POINT OBJECT EXTRACTION FROM SCANNED TOPOGRAPHIC MAPS

By

Yuying Chen

A thesis submitted in partial fulfillment of

the requirements for the degree of

Master of Science

(Cartography and GIS)

at the

UNIVERSITY OF WISCONSIN-MADISON

2015

Acknowledgements

First, I would like to thank my academic advisor Dr. Jim Burt of the University of Wisconsin-Madison, Department of Geography for his constant guidance and encouragement throughout this challenging project. It was a pleasure working with him. I will always be grateful for his expertise and support these past two years.

I would also like to thank my committee members Dr. A-Xing Zhu and Dr. Qunying Huang—both of them provided valuable advice and feedback throughout the process. They helped me to think carefully about each step of the project.

I am also grateful to Jing Liu, Fei Du, Guiming Zhang. They all provided critical feedback and useful suggestions in different stages of the work. They have also helped me both in academics and in life during the past two years.

I am thankful to the United States Geological Survey, which provided support for the project.

Finally, I want to thank all my family and friends for their support in my life. I could not achieve anything without them.

Table of Contents

1 Introduction

Geographic information science has been developing quickly in recent years. It has provided software in the form of geographic information systems (GIS) that offer powerful tools to process and analyze geographic information. It has also created great demand for digital geographic information. Topographic maps are one of the best and most abundant sources of geographical information. A topographic map is a graphic representation of cultural and natural objects on the ground. Most topographic maps are prepared by government agencies as part of ongoing projects. For example, the United States Geological Survey (USGS) has mapped all U.S. states and territories at scales as large as 1:24000, and has issued hundreds of thousands of individual map sheets over the last 100-plus years. The entire collection has been scanned at high resolution (500-600 dots per inch), georeferenced, and it is available through the National Map website[1]. This opens the door for the geographic data in topographic maps to be processed by GIS and used in different applications. For example, in the Mineral Deposit Database Project (USMIN) of USGS, mine symbols in USGS topographic maps are being cataloged to develop a mineral deposit database for the United States. Such a database can be used in mineral resource assessment projects, land management planning, abandoned mine programs, etc.

The traditional way to extract map information is manual digitization, which is done by tracing over the map objects either on a digitizing tablet or by heads-up screen digitizing. This is a time-consuming and expensive procedure and thus impractical for large map

[1] nationalmap.gov

suppliers like USGS. For example, in the USMIN, there are more than 1,900 7.5-minute quadrangle maps with over 105,000 mine symbols to be detected in the state of Nevada alone. Therefore, it is highly desirable to automate the extraction of these symbols from scanned topographic maps [1].

1.1 Research Problem and Challenge

The fundamental research problem this thesis addresses is how to extract point objects automatically from scanned topographic maps. Map objects include natural objects such as rivers and lakes as well as artificial objects like roads and mines. Some objects have spatial extent (such as a lake or road), whereas others are conceptualized as point objects defined by a single coordinate location. This research is a case study that focuses on the extraction of mine symbols from USGS topographic maps. All of the mine symbols of interest here are point objects. Of course, topographic maps contain many other point objects, as well as linear and areal objects. They also include textual information, which in some databases is also considered an object class.

Like other objects, extracting point objects faces the challenge of object intersection and scanning artifacts. Many different kinds of map objects can overlap with one another and form a complex mixture. Any map object extraction method has to perform some sort of separation to identify the target objects. When scanning artifacts, problems such as aliasing (Figure 1-1), the blurring and mixing of colors, further complicates the process. For example, contour lines are one of the main objects on topographic maps, but these intersect with most other map objects. Since contours are not usually the same color as the other objects on the map, pixels near the intersecting area are frequently affected by the blending of colors.

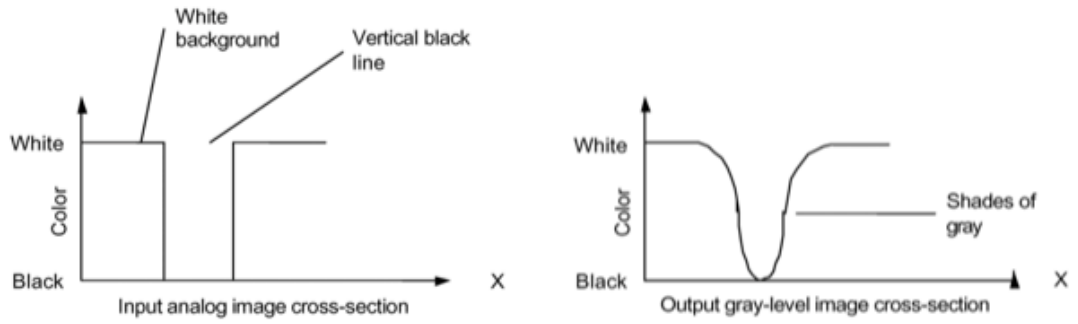


Figure 1-1: Aliasing effect of a black line on a white background in the gray-scale cross section [1].

In addition to these challenges, the extraction of point objects can be complicated by other issues as well. Some point objects, like the mine symbols in the case study, are very small compared to the size of the scanned maps and may disappear due to image down-sampling. Therefore, the extraction must be processed at full image resolution. Some point objects can easily be confused with other map objects. For example, in Figure 1-2b, tow long lines cross each other and create a shape that is very similar to prospect pit within the rectangular detecting window. Another problem is the rotation and appearance variation of symbols. Figure 1-3b shows a hand-drawn quarry symbol on the map, the shape of which is difference from the standard one shown in map legend.

(a) (b)

Figure 1-2: (a) Prospect pit (b) Similar shape formed by two crossed lines.

(a) (b)

Figure 1-3: (a) Standard quarry shown in map legend (b) Hand-drawn quarry on map.

1.2 Research Approach

To address the above question, this research developed a new method to automate the extraction of point objects from scanned topographic maps. The method first preprocesses scanned maps to remove background noise. Then the extraction of point objects is performed using a combination of template matching and Oriented FAST and Rotated BRIEF (ORB) feature matching.

This preprocessing step utilizes the technique of color segmentation, which is widely used in object extraction from color maps. It first converts the scanned map from RGB (red, green, blue) to the HSV (hue, saturation, value) color space, a step that greatly simplifies the selection of the color range of the target object. Noise pixels with a color value outside of this range can then be removed.

After the preprocessing, a three-stage template matching approach is performed to extract target objects. The first stage focuses on the entire structure of the target objects and is optimized for speed, but generates many false matches. The second stage runs a connected component test. The third stage performs another round of template matching that focuses on individual segments of the target symbols.

ORB feature matching is performed on the locations that pass template matching. In computer vision, the term “feature” refers to a unique image structure such as a corner that

can be used to identify target objects like mine symbols in different images. ORB matches image features between the image of target object and the image that might contain the object. It can then be determined whether a target object in the latter one exists based on the matching result. This method can be used to find false hits that cannot be easily removed by template matching, especially for some hand-drawn symbols.

1.3 Contribution

The proposed method combines template matching and ORB feature detection. Template matching is fast and can be configured so that all objects are found, but only at the cost of incorrectly labeling many locations as containing an object (false positives). By comparison, ORB is sophisticated and accurate, but far too slow to be applied to large topographic map images. The combined use of template matching and ORB constitutes the main theoretical contribution of this work. In addition, this thesis provides a software tool that can be used by others for map object extraction.

The proposed method is applied in a case study that extracts mine symbols from USGS historical topographic maps. These objects are valuable mine resources. Some of them may constitute a significant environmental hazard. Therefore, it is important to extract them so that they can be effectively integrated with other spatial layers to facilitate scientific research. The extraction result also helps to build a mineral deposit database, which is part of the USMIN project of USGS. All the tested topographic maps have high complexity and high density of objects. This method helps to reduce the processing time for one scanned map from several hours (by manual digitization) to less than an hour, and it also requires much

less user input compared to the traditional method. In general, it proves to be both reliable and effective in the extraction of point objects from scanned topographic maps.

1.4 Thesis Structure

The remainder of the thesis is divided into four sections. Section 2 presents an overview of related work. Section 3 describes the methodology, and it is divided into four subsections. First, an introduction of data sources and project requirements is provided. The next section explains the removal of background noise, which is followed by discussion of all the object extraction methods, including template matching using two types of templates and ORB feature detection. Section 4 presents the case study results and some performance comparison with other methods. Finally, Section 5 discusses some conclusions and future research suggestions.

This thesis represents a collaborative effort involving myself, my advisor Jim Burt, and personnel in the USMIN project directed by research geologist Greg Fernette. Throughout the thesis I use the pronoun “we” to refer to that group.

2 Literature Review

2.1 Map Object Extraction

The majority of map object extraction methods reported in the literature focus on the extraction of linear objects or text information, but relatively few have focused on the extraction of point objects. Some map understanding systems have been developed to extract contours lines and area objects like buildings. The involvement of highly trained expertise is still commonly required. Moreover, the problem of small object extraction from complex topographic maps with dense intersecting objects is rarely properly addressed.

Khotanzad and Zink [2] present a method for the extraction of contour lines from scanned topographical maps with aliasing and false colors. They use a color key set and RGB color histogram analysis to preprocess the scanned image. Then a valley-seeking algorithm is performed to extract linear objects. Finally, they apply a pathfinding algorithm to close gaps in linear objects caused by intersecting objects. The method was shown to work well on separating objects with different colors on scanned maps.

Yamada, Yamamoto and Hosokawa [3] present an algorithm to extract linear and area objects. The algorithm first computes the directional object plane from the input images. Then the linear objects are extracted using erosion-dilation operations on the directional object planes. The area symbols are extracted using the MAP matching method. However, the algorithm only works on binary maps and it cannot be easily modified to extract of objects from scanned color maps.

Cao and Tan [4] present a method to detect and extract characters that are touching graphics in street maps. It segments intersecting text and graph using line continuation

combined with object line width. The method works well when the amount of intersection is relatively small, but the recognition rate is far lower when the number of intersections increases. Therefore, it is not suitable for scanned topographic maps that contain large quantities of intersecting objects.

Velázquez and Levachkine [5] separate and recognize the touching/overlapping alphanumeric characters from maps. For target text objects, the method trains an artificial neural network to recognize characters. The method assumes that the target object can be separated from other map objects by converting the inputs into binary maps. However, many point objects are very small and may disappear after this kind of conversion, so this method cannot be used to separate point objects.

Dhar and Chanda [6] present a method for extracting and recognizing symbol objects from topographic maps. The method first separates the map into layers and then recognizes the objects in different layers based on symbol-specific geometrical and morphological attributes. It assumes that different objects are in separate colors, but the problem of intersecting objects with the same color is not well addressed. Therefore, it cannot be used to extract the black mine symbols that may intersect with many other black map elements.

Pezeshk and Tutwiler [7] use a semiautomatic method to extract contour lines from scanned color images of topographic maps. The contour lines extraction algorithm utilizes the quantization of the intensity image combined with contrast limited adaptive histogram equalization. This method could be used in extracting objects from maps containing mixed color pixels and aliasing; however, it requires user involvement throughout the object extraction process. Therefore, it has limited use in batch processing scanned topographic maps.

Kerle and Leeuw [8] use object-oriented analysis to extract objects from legacy maps. The map images are first segmented based on their color information. Then each segment is classified according to its attributes like color, size and shape. The method could extract objects with variations of color or size and intersection with other objects. However, it also requires user involvement through the extraction and therefore cannot be applied in projects like ours.

