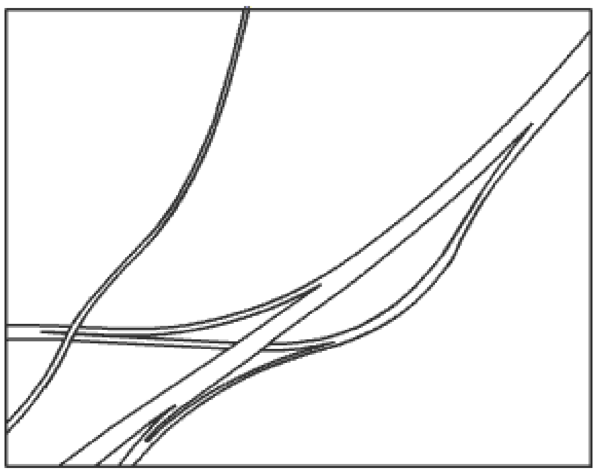
Topographic map model feature recognition

Panja Sae-Ui



M.Sc. in Informatics

Division of Informatics

University of Edinburgh

2003

Abstract

This project is a part of the research project proposed by Ordnance Survey in order to develop a system that recognizes topographic objects from the imagery. An ultimate goal would be to extract the topographic objects from the imagery, vectorise those objects and store them in the database – all in a fully automated manner. Due to the complexity of aerial imagery, it is going to be some time before this goal is likely to be achieved. The aim of this project is to refine the methodology for feature extraction, one of the initial steps in such a system. Two primitive objects, rectangles and parallel lines, are focused.

Table of Contents

Chapter 1

Introduction

Preview

- Introduction to the project
- Aim of the study
- Related work

1.1 Introduction

This study is part of research project proposed by Ordnance Survey in order to develop a system that recognizes topographic objects from the imagery, for example, buildings, roads, field boundaries, and car parks. While updating and maintaining the topographic database has always been a demanding task, the Mapping agencies undertake this task using manual techniques in an intensive manner. This has led to an interest in the theme of computer vision, image processing and remote sensing. An ultimate goal would be, according to a mapping specification, to extract the topographic objects from the imagery, vectorise those objects and store them in the database – all in a fully automated manner. There is a consensus amongst the research literature that due to the complexity of both natural and man-made landscapes, it is going to be some time before this goal is likely to be achieved. Therefore, the aim of this study is to refine the methodology for one of the initial steps in such a system.

There is a requirement for the development of a methodology that derives improved geometry for an object recognition model. Many edge-enhancement techniques exist to extract outlines that can then be inferred to represent a topographic object after applying a subsequent classification procedure. However, work is still needed to develop methods that turn data produced after an edge-enhancement procedure into geometrically and positionally accurate data object primitives that can then be fed into a classification procedure.

1.2 Aim

This project's task is to investigate the extent to which an edge-enhancement methodology can be improved to remove unnecessary noise from its results, and bridge gaps between data primitives, to produce clean shapes to use in future object-classification methodologies. For example, such a technique will accurately extract a building outline before it has been classified as a building! Digital topographic data is provided against which to assess the results.

The feature extraction problem can be called a representation conversion. The characteristic properties of a feature shape are the key point to be extracted in the project rather than the characteristics of an object itself. For example, parallel lines are extracted whether they are roads, rivers or railways. The objective is to develop an automatic topographic feature extraction system such as for rectangle or parallel line extraction not an object feature extraction system such as for building or road extraction. The extraction for different feature shapes can be considered as different problems.

The experiments will be carried out based on the OS MasterMap™ Colour-scanned aerial imagery. To clarify and simplify the problems, the system is therefore separated into modules divided by feature shapes. As mentioned above, two significant feature shapes, rectangles and parallel lines, are the target shapes. The rectangle extraction module extracts rectangle features under the condition that the rectangles must have constant interiors. The parallel line extraction module also extracts parallel lines under conditions such as when the width of the parallel lines should be more or less constant or the length of the lines should be long enough to distinguish the lines from noise. The modules will be discussed in details in chapter 2 and chapter 3 respectively. Example results for both modules are shown in Figure 1.1.

