Khoury. (2010). *Numerical Analysis for Engineering* [online]. Available: https://ece.uwaterloo.ca/~dwharder/NumericalAnalysis/05Interpolation/

[18] M. Heath, S. Sarkar, T. Sanocki and K. Bowyer, "Comparison of Edge Detectors: A Methodology and Initial Study," Proceedings CVPR '96, 1996 IEEE Computer Society Conference., pp. 143 – 148, 1996.

[19] M. K. Hu, "Visual pattern recognition by moment invariants", Information Theory, IRE Transactions, pp. 179-187 February 1962.

[20] R. Keys, "Cubic convolution interpolation for digital image processing", Acoustics, Speech and Signal Processing, IEEE Transactions, pp. 1153 - 1160, December 1981.

[21] S. Khan. (2010). *Draw Bounding Box* [online], Available: http://www.mathworks.com/matlabcentral/fileexchange/26296-draw-bounding-box

[22] T.M. Lehmann, C. Gonner, K. Spitzer, "Survey: interpolation methods in medical image processing", Inst. Of Med. Inf., Tech. Hochschule Aachen, Germany, pp. 1049 – 1075, November 1999.

[23] E. Maeland, "On the comparison of interpolation methods," *IEEE Trans. Med. Image.*, vol. MI-7, pp. 213 217, 1988.

[24] R. M. Prejean, "Key Manufacturing Method" U.S. Patent 6 647 308, November 2003.

[25] S. Pacenzia, E. Casangrande, E. Foscan, "Method To Identify a Key Profile, Machine To Implement The Method and Apparatus for the Duplication of Keys Utilizing the Machine", U. S. Patent 6 895 100, May 2005.

[26] Shermina.J, "Illumination Invariants Face Recognition Using Discrete Cosine Transform And Principal Component Analysis", Emerging Trends in Electrical and Computer Technology (ICETECT), pp. 826 830, May 2011.

[27] The MathWorks, Inc. (2010)., *Image Processing Toolbox*, [online]. Available: http://www.mathworks.com/products/image/index_b.html

[28] J. S. Titus, J. E. Bolkom, "Key Measurement Apparatus and Method", U.S. Patent 6 406 227, June 2002.

[29] J. S. Titus, W. Laughlin, J. E. Bolkom, "Key Duplication Apparatus and Method", U.S. Patent 6 152 662, November 2000.

[30] P. R. Wills, R. F. Kromann, N. N. Axelrod, W. A. Schroeder, J. A. Berilla, B. Burba, "Method andApparatus for Using Light to Identify a Key", U.S. Patent 6 064 747, May 2000.

[31] V. Yanovsky, "Shadow image acquisition device", U.S. Patent 6 175 638, January 2001

[32] V. Yanovsky, A. Sirota, "Key Imaging System and Method", U.S. Patent 6 449 381, September 2002.