2.2 Template Matching

Template matching is a technique in digital image processing. A template is a prototype, or archetypical example of the target object. In template matching one examines a location in the image to see how well pixels surrounding the location correspond to the template. If image pixels are consistent with the template, the location is labeled as a hit or positive detection. If an object is actually present at the location the detection is a true positive, otherwise it is a false positive. Template matching can be used in manufacturing as a part of quality control [12], a way to navigate a mobile robot [13], or a way to detect objects in images. It also has been applied to the field of map object extraction in some previous works, and it is used in this thesis.

Leyk, Boesch and R. Weibel [9] developed a multi-step recognition process to extract forest cover information from manually produced scanned historical topographic maps. Their method combines character recognition, line detection and structural analysis of forest symbols. Background noise is first removed by performing color segmentation. Then a two-stage template-matching algorithm is performed to detect text area. The final forest area is extracted using an object-oriented method. The performance of template matching relies on

the quality of background noise removal. The method cannot be extended for the extraction of features that cannot be easily separated by color segmentation.

Kim and Araújo [14] proposed a gray-scale template matching method that is invariant to rotation, scale, translation, brightness and contrast. The traditional "brute force" template matching rotates through every angle, translates to every position and scales by every factor. Their method substantially accelerates this process, while obtaining the same result as the original brute force algorithm. This method can detect objects with various orientation and scale. However, it is too slow for the batch processing of large map images.

Lin and Chen [15] presented a new method for template matching that is invariant to image translation, rotation and scaling. The method first converts a 2D template into a 1D gray-level signal. Then it performs template matching by constructing a parametric template vector of the 1D gray-level signal with different scaled templates of the object. The method is not only invariant to rotation and scale, but also conceptually simple and easy to implement. Like the previous method, the application of this method is limited by its processing speed.

2.3 Image Feature Detection

In computer vision, "features" are unique image structures that can be used to identify objects in different images. Feature detection algorithms examine each image pixel and its surrounding region to see if there is a feature present at the pixel. If so, then the center pixel is a keypoint. Once features have been detected, the image patch around each keypoint can be extracted and encoded, resulting in the so-called "feature descriptors" described below. By comparing such descriptors, we are able to match features in different images to identify target objects.

Image feature detection and matching is one of the basic issues in computer vision. Much research has been carried out on image matching and numerous algorithms have been developed. However, these techniques have rarely been used in the field of map object extraction.

One of the basic image features is corner. Corners are rotation invariant, which means we can find the same corner even when the image is rotated. Harris (Chris Harris & Mike Stephens, 1988) detects corners by finding difference in intensity for a displacement of (u,v) in all directions. It creates two eigenvalues to describe whether a region is corner, edge or flat. The final result of Harris corner detector is a gray-scale image with the "corner score." Such result can be used as the measure of corner in more advanced feature detection algorithms.

Harris corner detector cannot detect features with scale change. In 2004, Lowe developed a new algorithm called Scale Invariant Feature Transform (SIFT) [16]. The algorithm first extracts potential keypoints by finding the local extreme over scale and space. Then it refines these keypoints by eliminating any low-contrast keypoints or edge keypoints. After computing the gradient magnitude and direction in the neighborhood of the keypoints, an orientation is chosen from the histogram for each keypoint to achieve the invariance of image rotation. The keypoint descriptor is a vector computed from the orientation histogram. SIFT is reliable, but can be quite slow and consumes large memory space.

Bay, Tuytelaars, and Gool, developed Speeded Up Robust Features (SURF) [17] by improving each step of SIFT. SURF increases the speed of SIFT threefold and performs almost as well. SURF is good at handling images with blurring and rotation, but not good at handling viewpoint and illumination changes. Both SIFT and SURF use a vector of hundreds

of floating point numbers as the descriptor. The cost of memory space increases significantly with the incensement of features, and therefore neither is suitable for processing large images.

In 2011, Rublee et al. proposed a new feature matching method called ORB (Oriented FAST and rotated BRIEF) [21]. ORB is developed based on the FAST feature detector and BRIEF feature descriptor. It adds many modifications to these two algorithms to improve the performance. Experiments show that ORB costs less computation resources and is two orders of magnitude faster than SIFT, while performing just as well in many situations. In addition, unlike SIFT and SURF, ORB is not patented, which gives it a great advantage in open-source projects. It is used in this thesis for the matching of mine symbols. FAST, BRIEF and ORB will be explained in detail in section 3.5.

3 Methodology

3.1 Methodology Overview

In this research, scanned map images are first preprocessed to remove background noise using color segmentation. Two types of matching techniques are then applied on the processed image to extract target map symbols. The first is template matching, which is accomplished in three stages using different templates. The first stage focuses on the entire structure of the target symbol, ensuring there is overall agreement between a location and the template. As will be described below, this test is highly optimized for speed, at the cost of generating many false matches. Following this, each match is subjected to a connected component test (also described below). Locations passing this test are subjected to a second round of template matching, which requires that individual segments or components of the template be present, including gaps of white pixels surrounding the object. After template matching, ORB feature detection is applied to reduce the amount of false positives. Our assumption is that the output of this step will be visually examined to produce the final hits (this step is not discussed in this thesis). Figure 3-1 shows the overall workflow of the proposed method.

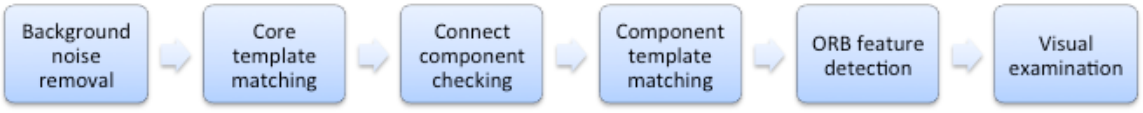


Figure 3-1: Workflow of the proposed method.

3.2 Data Source

The scanned maps are from the USGS Historical Topographic Map Collection (HTMC). They can be downloaded from the USGS website[2] as GeoPDF with georeferencing information. There are around 180,000 maps in total for the US—several thousand for each state. These 1:24,000 maps were typically scanned at 600 dpi. Each of them has around 13000×16000 pixels. Due to the sheer size of the historical maps and the high resolution of the scanned image, it is not practical to perform manual on-screen digitizing.

The target point objects are mine symbols shown in Figure 3-2. There can be zero to several hundred mine symbols in one quad. The bounding box of mine symbols is around twenty-five pixel squares, which means they are very small and disappear if the scanned image is down-sampled. Therefore, the extraction must work at full image resolution.

Mine symbols may differ in appearance in different maps (Figure 3-3). For gravel pit and quarry, many of the symbols are hand-drawn and distorted. Prospect pits may contain a small circle in the center in some maps. They may also have small variation in scale. Mine shafts have a different version that is smaller and has an opposite orientation. The other version is called “mine shaft alt” in the thesis. Adits have various orientations and may contain lines of different thickness.

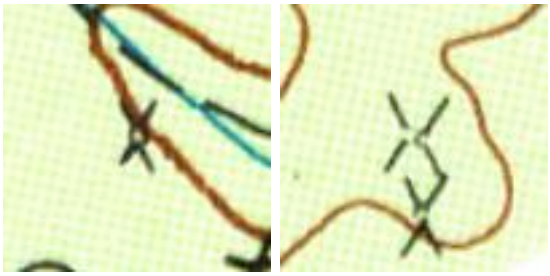
[2] http://www.usgs.gov/

Gravel Pit Prospect Pit Mine Shaft Adit Quarry

Figure 3-2: Mine symbols in map legend.

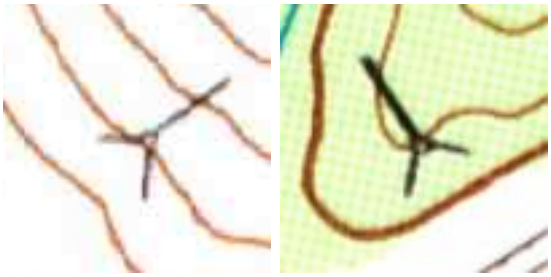
Gravel Pit

Prospect Pit



Mine Shaft/Mine Shaft Alt

Adit



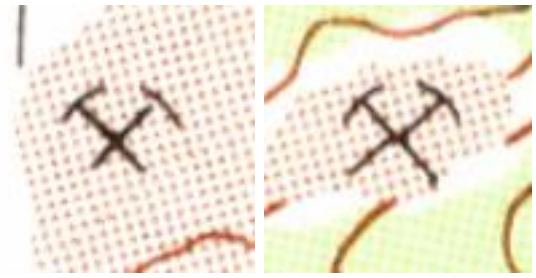
Quarry 

Figure 3-3: Variation of mine symbols on maps.

The USGS group created a database of more than 7000 known locations in 61 Nevada quads. These locations were extracted from the scanned maps by hand digitizing. The data came from both provisional and final map editions. We selected seven quads from the database for our experiments. Table 3-1 shows the number of each mine symbol in the selected quads.

Table 3-1: Number of symbols in the selected quads.

	Adit	Gravel Pit	Mine Shaft	Mine Shaft Alt	Prospect Pit	Quarry
Black Point	0	1	1	0	5	1
Carvers NW	7	4	3	0	17	0
Ellsworth	32	0	30	0	160	6
Gabbs	15	0	21	0	62	4
Granny Goose Well	2	4	9	0	95	1
Silverado Mountain	28	6	0	14	29	4
Treasure Hill	60	0	103	0	8	7
Total number of symbols	144	15	167	14	376	23

Some symbols in the database were misclassified or poorly digitized. In order to adjust the data, we created a program to check the known locations visually. Figure 3-4 shows the program's interface. It opens the symbol database, extracts images from quadrangles and lets the user drag them around to adjust their central coordinates. The program saves the adjusted

coordinates and will save symbol images if desired. We used the program to update all the symbols in the quads we have been using so that the database coordinate is near the center of the symbol.