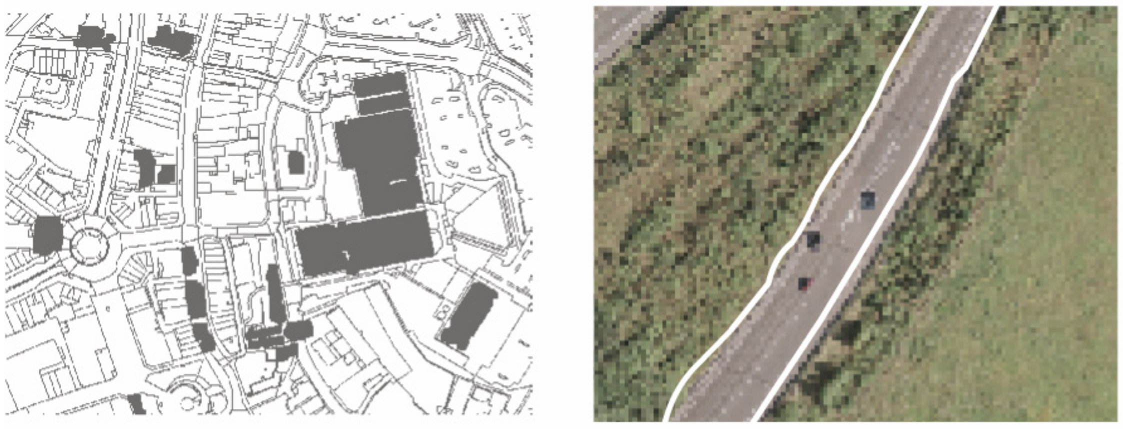

(a) Shaded areas are the extracted rectangles (b) The white lines are the extracted lines

Figure 1.1: Example results for the constant rectangle extraction module (a) and the constant parallel line extraction module (b)

1.3 Related Work

1.3.1 Constant rectangle extraction

There are plenty of previous works relating to the constant rectangle extraction. In early works of Image Understanding System (IUS), low-level image segmentation, such as to detect a rectangle, was considered as a dumb process [24]. IUSs usually consisted of two stages, low-level stage and high-level stage, to deal with different levels of information. Reasoning processes were integrated into a high-level stage [24], [40] so that it can manage high level information. A low-level stage employed inflexible image processing techniques [24] to provide a routine service to the higher levels. Some early works used the high-level stage to adjust parameters for the low-level stage [41]. 'Low-level image segmentation' by Nazif and Levine [42], for example, produced primitive feature shapes such as regions, lines, edges as its outputs [24]. The goal of this work is more or less similar to that of this project. Some works such as a rule-based system [24] employed contextual information to define primitive objects in aerial photographs.

Instead of using context to help identify objects, semi-automated systems require a guide from a human operator. For example [25] corrected results produced by its fully automated process using guides from a human operator. In [43], a model-based approach by Mueller and Olson required a big hint from users. The approach receives the valid range of parameters and an initial location of features from a user. It then focused on a model-based optimization techniques such as Snakes to perform feature extraction [25]. Feature constraints embedded in the optimization reduces the complexity of feature extraction problems. Although this approach can be classified as a manual work with an optimization process rather than a semi-automated system, [43] provides the implementation for the final process of feature extraction, which is optimization.

This approach can be further developed to be a fully automatic extraction by integrating an automatic parameter-adjusting process and an automation initialization into it. [43] was the motivation for this project. This project can be considered as a development of automatic initialization to provide potential feature candidates to an optimization process. The project's goal is to extract primitive shapes, which are constant rectangles, from aerial images without reference to context. Further work could be the integration of optimization and initialization process. Ultimately, an automatic parameter-adjusting process would be developed and integrated into the two processes to build the desired fully automatic feature extraction system.

1.3.2 Constant parallel line extraction

Previous works relating to constant parallel line extraction can be referred to as road extraction according to the similarity of feature shapes. There are many related works and proposed methods are different based on the goals and the available information. The goals, for example, are to track parallel lines with homogeneous intensity between them [30], to extract and construct road network [22], to bridge gaps caused by occlusions and shadows [18], [23]. The available information for example, is low-resolution imagery, high-resolution imagery or contextual information [19], [22]. Additionally, the information acquisition can be categorised into two manners, semi-automatic and fully automatic manner. In semi-automatic schemes, a human operator provides a big guide to the system, for instance, by manually selecting an initial point and a direction for a road tracking algorithm [19], [31], [32] whereas in fully automatic approaches, the initial points are automatically detected [33]. To analyze each method, the available information, which is imagery resolution and contextual information including information acquisition, is discussed first and techniques used in each approach are discussed next.