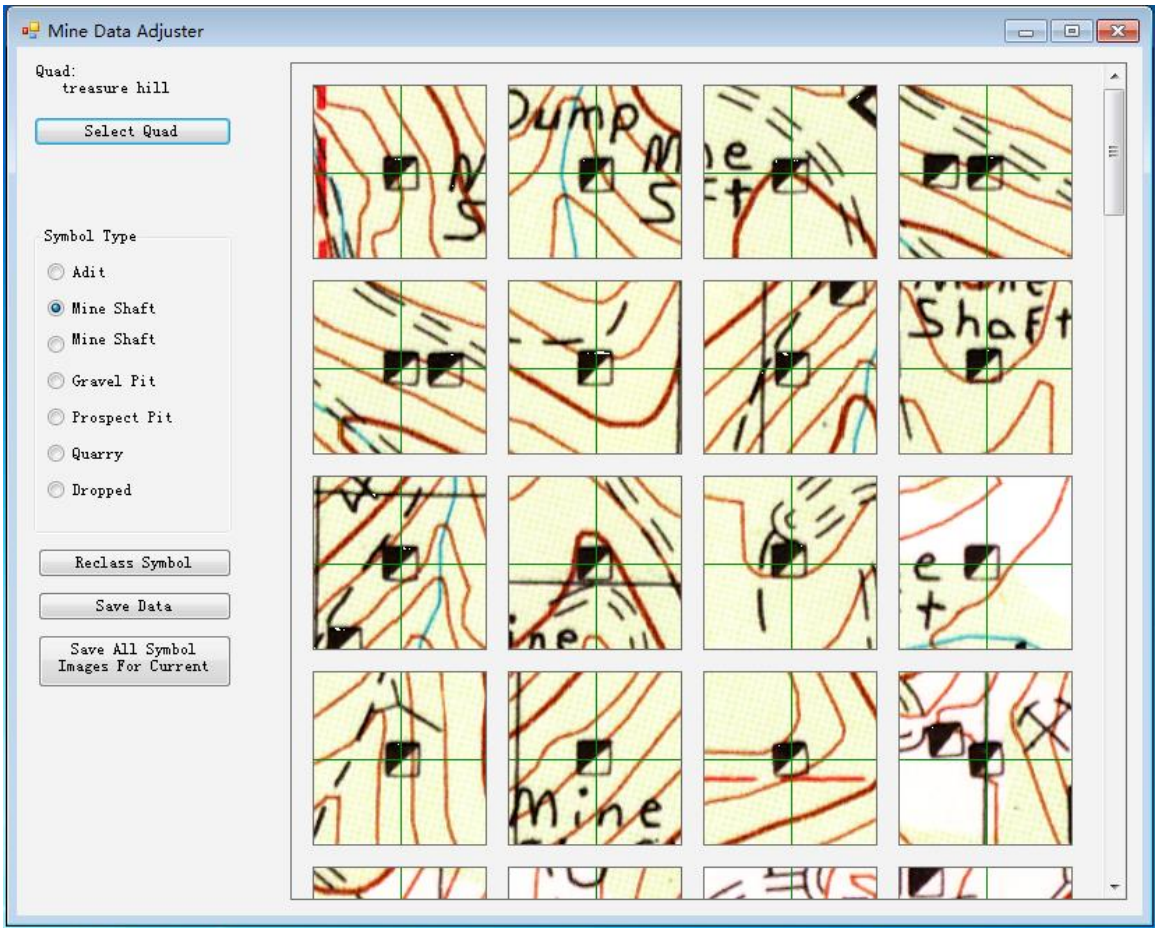


Figure 3-4: Interface of the mine data adjuster.

3.3 Background Noise Removal

3.3.1 Background Noise

Mine symbols often intersect with many other map elements such as contours and text, which significantly increases the difficulty of symbol extraction. For example, if we apply

template matching on the original map to search for prospect pits, we may get many false matches (Figure 3-5). These false matches are very similar to the symbol pattern. Therefore, it is not practical to directly extract symbols from original images: a preprocess of background noise removal should be applied first.

(a) (b) (c)

Figure 3-5: False positive output of template matching: (a) Prospect pit (b) False match (c) False positive and its surrounding area.

In this research, the focus of noise removal is on those objects in a different color than mine symbols. Most of the intersecting objects, like contours, are comprised of non-black dark pixels. Thus, they can be easily removed by the color segmentation technique. For other black objects like text, even though they can be separated by some sophisticated methods [1], the cost is too high compared to color segmentation or even the following extraction steps. Therefore, these objects are left with the target symbols on preprocessed maps.

3.3.2 Color Segmentation

The original map images are in the RGB color space, but the segmentation method is more easily understood using HSV. Therefore, the maps are first transformed to HSV space.

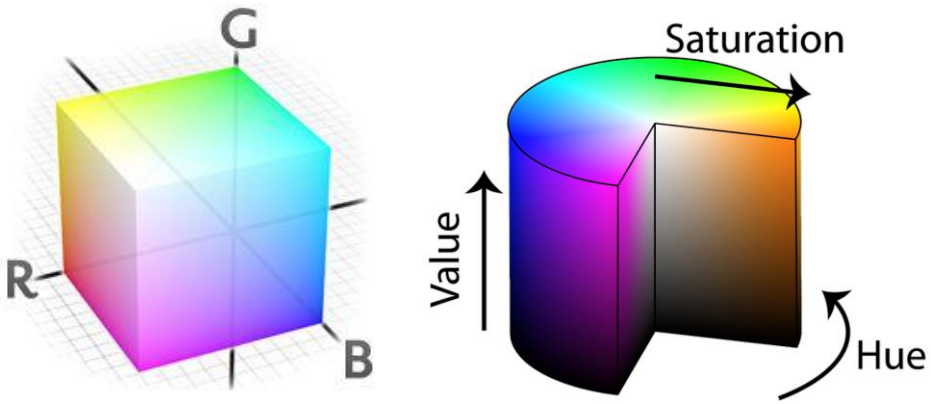


Figure 3-6: RGB color space and HSV color space[3].

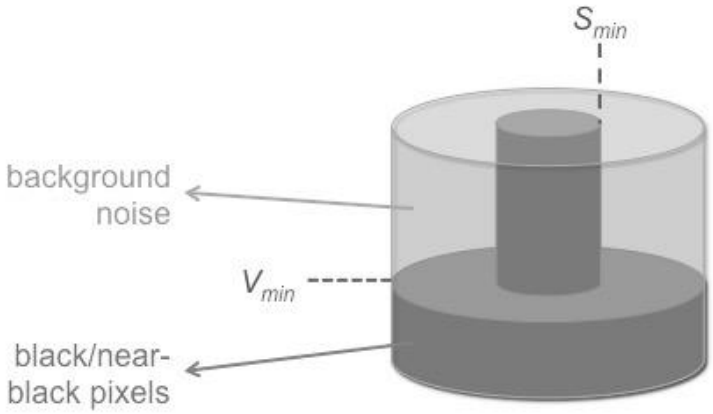


Figure 3-7: Ideal color segmentation model.

After the transformation, color segmentation was performed on HSV color space by selecting a target range of color values. A non-black pixel should have a high value of Saturation or Value. Hue only affects visual sensation of the color, like red or green. If we only look for the non-black pixel of any color, the value of H is irrelevant. Therefore, if non-

[3] https://en.wikipedia.org/wiki/HSL_and_HSV

black is the only criterion, we can use the model in Figure 3-7 to separate background noise and symbols.

The real situation is more complicated than this. Many pixels are located in the overlapping area between symbols and similarly dark background noise objects like contours. Affected by the blending of colors caused by scanning, they are lighter than regular symbol pixels, and have close value of S and V to many background noise pixels. If the thresholds of S and V were set low enough to remove all noise pixels, these pixels in overlapping areas would also be removed, which results in broken symbols (Figure 3-8). Therefore, it is not practical to remove background noise simply by setting the threshold values of S and V. Since most of these overlapping pixels are created by the intersection of symbols and contour lines, one solution to this problem is to deal with contours and other kinds of background noise separately.

In HSV space, Hue ranges from -180 to 180 degrees, Saturation and Value range from 0 to 1. Contours are in dark red or brown and therefore have hue from -5 to 65 degrees. In our program, if a pixel has a V above 0.8, it is set to white. If the pixel has H in the range [-5,65], and both S and V above 0.4, it is also set to white. All other pixels were set to black, and thus the original image was reduced to a two-color binary raster. Figure 3-9 shows the comparison between the original image and the new image after background noise removal.

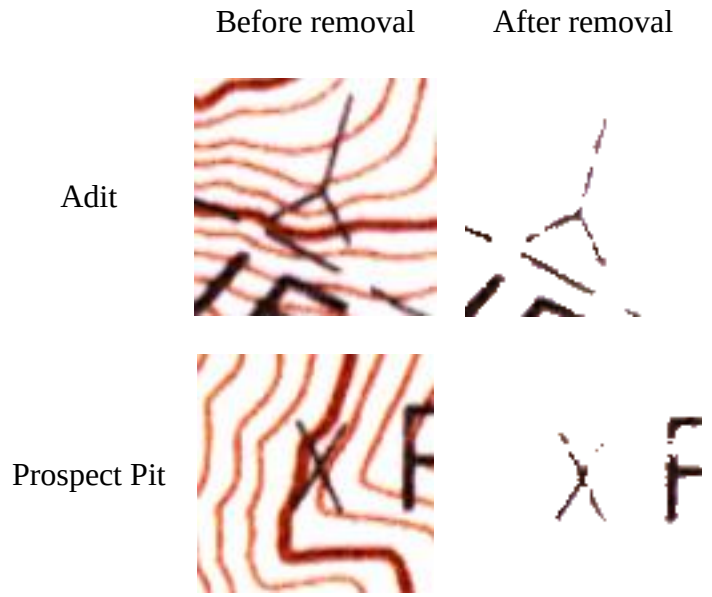


Figure 3-8: Broken symbols caused by removal of overlapping pixels.

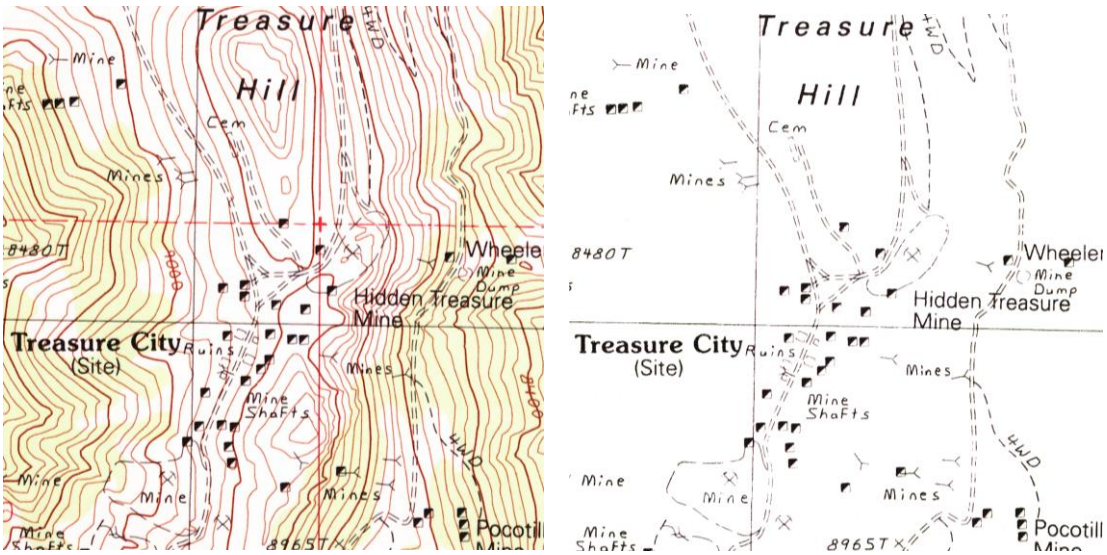


Figure 3-9: Original map and map after background noise removal.

3.4 Template Matching

3.4.1 Template Matching

Template matching is a technique in digital image processing for finding small parts of an image that match a template image [11]. As an important low-level image processing technique, template matching can be utilized in many applications, including face recognition, character recognition, digital watermark and medical image processing.

In this application, a template *T* consists of a list of two-dimensional coordinates and the associated pixel colors. In this application, template pixels are either black or white. The central pixel of the template is given a coordinate of (0,0). Other coordinates are pixel offsets from the central pixel. Thus if an object has a black pixel, two pixels to the right, and one pixel down from the central pixel, the list will contain an entry for coordinate (2,1) with the color black. Template matching amounts to placing the template at some location in the image and comparing surrounding image pixels with the colors in the template list. Only image pixels whose offsets match those in the list are compared. Because both the image and the template pixels are either black or white, we need only to count the number of pixels that agree in order to obtain a measure of deviation. Equivalently, we can report the fraction of template pixels that do not match.

Denote $I(u,v)$ is the color at image coordinate (u,v), which we can take as zero (black) or unity (white). Let N be the number of template pixels and $T_k(c)$ be the color of template pixel k, also zero or unity as mentioned above. Using $T_k(\quad u)$ and $T_k(\quad u)$ as the offsets for pixel k, the fraction of disagreements is:

$$e(u,v) = \sum_{k=1}^{N} \left| T_k(c) - I[u + T_k(\Delta u), v + T_k(\Delta v)] \right| / N$$

The numerator counts the number of disagreements, and accordingly is a number between zero and unity reflecting the proportion of template pixels that do *not* match the image. Obviously, low values signify close agreement. In addition to $e(u,v)$, we will also report the agreement $A(u,v) = 1 - e(u,v)$. This value is also a relative measure, but a value of unity indicates a perfect match. If A is zero, an image location has no pixels in common with the template.

To be labeled a match (or "hit"), a pixel's agreement value must exceed some defined threshold, t_A. The threshold value is a tuning parameter; the results section of the thesis discusses how the value was selected. In testing individual locations, we implemented two obvious optimizations. First, to avoid subtraction, we tested for disagreement between template and image pixels and incremented a running count accordingly. Note that if the count reaches $N(1 - t_A)$, a location cannot possibly be a hit, even if all remaining pixels agree with the template. The second optimization was to abandon the evaluation of $A(u,v)$ whenever the count exceeded the largest admissible value. That is, the summation above was terminated when the disagreement count reached the largest value possible for a given t_A.

3.4.2 First Pass Template Matching

A mine symbol may have more than 400 non-white pixels. Comparing all of them is very time-consuming. Therefore, it is not practical to use all non-white pixels to construct the template. The template should capture the key structure of the symbols with as few pixels as possible.

To ensure the preservation of all key structures, the template construction is divided into two steps. A bounding box is placed around the template, and every other row and column in

the box is examined. If the template contains that row and column, the template pixel is included. This ensures that the full extent of the template is included and at the same time reduces the number of pixel templates used. Additional pixels are added by randomly sampling the original list until a specified minimum number of pixels are included with no duplicates. The second step adds pixels in proportion to their density in the template, and so ensures large blocks of template pixels are properly represented.

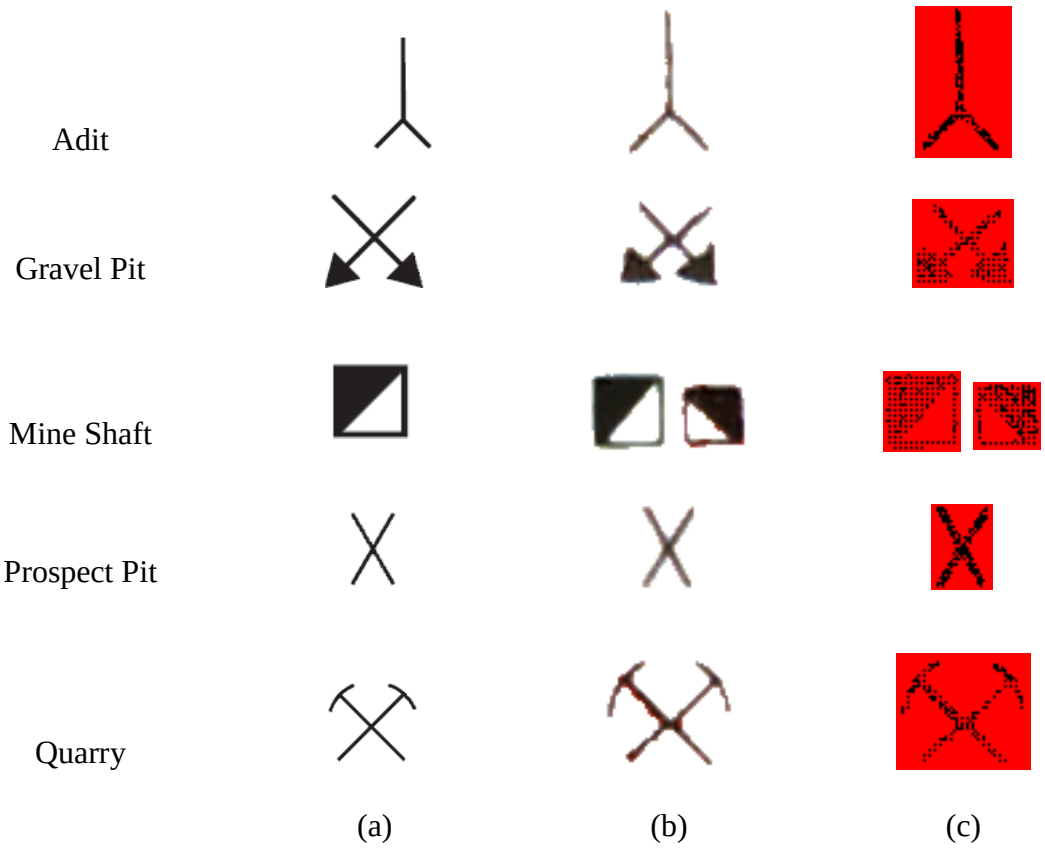


Figure 3-10: Comparison of symbols from (a) map legend, (b) the scanned map and (c) the first-pass templates.

Figure 3-10 shows the map symbols and the corresponding templates. Templates were taken from actual scans in order to match with objects in map image files. Adits require special treatment because they can appear on a map in any orientation. To find adits regardless of angle, we prepared multiple templates by rotating the "vertical" template shown in 3-10. The number of templates (rotation angles) is variable. Too large a number will slow the method unnecessarily, whereas too few will cause false negatives as true objects at some angle are missed.

Notice from Figure 3-10 that the template sampling process preserves the full extent of the template, which is very important considering their small size. At the same time, sampling reduces the computational cost by $1 - N'/N$ where N' is the number of pixels retained. Two other optimizations are possible in the first pass. First, it may be that not every pixel in the image needs to be examined. Because even the linear components of mine symbols are several pixels wide, it could be that examining every other or even every third image pixel will be sufficient. So long as each template match is done at full image resolution and A_t is small enough, it could be that no additional objects are missed beyond what would be missed by evaluating every image location. In effect this amounts to sampling the original image for possible objects. A sampling rate of 1/3rd would reduce search time by a factor of nine. A 25% sampling rate would reduce it by a factor of four. Experiments are needed to determine the appropriate sampling rate. A second optimization is to store image segments in memory as linear arrays rather than two-dimensional arrays. There is some cost to linearizing the image bitmap, but at least in principle that cost might be more than recovered by the advantage of working with a single subscript. The results section presents results for both of these optimizations.

3.4.3 Connected Component Test

The connected component checking is applied after the first stage template matching. The goal of this step is to remove first-pass hits that are too full of pixels to be the target object.

Figure 3-11 shows one example of first-pass hits that can be removed by the component checking. The area in the rectangle has a high agreement (0.853) with the first-pass template of a gravel pit. However, the largest connected component in the bounding of the object contains 755 connect pixels, which is far more than the regular amount of pixels contained by a gravel pit (around 430). Therefore, it is rejected as a false positive.

Figure 3-11: First-pass false positive that can be removed by component checking.

For each first-pass hit, a box is placed around the center pixel. The program finds all connected components in that box. The size of the box is a tradeoff between too little discrimination if it is too small (in the limit it contains one pixel and offers zero information) and too large so that it rules out hits that happen to touch a long black line or piece of text (Figure 3-11). In the program, we used the template's minimum bounding box as the test box. For each first pass hit, the program finds the largest connected component *C* in the bounding

box. Then it computes the ratio r of the number of connected pixels in C to the number of pixels in the mine symbols. For the given CC lower bound r_1 and upper bound r_2, if $r < r_1$ or $r > r_2$, then the hit is rejected. The procedure only considers the largest connected component in the box because the issue is whether the hit is part of a large number of connected pixels. The total number of connected pixels is secondary, as is the number of connected components. The threshold of connected component test for hit retention will be discussed in the results section.

(a) (b)

Figure 3-12: (a) Mine shaft crossed by a black line (b) Prospect pit connect to a piece of text.

3.4.4 Symbol Segment Matching

The first-pass template focuses on the overall structure of symbols. A location must have a reasonably high average agreement with the template to be labeled as a hit, but individual parts of the object can be missing. As mentioned above, the first pass is designed to capture all map objects rapidly, admittedly at the cost of many false positives. Many of the false positives can be removed by checking individual segments of the template and requiring that all segments be present to some within some level of agreement. Segments are tested sequentially, so the number of hits drops from one pass to another. Segments include both

black pixels of the symbol itself and immediately surrounding areas that would be white in the ideal situation of no confounding noise or overlap with other map information (Figure 3-13).

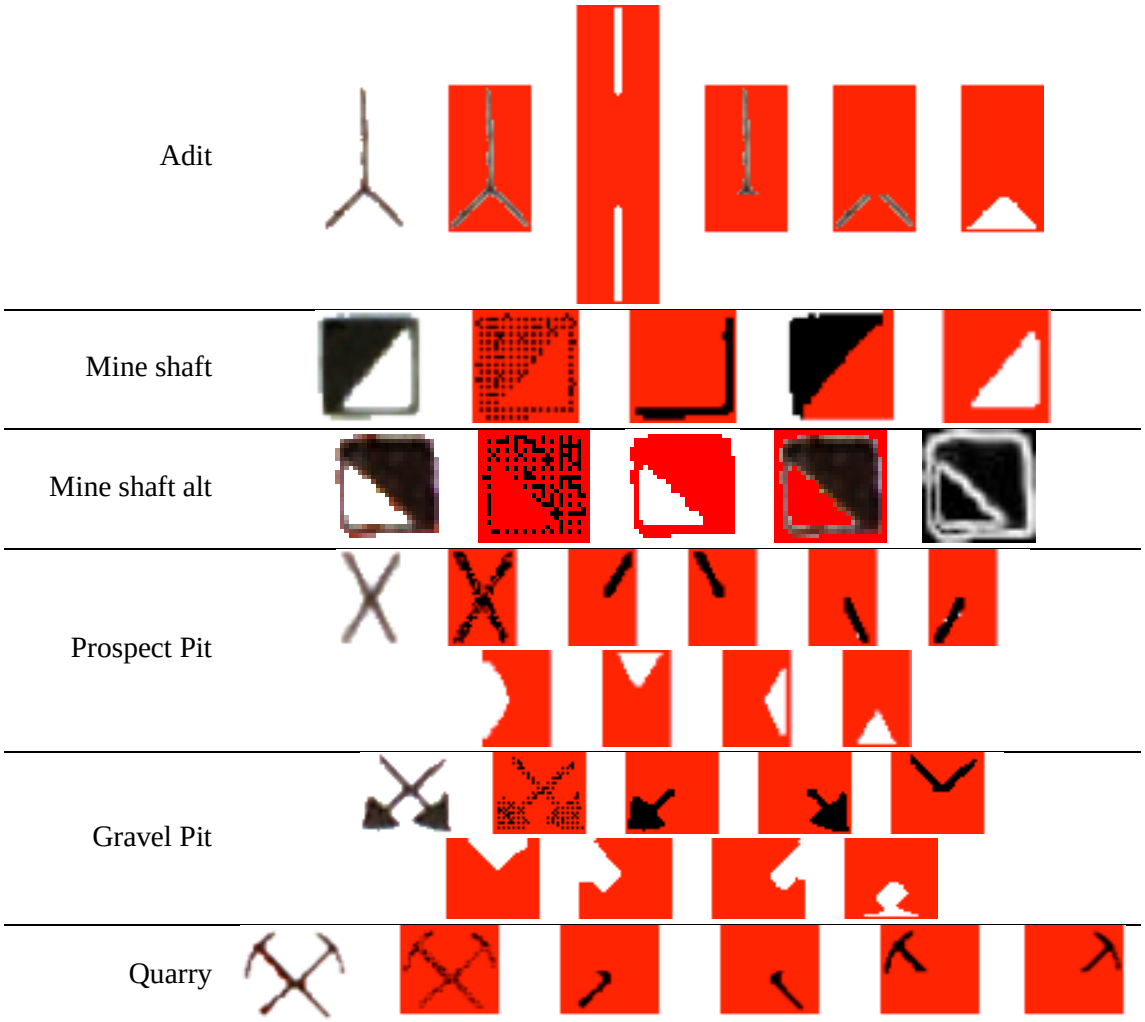
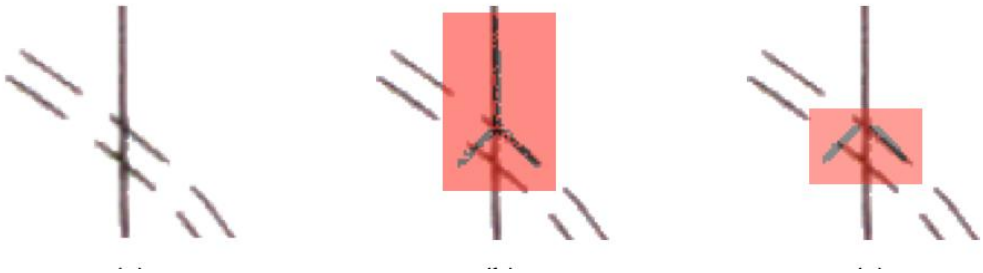