Road extraction in low resolution can be considered as linear feature extraction since unclear details and lack of information in low-resolution imagery make roads ambiguous and difficult to distinguish from other linear structures [23]. This certainly gives disadvantages in the extraction. Low resolution, however, provides advantages; it dramatically reduces the complexity of road extraction to the general problem of line extraction [26]. Moreover, low details in low resolution do not require a big amount of computational effort and a complicated algorithm [27] that are required in high-resolution extraction. To fully take advantage of detailed information in a high-resolution image, extraction algorithms have to be complicated. This becomes a disadvantage of road extraction in high resolution. A higher-resolution image, however, provides more information to verify a road hypothesis whether it should be accepted or rejected [23]. The approaches in [18], [19] exploit multiple scales, both high and low resolution. The method is so called "multi-scale modelling". It tracks lines in low resolution then uses information in high resolution for profile matching or detection of roadsides [19].

Background contexts have a strong effect on the characteristics of roads at least to the appearance of roads in aerial imagery [19]. Roads in urban area, for instance, can be covered by shadows from buildings, cars, road markings, etc. whereas roads in forests can have shadows cast by trees. If relations between roads and other objects are ignored, a reliable extraction is often difficult [19]. [34] strongly exploited contextual information guiding extracting in complex scenes [19]. Urban contexts were mainly utilized to model urban road networks in [22]. [19] extends multi-scale modelling with contextual information. It made use of detailed information in fine scale and context to verify road candidates. For instance, a road can have markings on it while forests cannot.

Roads in aerial images can be so complicated that interaction with a human operator is necessary [20]. This is called semi-automatic extraction. Many techniques such as correlation-based trackers [28] and edge linkers [35] make use of interaction to obtain initial information, for instance, a starting point, a direction and its width. The semi-automatic extraction is extended to automatic initial point detection in [33] and fully automatic methods in [20], [19]. In [18], the extraction started with fully automatic manner and the result was manually edited afterwards. Stochastic models presented in [20] are used to initialize starting points then dynamic programming technique based on assumptions about the geometry of roads is performed to extract roads [19].

In general, local methods to extract roads can be categorised into three major types: edge linkers, correlation trackers and region-based followers as presented in [20]. As suggested by the name, local methods exploit local criteria to track roads. A region follower, for example, assuming that roads have a homogeneous intensity surface and the road intensity contrasts with the background [36], is used with a correlation tracker, which assumes the existence of a road surface pattern, to extract roads [20]. Since the methods deal directly with image properties, the main disadvantages are errors caused by local anomalies and a large number of parameters to be adjusted such as an intensity threshold, an acceptable error. In addition, parameters have to be readjusted when image contexts or images change [23]. More general road extracting operators, which are independent of absolute intensity of roads, appeared in papers to eliminate the weakness of threshold setting such as DRO (Duda Road Operator) in [37] and other general methods in [26].

Probabilistic models were presented in [20]. The aim of [20] is a completely automatic system to find the main roads in aerial images exploiting geometric probabilistic models. Before the advent of probabilistic models in [20], a Bayesian approach in low-level boundary estimation was introduced in [38] and [39]. Instead of exploring the absolute value of images such as intensity, the probabilistic models provide more general operation for road extraction.

Active Contour Models, also called Snakes, were introduced in [44] and [29]. Snakes overcome errors with local anomalies occurring in purely local criteria methods; moreover, it reduces a large number of adjustable parameters. Snakes combine geometric properties of features in the form of constraints guiding search in deformation process [23]. The whole process of snakes is a deformation process to move snake curves to a position where the curves provide minimum total energy; the process is also called an optimising process. The total energy is the sum of all relevant energies calculated from force constraints. An internal force and an image force are incorporated in snakes presented in [21] to define an elastic energy and a photometric energy respectively. An elastic energy is derived from geometric constraints such as curvature while an image energy is from photometric constraints such as contours of contrast. A compromising position between these two energies gives energy minimizing curves which fit the local maximum of the specified road properties [23] comparing to the surrounding areas. As presented in [19], [18], [22] and [23], since an internal force ignores local photometric anomalies, snakes can successfully trace roads through occlusions and shadows.