Figure 3-13: Template collection of different segments of the mine symbols.

Figure 3-14 (a) shows a false positive output of adit from the first pass. The image has high agreement (0.793) with the template used in the first pass (b). This is because the

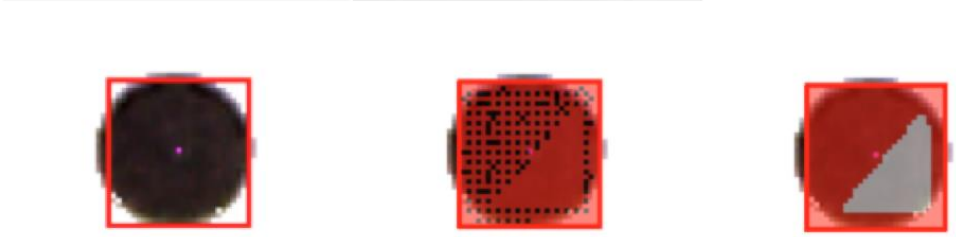
crossed black lines form a shape very similar to adit and therefore have a high overall match. However, if we check the two short lines of adit separately (c), we can see that one of them is a match, but the other one has nearly no overlapping pixels with the underlying image. This false hit can then be removed based on that second component test.

Figure 3-15 (a) shows a false positive output of mine shafts from the first pass. The image has high agreement (A = 0.907) with the template used in the first pass (b). However, if we use the white triangle area of the mine shaft as the template, the agreement is near 0 (c) and this false hit can be easily removed.



(a) (b) (c)

Figure 3-14: (a) False positive of adit from the first pass (b) Overlap with the first pass's template (c) Overlap with one of second pass's templates.



(a) (b) (c)

Figure 3-15: (a) False positive of mine shaft from the first pass (b) Overlap with the first pass's template (c) Overlap with one of second pass's templates.

In this step, location agreement $A(u,v)$ is calculated in the same way as for the first pass, but there are two significant considerations unique to the subsequent passes. First, because the template components have so few pixels, they are not sampled on the first pass—we use all of the template pixels in calculating $A(u,v)$. Second, we attempt to allow for variation in symbol appearance by using somewhat looser positional requirements for components. A completely strict test would center each component on the hit location and measure $A(u,v)$ there and only there. Instead, we place the component at the central location and at the eight surrounding pixels. The largest agreement value for those nine pixels is taken as $A(u,v)$. In this way, a location can pass a test even if its components are not completely consistent with one another as would occur if the symbol were perfectly drawn. Experimentation showed that this procedure strikes a balance between ensuring all components are present and not requiring unreasonable fidelity to the template.

3.5 ORB Feature Matching

In computer vision, features are unique image structures that can be applied to identify objects like mine symbols. An object may contain many features, including edges, corners, blobs or ridges. By matching these features, target objects can be recognized in different images.

Typically, feature matching between images has three steps:

1. *Keypoint* detection: detect a set of distinctive *keypoints* as the center of features in both images.
2. Feature description: for each keypoint, compute a *descriptor* from its surrounding pixels to encode the feature structure.
3. Feature matching: matching is performed by comparing feature descriptors using some sort of similarity measurement.

Based on the number of matching keypoints, we can determine whether there exists a match (with the target object) between these two images or not.

In this research, we use ORB (Oriented FAST and rotate BRIEF) for the matching of mine symbols. The details of ORB are discussed in the following sections.

3.5.1 Keypoint Detection

The ORB keypoint detection algorithm is based on FAST (Features from Accelerated Segment Test). ORB adds Harris corner measure to the keypoints detected by FAST, and uses intensity centroid to measure the orientation of keypoints.

1) FAST

FAST was originally proposed by Rostern and Drummond in 2006 [18] and later revised in 2010 [19]. It was used to detect corners in images. Before FAST, there were several well-established corner detectors like Harris and SUSAN. However, none of them were fast enough for real-time applications. FAST was then proposed as a solution to this problem.

FAST must be performed on grayscale images. It determines whether an image pixel is a keypoint or not by checking the pixels on its surrounding circle (Figure 3-16). For a pixel p in the image, let its intensity be I_p. Select an appropriate threshold intensity value t. Consider

a circle of sixteen pixels around the pixel *p*. If there exists *n* (typically set as twelve) contiguous pixels in the circle which are all brighter than $I_p + t$, or all darker than $I_p - t$, then p is a keypoint of the corner. To make the selection of keypoints faster, the algorithm first examines the four pixels at 1, 9, 5 and 13 (see Figure 3-16). Suppose $n \geq 12$, then at least three of these four pixels should satisfy the threshold criterion above so that the keypoint will exist. This test can be used to exclude a large number of keypoint candidates in a short time.

There are several limitations to this algorithm. One problem is that it cannot reject as many keypoint candidates when $n < 12$. The speed of the algorithm is also largely affected by the order in which the sixteen pixels are queried. In addition to the speed issue, it can detect multiple keypoints adjacent to one another for the same corner.

To address the speed issue, a machine learning approach has been added to the original algorithm. The approach first selects a set of images for training and runs FAST in every image to detect the keypoints. For every keypoint, it stores sixteen pixels around it as a vector *P*. Each pixel *x* in these sixteen pixels can have one of the following three states:

$$S_{p\to x} = \begin{cases} d, & I_{p\to x} \leq I_p - t \quad \text{(darker)} \\ s, & I_p - t < I_{p\to x} < I_p - t \quad \text{(similiar)} \\ b, & I_p + t \leq I_{p\to x} \quad \text{(brighter)} \end{cases}$$

Depending on these states, the feature vector *P* is subdivided into three subsets, P_d, P_s, P_b. Define a new Boolean variable, K_p, which is true if *p* is a keypoint and false otherwise. Build a decision tree classifier to query each subset using the variable K_p for the knowledge about the true class. In other words, select the pixel *x* that has the most information about the center pixel *p*. The decision tree is used for fast detection in other images.

The problem of adjacent keypoints can be solved by using Non-maximum Suppression. It first computes a score function *V* for each of the detected keypoints. *V* is the sum of absolute difference between the center pixel and sixteen surrounding pixels values. For two adjacent keypoints, compare their *V* values and discard the one with the lower *V* value.

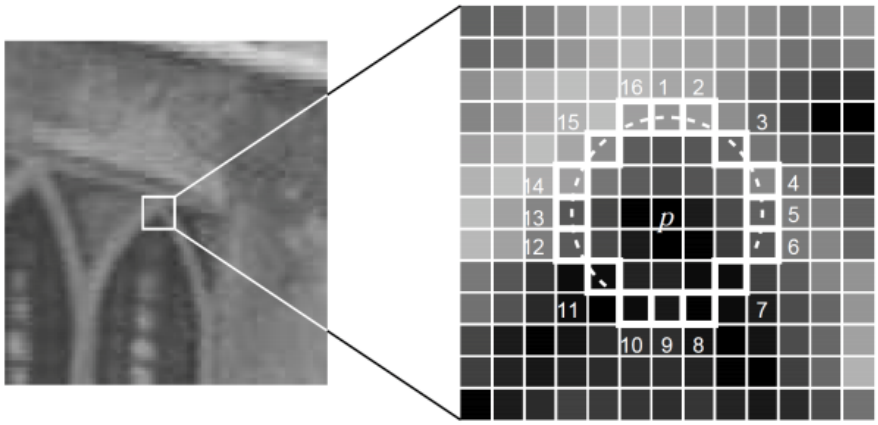


Figure 3-16: Keypoint detection by FAST (image from [18]). Pixel *p* is a candidate keypoint. The squares with bold outlines are the pixels on the circle. The dashed line indicates twelve contiguous pixels that are brighter than *p*.

2) Harris corner measure

FAST does not provide a measurement of keypoints. To address this issue, ORB applies Harris corner measure [24] to the keypoints detected by FAST. The keypoints can then be ordered based on their Harris measurement.

Harris corner measure determines whether a region can contain a corner or not. For a grayscale two-dimensional image, take an image patch over the region centered at point *p* in

location (u, v) and shift it. Let I_x and I_y be the image derivatives in x and y direction respectively. We can get the Harris matrix *M*:

$$M = \sum_{u,v} w(u,v) \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix}$$

Let λ_1 and λ_2 be the eigenvalues of *M*. We can decide whether a region is corner, edge or flat according to the value of λ_1 and λ_2:

- If $\lambda_1 \approx 0$ and $\lambda_2 \approx 0$, then this region has no feature of interest.
- If $\lambda_1 \gg \lambda_2$ or $\lambda_2 \gg \lambda_1$, then this region contains an edge.
- If both λ_1 and λ_2 are large, then this region contains a corner.

3) Intensity centroid

FAST is not rotation invariant. As a modification to this, ORB uses *intensity centroid* [25] to measure of the orientation of keypoints. For each keypoint, find the intensity centroid of its surrounding circular region of radius *r*. The orientation of the keypoint can then be defined as the direction from the keypoint to the intensity centroid. This approach performs better than other gradient-based measures under large image noise [21].

3.5.2 Feature Description

Feature descriptor encodes the feature region surrounding the keypoints. ORB feature detection is based on Binary Robust Independent Elementary Features (BRIEF). It also introduces a learning step to improve the performance of BRIEF under rotation.

BRIEF [20] uses binary strings as the descriptors of keypoints. It first smoothes the entire image. In a smoothed image patch *ip* around the center keypoint, select a set of location pairs (*x*,*y*) in the surrounding area of *g*. Define a binary test τ:

$$\tau(ip; x, y) \coloneqq \begin{cases} 1 : g(x) < g(y) \\ 0 : g(x) \geq g(y) \end{cases}$$

where $g(x)$ is the intensity of location *x*. The feature descriptor is then defined as a vector of *n* binary tests:

$$f_n(ip) \coloneqq \sum_{1 \leq i \leq n} 2^{i-1} \tau(ip; x_i, y_i)$$

ORB uses a Gaussian distribution [20] of locations pairs around the center keypoint. The vector length is set as 128.

BRIEF performs poorly with rotation. To address this issue, ORB "steers" BRIEF according to the orientation of keypoints. For a descriptor of *n* binary tests, define a matrix *S* with coordinates of (x_i, y_i):

$$S = \begin{pmatrix} x_1, \ldots, x_n \\ y_1, \ldots, y_n \end{pmatrix}$$

Construct a "steered" version S_θ from S using the rotation matrix R_θ:

$$S_q = R_q S$$

Then the steered BRIEF descriptor is:

$$g_n(ip, \theta) \coloneqq f_n(ip) | (x_i, y_i) \in S_\theta$$

ORB discretizes the angle to increments of twelve degrees, and constructs a lookup table of pre-computed BRIEF patterns. The correct set of location pairs S_θ can be found in the

lookup table as long as the keypoint orientation is consistent across views. S_θ will then be used to compute the binary descriptor.

3.5.3 Feature Matching

A Brute-Force approach is applied for the matching of features between two images. For each feature in one image, Brute-Force matcher computes its deviation (distance) from all features in the other image using their descriptors. It then returns the feature with the closest distance as the best match.

We use Hamming distance as the distance measurement between ORB descriptors. Hamming distance counts the number of positions at which the corresponding symbols are different between two strings of equal length. For example, the Hamming distance between “1001” and “1111” is two. ORB uses binary strings as feature descriptors. For binary strings *a* and *b*, the Hamming distance is equal to the number of ones in *a* XOR *b*. This computation can be completed very quickly for CPUs.

Crosschecking is applied to the matching results to improve the consistency. In some cases, features with the closest distance may not be true matches. This may happen due to image noise or distortion. To address this problem, crosschecking accepts a match only when two features match each other. That is, if feature *i* in image *A* has feature *j* in image *B* as the best match, then *j* should also have *i* as the best match.

3.5.4 Mine Symbol Detection and Matching

For each mine symbol, a 50×50 pixels square image centered at the symbol is extracted from the scanned maps as the symbol image. The symbol images are larger than the mine

symbols because the ORB feature detector requires a detecting circle with radius of about 10 pixels around the center keypoint. If the symbol image is too small, some keypoints at the edge of the mine symbols may not be detected. For each template-matching hit, a hit image of the same size is extracted from the map centered on the hit coordinates.

ORB only works with grayscale images. Therefore, all images are converted to grayscale first. Then ORB is performed on both symbol and hit images to detect feature keypoints and compute their descriptors. A hit would be dropped if no keypoint was detected in its hit image. After the detection, the Brute-Force matcher compares features in the symbol images with features in each corresponding hit image. For each hit image, compute the ratio of matching keypoints to the total number of keypoints in the corresponding symbol image as the ORB agreement of the hit. If the ORB agreement were lower than some threshold, the hit would be rejected as a false positive. According to our experiments, different agreement thresholds should be applied for different mine symbols. This is discussed in the results section.

Figure 3-17 presents two examples of matching images. The best N (different from the dividing threshold) keypoint matches (ten for gravel pit, fifteen for quarry) are shown with linked lines.

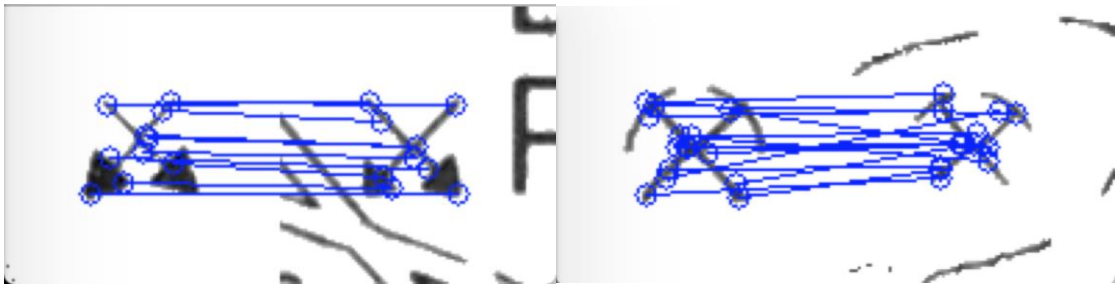


Figure 3-17: Examples of matches for gravel pit and quarry.

3.6 Implementation

A C# program was written for background noise removal, template matching and connect component checking. For map image I/O, we used the open-source library Geospatial Data Abstraction Library (GDAL[4]).

The program takes scanned topographic maps as input. It first removes noise pixels in map images and outputs new images. To make this process faster, only pixels within the quadrangle proper are considered.

After the removal, the three-stage template matching is performed on the new images. Because there are many false matches along quad neatlines, the program only considers pixels that are five pixels or more away from a neatline, which also makes the procedure faster.

Next, ORB is performed on the hits that pass the template matching. For ORB feature detection, we used the implementation available in OPENCV[5], which is an open-source computer vision library written in C++. Bindings to Python are available, which greatly simplified our use. A Python script was written and then converted to an executable Windows program using py2exe[6]. The program can run without Python installation, and it can be invoked by the main program using a standard system call.

---

[4] http://www.gdal.org/

[5] http://opencv.org/

[6] http://www.py2exe.org/

All tuning constants, like the thresholds in template matching, can be easily changed in the program. All final hits can be exported to a CSV file. If requested, the program extracts a small portion of the scanned image surrounding all hits.

4 Results and Discussion

4.1 Template Matching

Table 4-1 shows the statistics of template matching hits after each pass with different error thresholds. The first pass uses the template of the whole symbol. The second and following passes use templates on different segments of the symbols. The number of passes may vary for different symbols due to different segmentation. Error thresholds decrease from 0.7 to 0.3 with an interval of 0.1 in the test. Each location is considered a hit if its mismatch rate is lower than the error threshold, otherwise it is rejected. The table also gives the count and percentage of false negatives in each pass.

The threshold of each pass should be set to retain as few positives as possible while keeping the false negative rate close to 0. We can see from the table that the "best" threshold is 0.3 for the two types of mine shafts, 0.5 for gravel pit/adit/prospect pit and 0.6 for quarry in all passes. With these settings, the false negative rate for all symbols can be kept lower than 0.5%.

Figure 4-1 shows the percentage and count of hits retained at each pass with the "best" thresholds. We can see that the performance of template matching varies in different passes and from one symbol to another. For adit, the second pass significantly reduces the amount of hits, while the following passes achieve very little. For gravel pit, passes 2-4 remove around 20% of first-pass hits, and the following four passes only remove around 10%. For mine shaft and mine shaft alt, all passes on individual segments of the symbols do not work very well, except for the final pass of mine shaft. For prospect pit, passes 2-5 remove 30% of first

pass hits, while the following four passes only remove 10%. For quarry, all passes work well except for the third one.

Table 4-1: Statistical analysis of hits after each pass of template matching

Symbol	Pass	Error threshold	Hits	Rejection rate	False negative	False negative rate
Gravel pit	1	0.7	10542	0.0	0	0.0
		0.6	8701	17.5	0	0.0
		0.5	5033	52.3	0	0.0
		0.4	2362	77.6	1	6.7
		0.3	876	91.7	2	13.3
	2	0.7	9441	0.0	0	0.0
		0.6	8059	14.6	0	0.0
		0.5	4915	47.9	0	0.0
		0.4	2340	75.2	1	6.7
		0.3	876	90.7	2	13.3
	3	0.7	7523	0.0	0	0.0
		0.6	6943	7.7	0	0.0
		0.5	4587	39.0	0	0.0
		0.4	2290	69.6	1	6.7
		0.3	876	88.4	2	13.3
	4	0.7	5881	0.0	0	0.0
		0.6	5597	4.8	0	0.0
		0.5	4000	32.0	0	0.0
		0.4	2082	64.6	1	6.7
		0.3	842	85.7	2	13.3
	5	0.7	5784	0.0	0	0.0
		0.6	5500	4.9	0	0.0
		0.5	3908	32.4	0	0.0
		0.4	1996	65.5	1	6.7
		0.3	776	86.6	2	13.3
	6	0.7	5635	0.0	0	0.0
		0.6	5352	5.0	0	0.0
		0.5	3771	33.1	0	0.0
		0.4	1882	66.6	1	6.7
		0.3	701	87.6	2	13.3
	7	0.7	5461	0.0	0	0.0
		0.6	5178	5.2	0	0.0
		0.5	3604	34.0	0	0.0
		0.4	1726	68.4	1	6.7
		0.3	597	89.1	2	13.3
	8	0.7	5269	0.0	0	0.0
		0.6	4986	5.4	0	0.0
		0.5	3417	35.1	0	0.0
		0.4	1557	70.4	1	6.7
		0.3	477	90.9	2	13.3
Mine shaft	1	0.7	5366	0.0	0	0.0
		0.6	5297	1.3	0	0.0
		0.5	4792	10.7	0	0.0
		0.4	2977	44.5	0	0.0
		0.3	1460	72.8	0	0.0
	2	0.7	4949	0.0	0	0.0
		0.6	4881	1.4	0	0.0

Symbol	Pass	Error threshold	Hits	Rejection rate	False negative	False negative rate
		0.5	4424	10.6	0	0.0
		0.4	2817	43.1	0	0.0
		0.3	1434	71.0	0	0.0
		0.7	4946	0.0	0	0.0
		0.6	4881	1.3	0	0.0
	3	0.5	4424	10.6	0	0.0
		0.4	2817	43.0	0	0.0
		0.3	1434	71.0	0	0.0
		0.7	4687	0.0	0	0.0
		0.6	4632	1.2	0	0.0
	4	0.5	4193	10.5	0	0.0
		0.4	2594	44.7	0	0.0
		0.3	1239	73.6	0	0.0
Mine shaft alt		0.7	10059	0.0	0	0.0
		0.6	9479	5.8	0	0.0
	1	0.5	8339	17.1	0	0.0
		0.4	5611	44.2	0	0.0
		0.3	2637	73.8	0	0.0
		0.7	9752	0.0	0	0.0
		0.6	9175	5.9	0	0.0
	2	0.5	8059	17.4	0	0.0
		0.4	5360	45.0	0	0.0
		0.3	2408	75.3	0	0.0
		0.7	9752	0.0	0	0.0
		0.6	9175	5.9	0	0.0
	3	0.5	8059	17.4	0	0.0
		0.4	5360	45.0	0	0.0
		0.3	2408	75.3	0	0.0
		0.7	9744	0.0	0	0.0
		0.6	9167	5.9	0	0.0
	4	0.5	8051	17.4	0	0.0
		0.4	5352	45.1	0	0.0
		0.3	2400	75.4	0	0.0
Prospect pit		0.7	70277	0.0	0	0.0
		0.6	35227	49.9	1	0.3
	1	0.5	16554	76.4	2	0.5
		0.4	7431	89.4	14	3.7
		0.3	3107	95.6	61	16.2
		0.7	48921	0.0	0	0.0
		0.6	28534	41.7	0	0.0
	2	0.5	15153	69.0	0	0.0
		0.4	7211	85.3	12	3.2
		0.3	3087	93.7	59	15.7
		0.7	35029	0.0	0	0.0
		0.6	24161	31.0	0	0.0
	3	0.5	14115	59.7	0	0.0
		0.4	7012	80.0	12	3.2
		0.3	3055	91.3	59	15.7
		0.7	26114	0.0	0	0.0
		0.6	20658	20.9	0	0.0
	4	0.5	13288	49.1	0	0.0
		0.4	6886	73.6	12	3.2
		0.3	3049	88.3	59	15.7
		0.7	20425	0.0	0	0.0
		0.6	17253	15.5	0	0.0
	5	0.5	11775	42.4	0	0.0
		0.4	6531	68.0	12	3.2