The goal in this chapter is to extract parallel line feature with more or less constant intensity between them. Thus, it is not necessary to consider contextual information as road extraction methods in some papers do since the target objects are geometric shapes not a meaningful representation. The snake in [23] is the motivation for this paper. However, the paper did not provide the details of the method to initialize starting control points for snakes. In this paper, the initialization for snakes exploits geometric knowledge about parallel line features such as low curvature, length and width to track centrelines of roads and spread the centrelines out to hypothesised roadsides. The roadsides are used as snakes' starting points (control points) then the snake optimizing process optimised hypothesised roadsides to locate correct roadsides. The approach can be considered as a multi-scale technique since input imagery is blurred to reduce the details to track centrelines and the optimization delineates snakes using information in high resolution. In addition, to simplify extraction in complicated images, a big guide from a human operator is provided. A group of feature pixels is selected at the beginning of the process to build a colour filter to filter out irrelevant pixels; the use of colour filter is discussed in the strategy section in chapter 3. Although a hint is given, the approach still extracts the features mainly in an automatic manner. Therefore, it can be considered as fully automatic extraction. The details of strategies and the implementation are presented in chapter 3.

Chapter2

Constant Rectangle Extraction

Preview

- The problems and the goals
- Strategies for the automatic extraction system and its problems during the design and development process
- The implementation details, experiment and results

2.1 The Problems and the goal

This chapter focuses on problems related to extracting constant rectangles from high-resolution aerial images provided by Ordnance Survey. The constant rectangle is defined by its interior's property of having constant or nearly uniform intensity pixel. Examples of such rectangles appearing in input imagery are buildings' roofs, car parks, and field boundaries.

A problem occurs when the interior of a constant rectangle has the ambiguity of intensity pixels. The target constant rectangles cannot be precisely defined in an aerial image. Some features may be occluded by its surroundings. These cause problems to the extraction. For example, three flat roofs could not be detected as three constant rectangles because the intensity of consecutive features is deceived by inconstant lighting.

This development of the existing system aims to filter out unnecessary noise until there are only significant features to be dealt with. Therefore, many steps employed in this approach are to achieve noise reduction wherein many geometrical techniques are applied.