Symbol	Pass	Error threshold	Hits	Rejection rate	False negative	False negative rate
		0.3	3018	85.2	59	15.7
	6	0.7	20128	0.0	0	0.0
		0.6	16957	15.8	0	0.0
		0.5	11490	42.9	0	0.0
		0.4	6284	68.8	12	3.2
		0.3	2814	86.0	59	15.7
	7	0.7	19346	0.0	0	0.0
		0.6	16185	16.3	0	0.0
		0.5	10750	44.4	0	0.0
		0.4	5615	71.0	12	3.2
		0.3	2268	88.3	59	15.7
	8	0.7	18992	0.0	0	0.0
		0.6	15832	16.6	0	0.0
		0.5	10411	45.2	0	0.0
		0.4	5307	72.1	12	3.2
		0.3	2029	89.3	59	15.7
	9	0.7	18560	0.0	0	0.0
		0.6	15400	17.0	0	0.0
		0.5	9999	46.1	0	0.0
		0.4	4957	73.3	12	3.2
		0.3	1776	90.4	59	15.7
Quarry	1	0.7	21413	0.0	0	0.0
		0.6	10451	51.2	0	0.0
		0.5	4530	78.8	3	13.0
		0.4	1671	92.2	6	26.1
		0.3	580	97.3	10	43.5
	2	0.7	16154	0.0	0	0.0
		0.6	9187	43.1	0	0.0
		0.5	4340	73.1	3	13.0
		0.4	1656	89.7	6	26.1
		0.3	579	96.4	10	43.5
	3	0.7	12318	0.0	0	0.0
		0.6	7795	36.7	0	0.0
		0.5	4020	67.4	3	13.0
		0.4	1614	86.9	6	26.1
		0.3	578	95.3	10	43.5
	4	0.7	10266	0.0	0	0.0
		0.6	7258	29.3	0	0.0
		0.5	3925	61.8	3	13.0
		0.4	1598	84.4	6	26.1
		0.3	577	94.4	10	43.5
	5	0.7	5206	0.0	0	0.0
		0.6	4672	10.3	0	0.0
		0.5	2987	42.6	3	13.0
		0.4	1416	72.8	5	21.7
		0.3	557	89.3	9	39.1
Adit	1	0.7	91588	0.0	0	0.0
		0.6	74247	18.9	0	0.0
		0.5	49590	45.9	0	0.0
		0.4	17274	81.1	5	3.5
		0.3	6579	92.8	26	18.1
	2	0.7	29431	0.0	0	0.0
		0.6	27363	7.0	0	0.0
		0.5	22585	23.3	0	0.0
		0.4	14771	49.8	5	3.5
		0.3	6579	77.6	26	18.1
	3	0.7	28725	0.0	0	0.0

Symbol	Pass	Error threshold	Hits	Rejection rate	False negative	False negative rate
		0.6	26939	6.2	0	0.0
		0.5	22458	21.8	0	0.0
		0.4	14771	48.6	5	3.5
		0.3	6579	77.1	26	18.1
	4	0.7	28725	0.0	0	0.0
		0.6	26939	6.2	0	0.0
		0.5	22458	21.8	0	0.0
		0.4	14771	48.6	5	3.5
		0.3	6579	77.1	26	18.1
	5	0.7	28379	0.0	0	0.0
		0.6	26593	6.3	0	0.0
		0.5	22114	22.1	0	0.0
		0.4	14430	49.2	5	3.5
		0.3	6262	77.9	26	18.1

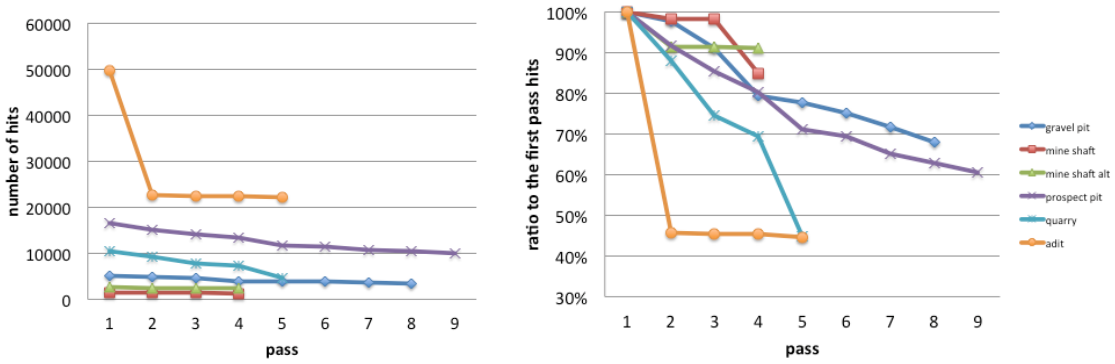


Figure 4-1: Percentage and count of hits retained at each pass with the “best” threshold setting.

4.2 Connected Component Test

We ran Connected Component (CC) test on the first pass template hits (with an error threshold 0.6) to evaluate what would happen if various lower and upper CC thresholds were used. Only one type of CC threshold is applied at a time. The lower CC thresholds range from 0.1 to 0.99, and the upper CC thresholds range from 1.01 to 5. Figures 4-2 and Figure 4-3 show the variation of rejection rate and false negative rate of different CC thresholds.

The selection of CC thresholds should allow the test to reject as many hits as possible while keeping the false negative rate close to 0. Table 4-2 gives the largest possible CC lower bound and lowest possible CC upper bound to keep the false negative rate less than 1%. It also shows the corresponding rejection rate of each symbol. We can see from the table that by applying a lower bound of 0.8 to mine shaft and 0.9 to mine shaft alt, we can remove a significant amount of false positives while keeping all true positives. The upper bound of these two symbols are not very useful. Applying either a lower bound or an upper bound does not accomplish anything for other symbols.

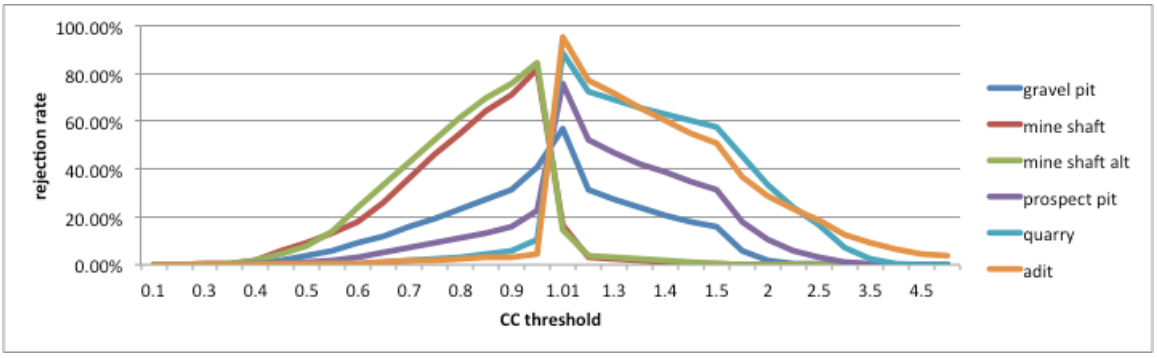


Figure 4-2: Hit rejection rate of different CC thresholds.

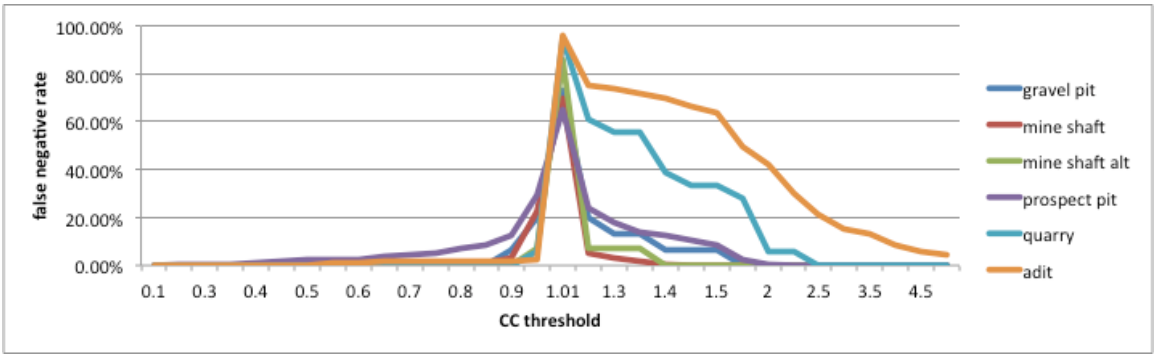


Figure 4-3: False negative rate of different CC thresholds.

Table 4-2: Connect Component thresholds and hit rejection rates

	Gravel pit	**Mine shaft**	**Mine shaft alt**	**Prospect pit**	**Quarry**	**Adit**
CC lower bound	0.85	0.8	0.9	0.35	0.9	0.6
Rejection rate	27.09%	55.19%	76.04%	0.02%	5.72%	0.68%
False negative rate	0%	0%	0%	0.27%	0%	0.79%
CC upper bound	1.75	1.4	1.4	2	2.5	>5
Rejection rate	5.95%	1.18%	1.45%	10.50%	16.65%	None
False negative rate	0%	0%	0%	0.54%	0%	None

4.3 ORB Feature Matching

We ran ORB on the hits of the last round of template matching with different thresholds. If the keypoint match rate for a hit was less than the threshold, that hit would be tossed out as a false positive. The thresholds range from 0.3 to 0.7. Table 4-3 gives the hit rejection rate and false negative rate of each threshold. We can see from the table that if we set the ORB threshold to 0.6 we will not lose any true positives. If we further decrease the threshold, we may start losing hits that should be retained.

In general, ORB works well for gravel pit, quarry and adit. For gravel pit, the largest possible rejection rate without losing true hits is around 25%. For quarry and adit, applying an ORB threshold of 0.5 does not cause any true hits to be lost, while around 60% and 20% of false positives are removed for each pass. If we decrease the threshold to 0.4 for adit, the rejection rate would increase to 30% at the cost of only 0.7% of false negatives.

However, ORB may not work for all symbol types. For mine shaft, mine shaft alt and prospect pit, although ORB does not remove too many true hits even at the error threshold of 0.3, it achieves little in rejecting hits. One possible reason for this is that ORB can detect very few (<10) keypoints in the symbol images for these three symbols.