2.2 Strategy

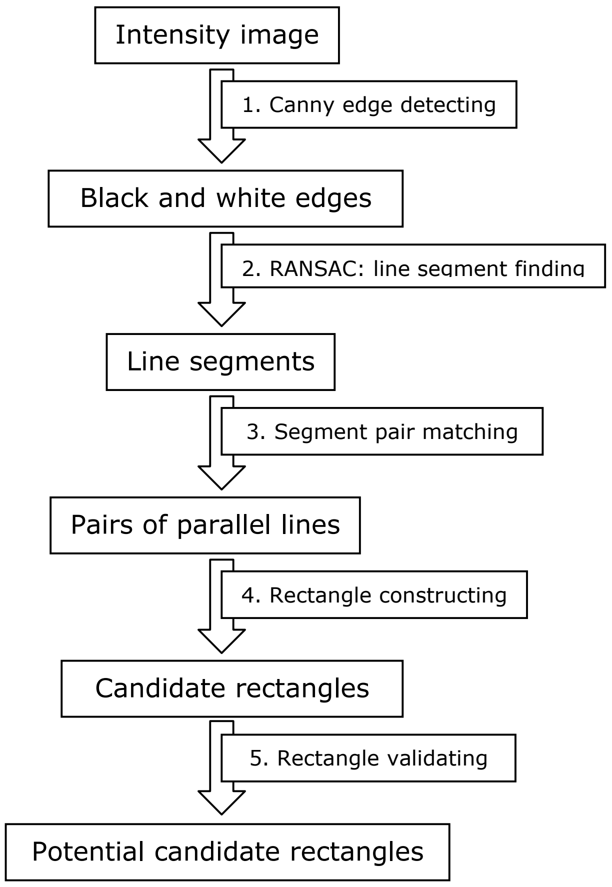


Figure 2.1: The first plan diagram

The strategy of this approach originated from the first plan diagram shown in Figure 2.1. The plan had five steps labeled in numerical order. To begin with, step 1 was to detect edges in an original intensity image. The edges would be detected easily using Canny edge detection technique [1]. Line segments could then be found using the RANSAC technique [2]. RANSAC selects two pixels at random and draws a line through them until it reached the boundary of the image. The fewer edge pixels on the line the more likely it was only noise. Only substantial lines would be kept; noise would be removed. The lines were shrunk from ideal lines to real segments. In order to reduce the number of segment candidates, a length constraint could be applied at this step; a too short line segment could be considered as noise pixels and could be rejected. Up to this point, clean line segments are found and a large amount of noise would be removed. Then, each segment was paired with parallel segments within the same range so that each pair could be treated as two parallel sides of a rectangle in the next step. Due to the assumption that only promising segments would be provided from the previous step, the pairing procedure could be simply implemented using exhausted search, which is to search all possible combinations. Next, rectangle candidates could be constructed from those segment pairs. Match any two pairs that were perpendicular to each other within tolerant degree and they must intersect to each other in the way that a rectangle could be constructed from as depicted in Figure 2.2 below.

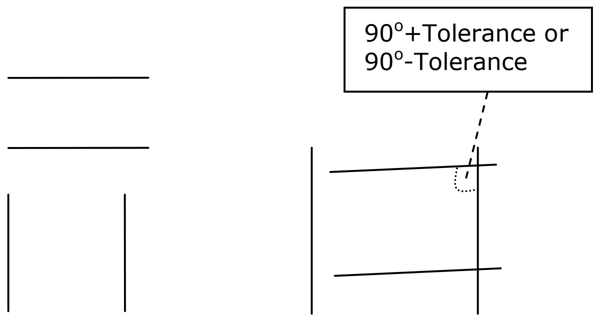


Figure 2.2: Match two segment pairs; even if the left pairs are perfectly perpendicular to each other, they cannot be matched since there is no intersection between them. The right pairs are an example where two pairs can be matched.

The rectangle validation filtered out the candidates that could not fit some constraints such as the size constraint and the internal intensity constraint. The internal intensity constraint verified that a potential rectangle has internal homogenous intensity. A rectangular car park, for instance, would not be classified as a potential rectangle if it were full of cars or markings that made its intensity inconstant.

The accuracy of the scheme could be evaluated by performing overlap comparison between the potential rectangles and ground truth of expected rectangles produced manually. This evaluation precisely assessed locations, size and shapes of the features. By changing parameters such as a size threshold, tolerant degree, etc, and executing the scheme iteratively, one could perform an experiment measuring accuracy of the scheme. The results of the experiment would be the number of achieved rectangles, percentage of overlapped area, the number of false negatives and false positives.

The scheme discussed above seemed theoretically achievable. However, a problem came up in the beginning of the implementation. Since the Canny edge detection produced too messy edges to be performed by the next step, the RANSAC line segment finding, the whole strategy had to be redesigned.

To detect lines from edges was rather computationally expensive and time consuming because the algorithm had to deal with each single edge pixel that could possibly be either a real edge pixel or noise. Moreover, the real segments producing a rectangle feature were so short that it was difficult to distinguish true lines from noise according to the algorithm discussed above.

If the binary edge image, where edges are white pixels and the others are black, is inverted, it will be transformed into a region image where regions are separated by black edge pixels as shown below.

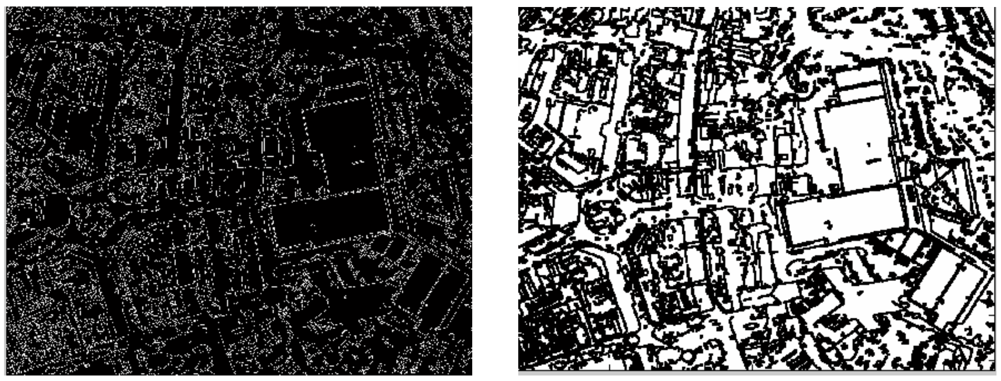


Figure 2.3: Edge image and region image

Since dealing with groups of pixels such as regions is easier for noise removal than directly dealing with each pixel, a region-based technique was then proposed and used as the main technique here. When a region approach is implemented, spurs can be removed by performing morphological operators such as opening and closing operator [3]. Then, the program reduces number of candidate by employing some constraints. A size constraint, for example, restricts size of potential regions; a region will be rejected if its size is not in the range of a valid threshold. A geometrical constraint, such as compactness, also can be used to reject weak candidates. Next, each region will be verified whether it is a rectangle or not. The first attempt to do this was to use the RANSAC technique to select two points on the perimeter of a region at random, draw a rectangle in the region using the selected points and evaluate the overlapped area as shown below.

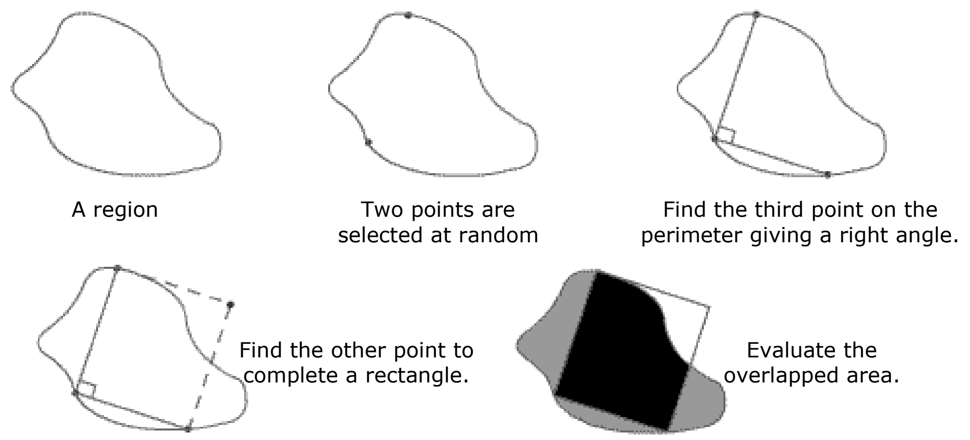

A region
Two points are selected at random
Find the third point on the perimeter giving a right angle.
Find the other point to complete a rectangle.
Evaluate the overlapped area.

Figure 2.4: Rectangle test using RANSAC

From Figure 2.4, there are two types of error areas, outbox and inbox error. An outbox error, the grey area, is a region area outside the hypothesis rectangle while an inbox error, the white area, is the hypothesis rectangle area outside the region. A region will be classified as a rectangle if the percentage of an overlapped area is higher than a threshold, seventy per cent by experiment, and both outbox and inbox area are lower than another threshold, ten to twenty per cent also by experiment. The RANSAC iteration is, therefore, a process of randomly fitting a rectangle to a region randomly. Because

RANSAC is a probability technique, it does not guarantee that the good fits will be found every time the RANSAC is performed. Although a success probability could be increased [2], it would also increase the number of iterations and make RANSAC too computationally expensive to be acceptable.

The second attempt at rectangle testing was to fit a bounding box to a region then evaluate an outbox error, inbox error and an overlapped percentage in a similar way to the first attempt. A bounding box is the smallest rectangle box bordering a whole region; it is one of region properties provided by 'regionprops()', a MATLAB built-in function. A bounding box lies on the standard x-y axis. Therefore, a region had to be rotated to the standard axis before calling regionprops() otherwise it is going to produce a big inbox error as shown below.

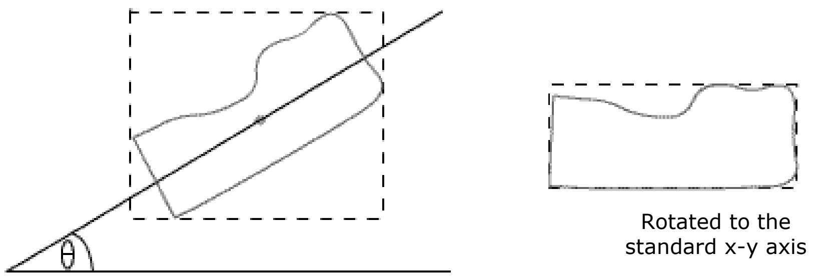


Figure 2.5: A bounding box

This technique was simple and provided a good result. However, it also provided some false negatives; it rejected good candidates. For example, the region in Figure 2.6 would be classified as a rectangle using RANSAC technique in the first attempt. It would, however, be rejected by the bounding box technique because of a big inbox error, the grey area in Figure 2.6.