Table 4-3: Statistical analysis of ORB with different thresholds

Symbol	Error threshold	Rejection rate	False negative	False negative rate
Gravel pit	0.7	0.0	0	0.0
	0.6	24.7	0	0.0
	0.5	56.3	2	13.3
	0.4	85.6	10	66.7
	0.3	98.9	14	93.3
Mine shaft	0.7	0.0	0	0.0
	0.6	1.1	0	0.0
	0.5	1.1	0	0.0
	0.4	3.1	0	0.0
	0.3	8.3	2	1.2
Mine shaft alt	0.7	0.0	0	0.0
	0.6	1.1	0	0.0
	0.5	1.1	0	0.0
	0.4	3.1	0	0.0
	0.3	8.3	2	1.2
Prospect pit	0.7	0.0	0	0.0
	0.6	0.0	0	0.0
	0.5	2.5	1	0.3
	0.4	6.2	2	0.5
	0.3	11.5	4	1.1
Quarry	0.7	0.0	0	0.0
	0.6	29.4	0	0.0
	0.5	63.0	0	0.0
	0.4	88.2	9	39.1
	0.3	98.6	17	73.9
Adit	0.7	0.0	0	0.0
	0.6	4.0	0	0.0
	0.5	19.9	0	0.0
	0.4	30.9	1	0.7
	0.3	53.5	14	9.7

Table 4-4: Number of ORB keypoints in symbol images

Symbol	Adit	Gravel Pit	Mine Shaft	Mine Shaft Alt	Prospect Pit	Quarry
Number of keypoints	15	53	8	7	10	49

4.4 Performance

The testing environment of this research is Windows 7, 64 bits. The machine has Intel dual core 2.4 GHz CPU and 2GB memory space. The total processing time for one quad is around 25 minutes. It takes 5-6 minutes for noise removal, approximately 15 minutes for the first pass template matching and connected component checking and approximately 5 minutes for the other passes of template matching. This includes the cost of extracting hit images. Without that, the processing time is about 1-2 minutes per quad. ORB requires less than a minute to process hits generated by the first pass template match. These values show that the method is applicable to USGS quads and for batch processing.

It is also interesting to compare the processing time of template matching and ORB for individual locations. For ORB, the average processing time per hit is around 3.4 milliseconds, while this time is less than 0.01 milliseconds for template matching. Though ORB is 3400 times slower, the entire processing time for ORB is not very high because it only needs to deal with the hits of the last pass of template matching, which are less than 10,000 for each quad. Note that nearly 100 hours would be needed if ORB were applied to a full image of 108 pixels. This illustrates the value of combining ORB with template matching.

In the first pass of template matching, we first sample the template of the original templates, which reduces the computational cost by $1 - N'/N$ where N' is the number of pixels retained. Then we apply the sampling strategy to the scanning of original map by only examining ever other image pixel. This further reduces the processing time of first pass template matching by a factor of four.

Table 4-5 shows the processing time of one map by different object extraction methods. The traditional method of hand digitization takes around 2-3 hours for one map [1]. The proposed method, using a combination of template matching and ORB, requires about 25 minutes. Note that naïve application of template matching with no image or pattern sampling would require about 6 hours. This clearly shows the advantages of the optimizations described above.

Table 4-5: processing time of one map by different extraction methods

extraction method		processing time for one quad
hand digitization		2-3h
ORB alone		~100h
proposed method (template matching + ORB)	Full map image resolution, full template resolution on first pass	~6h
	Full image resolution, sampled first-pass template	~1.5h
	Sampled map image and first-pass template	25m

5 Conclusion and Future Direction

5.1 Conclusion

Scanned topographic maps are one of the most abundant sources of geographic information. However, without extraction from scanned maps, such information cannot be easily integrated with other spatial layers for analysis and therefore has little use for scientific research. Traditional extraction methods mainly rely on manual digitizing, which is very costly and not practical for a large quantity of maps.

This thesis proposes a new method to automate the extraction of point objects from scanned topographic maps. The method combines the techniques of color segmentation, template matching and ORB feature detection.

Topographic maps are first processed to remove background noise using color segmentation. To simplify the selection of color ranges, the original map in RGB space is converted to a uniform HSV color space. Any pixels with a different color than the target map objects can then be removed from consideration. Although the target mine symbols are all in black in this research, this method can also be applied to objects with other colors. For objects in any color, background noise with other colors can easily be removed by selecting corresponding range of color values in HSV space. Based on our experience, it seems likely that this would greatly assist object detection.

The object extraction part consists of two principle steps: template matching and ORB object detection. The proposed method applies basic template matching using two types of templates. One type of template focuses on the entire structure of the symbol. It re-samples evenly over all the pixels of the symbols to guarantee both efficiency and the completeness of

the symbol template. The other type of template focuses on individual segments of the symbol. By analyzing each symbol, a set of templates consisting of different components is generated.

ORB feature matching proves to be useful for certain types of features on topographic maps. Distortion caused by scanning artifacts and hand drawing is very common on scanned historical topographic maps. This issue cannot be properly addressed by simple template matching without producing many false positives. By combining these two techniques, we removed more false hits. The use of simple, fast template matching also significantly reduces the total processing time of the more sophisticated ORB feature matching method.

Overall, the proposed method gives satisfactory results in reasonably short execution times. It successfully addresses the issues of scanning artifacts and symbol distortion. The method can also be applied to extract other point symbols with little modification.

5.2 Future Directions

Many other objects on topographic maps are as important as mine symbols. The current method could be extended to enable the extraction of more map objects. For example, bounded objects like gaging stations or campgrounds may be extracted using a similar template matching approach. ORB could also be helpful if enough object keypoints can be detected. Some area objects are constructed by small bounded symbols. The proposed method could be used to extract these objects as well. One limitation of the template matching approach is that it is not scale invariant and thus may require more advanced modifications to extract objects of various sizes.

Although the proposed method can speed up the process of object extraction, it is not completely automatic and still requires user input, which would limit the application of the method in some cases. For example, the templates for individual object segments are currently manually created in our method. This approach would not be very practical if there are many different types of objects to be extracted. Future work could focus on addressing these issues and improving the extraction method so that it is fully automated.

Reference

[1] A. Pezeshk and R. L. Tutwiler, “Automatic Feature Extraction and Text Recognition From Scanned Topographic Maps,” *IEEE Transactions on Geoscience and Remote Sensing*, vol. 49, no. 12, pp. 5047-5063, 2011.

[2] A. Khotanzad and E. Zink, "Contour Line and Geographic Feature Extraction From USGS Color Topographical Paper Maps,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 25, no. 1, pp.18-31, 2003.

[3] H. Yamada, K. Yamamoto and K. Hosokawa, "Directional Mathematical Morphology and Reformalized Hough Transformation for the Analysis of Topographic Maps," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 15, pp. 380-387, 1993.

[4] R. Cao and C. L. Tan, "Text/graphics Separation in Maps,” *Proc. GREC\'01—Selected Papers From the 4th International Workshop*, pp. 167-177, 2002.

[5] A. Velázquez and S. Levachkine, "Text/Graphics Separation and Recognition in Raster-Scanned Color Cartographic Maps,” *Proc. 5th Int. Workshop GREC*, pp. 63-74, 2003.

[6] D. B. Dhar and B. Chanda, “Extraction and Recognition of Geographical Features From Paper Maps,” *International Journal of Document Analysis and Recognition*, vol. 8, no. 4, pp. 232-245, 2006.

[7] A. Pezeshk and R. L. Tutwiler, "Contour Line Recognition & Extraction from Scanned Color Maps Using Dual Quantization of the Intensity Image,” *IEEE Southwest Symposium on Image Analysis and Interpretation*, pp. 173-176, 2008.

[8] N. Kerle and J. Leeuw, "Reviving Legacy Population Maps With Object-Oriented Image Processing Techniques,” *IEEE Transactions on Geoscience and Remote Sensing*, vol. 47, no. 7, pp. 2392-2402, 2009.

[9] S. Leyk , R. Boesch and R. Weibel , "Saliency and Semantic Processing: Extracting Forest Cover From Historical Topographic Maps," *Pattern Recognition*, vol. 39, no. 5, pp. 953-968, 2006.

[10] R. Pradhan, S. Kumar, R. Agarwal, M. P. Pradhan and M. K. Ghose, "Contour Line Tracing Algorithm for Digital Topographic Maps," *International Journal of Image Processing*, vol. 4, no. 2, pp. 156-163, 2010.

[11] R. Brunelli, *Template Matching Techniques in Computer Vision: Theory and Practice*, Wiley, 2009.

[12] Aksoy, M. S., O. Torkul, and I. H. Cedimoglu. "An industrial visual inspection system that uses inductive learning," *Journal of Intelligent Manufacturing*, vol. 15, no. 2, pp. 569-574, 2004.

[13] T. Kyriacou, G. Bugmann, and S. Lauria, "Vision-based urban navigation procedures for verbally instructed robots," *Robotics and Autonomous Systems*, vol. 51, no. 1, pp. 69-80, 2005.

[14] H. Y. Kim and S. A. Araújo, "Gray-scale Template-Matching Invariant to Rotation, Scale, Translation, Brightness and Contrast," *IEEE Pacific-Rim Symposium on Image and Video Technology*, vol. 4872, pp. 100-113, 2007.

[15] Y. Lin and C. Chen, "Template Matching Using the Parametric Template Vector with Translation, Rotation and Scale Invariance," *Pattern Recognition*, vol. 41, no. 7, pp. 2413-2421, 2008.

[16] D. Lowe, "Distinctive Image Features from Scale-Invariant Keypoints," *International Journal Of Computer Vision*, vol. 60, no. 2, pp. 91-110, 2004.

[17] H. Bay, T. Tuytelaars and L. V. Gool, "SURF: Speeded Up Robust Features," *Computer Vision*, vol. 3951, pp. 404-417, 2006.

[18] E. Rosten and T. Drummond, "Machine learning for high-speed corner detection," *Computer Vision*, vol. 3951, pp. 430-443, 2006.

[19] E. Rosten, R. Porter, T. Drummond, "Faster and Better: A Machine Learning Approach to Corner Detection," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 32, no. 1, pp. 105-119, 2010.

[20] M. Calonder, V. Lepetit, C. Strecha, and P. Fua, "BRIEF: Binary Robust Independent Elementary Features," *11th European Conference on Computer Vision (ECCV)*, Heraklion, Crete. LNCS Springer, September 2010.

[21] Ethan Rublee, Vincent Rabaud, Kurt Konolige and Gary R. Bradski, "ORB: An Efficient Slternative to SIFT or SURF," *IEEE International Conference on Computer Vision*, pp. 2564-2571, 2011.

[22] Brunelli, R. (2009) Front Matter, in *Template Matching Techniques in Computer Vision: Theory and Practice*, John Wiley & Sons, Ltd, Chichester, UK. doi: 10.1002/9780470744055.

[23] M. Muja and D. G. Lowe, "Scalable Nearest Neighbor Algorithms for High Dimensional Data," *Pattern Analysis and Machine Intelligence*, vol. 36, no. 11, 2014.

[24] C. Harris and M. Stephens, "A Combined Corner and Edge detector," in *Alvey Vision Conference*, pp. 147-151, 1988.

[25] P. L. Rosin, "Measuring Corner Properties," *Computer Vision and Image Understanding*, vol. 73, no. 2, pp. 291-307, 1999.