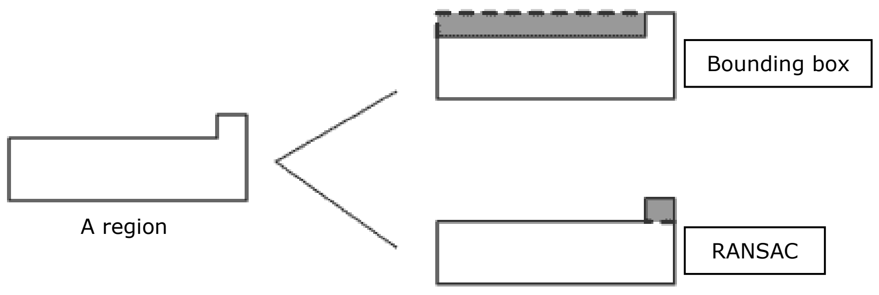


Figure 2.6: Negative false in bounding box

Since the bounding box technique eliminated many of good regions, it was not good enough and the technique itself was rejected. Although the RANSAC technique, discussed above, is slow, it gives good classification as displayed in Figure 2.6. Therefore, the final attempt is to speed up the RANSAC by decreasing the number of iterations. To achieve the goal, the number of points RANSAC can select has to be reduced. In the first attempt, RANSAC selected two points from the perimeter of a region. While there are plenty of those points in a single region, only few of them can generate a fitted rectangle. It was, therefore, useless to take irrelevant points into account. Corners are more promising to give a good fitted rectangle than any other points on perimeter and the number of corner is far smaller than the number of perimeter point. Selecting only corners, the number of iterations is dramatically decreased so that it is not necessary to use the RANSAC technique anymore. Even though all-possible combination of corners is exhaustively searched, the spent time is still acceptable. A corner detecting technique will be discussed in detail in the implementation section.

The final strategy used for this constant rectangle extraction can be depicted briefly in Figure 2.7. In the first step, an RGB aerial image is fed to an edge detector producing an edge image. Performing a couple of morphological operators, the edges are transformed into region candidates. Noise reduction techniques and filter constraints will be applied to the candidates in order to filter out unnecessary noise and weak candidates. Finally, each potential region will be tested if it is a rectangle using the mentioned-above technique. All processes are discussed in detail in the next section.

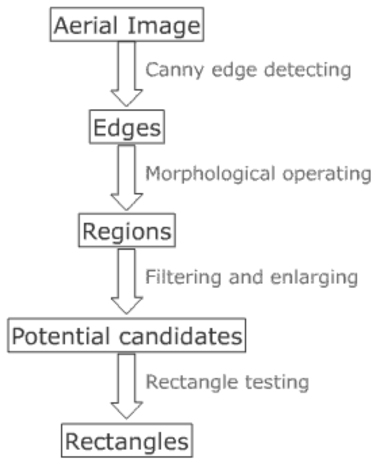


Figure 2.7: Strategy diagram for the rectangle extraction

2.3 Implementation

The four main processes in Figure 2.7 are detailed in this section respectively and the results are shown at the end of each process discussion. The discussion of the fourth process, rectangle testing, which is rather complicated, will be clarified by pseudo code.

2.3.1 Edge detection

There is a built-in function named ‘edge()’ in MATLAB ,which does edge detecting task. ‘edge()’ takes an intensity image as its input and returns a binary image of the same size as the input with 1’s where the function finds edges in the input and 0’s elsewhere. An original aerial image is converted from RGB to grayscale so that it is compatible with the edge function. The function supports six different edge-finding methods; Canny edge detection is the one used in this project. There are three adjustable parameters in Canny, two sensitivity thresholds, high and low threshold; and sigma, the standard deviation of the Gaussian smoothing filter [4]. Gaussian smoothing is a 2-D convolution operator used to blur image detail and noise. In this sense, it is similar to other filters such as mean filter, median filter or conservative filter but it uses a different kernel that represents the shape of Gaussian bell-shaped hump [5]. Briefly, the greater the sigma is, the more the Gaussian filter blurs the image. By experiment, suitable sigma values for this system vary from 0.1 to 3.3.

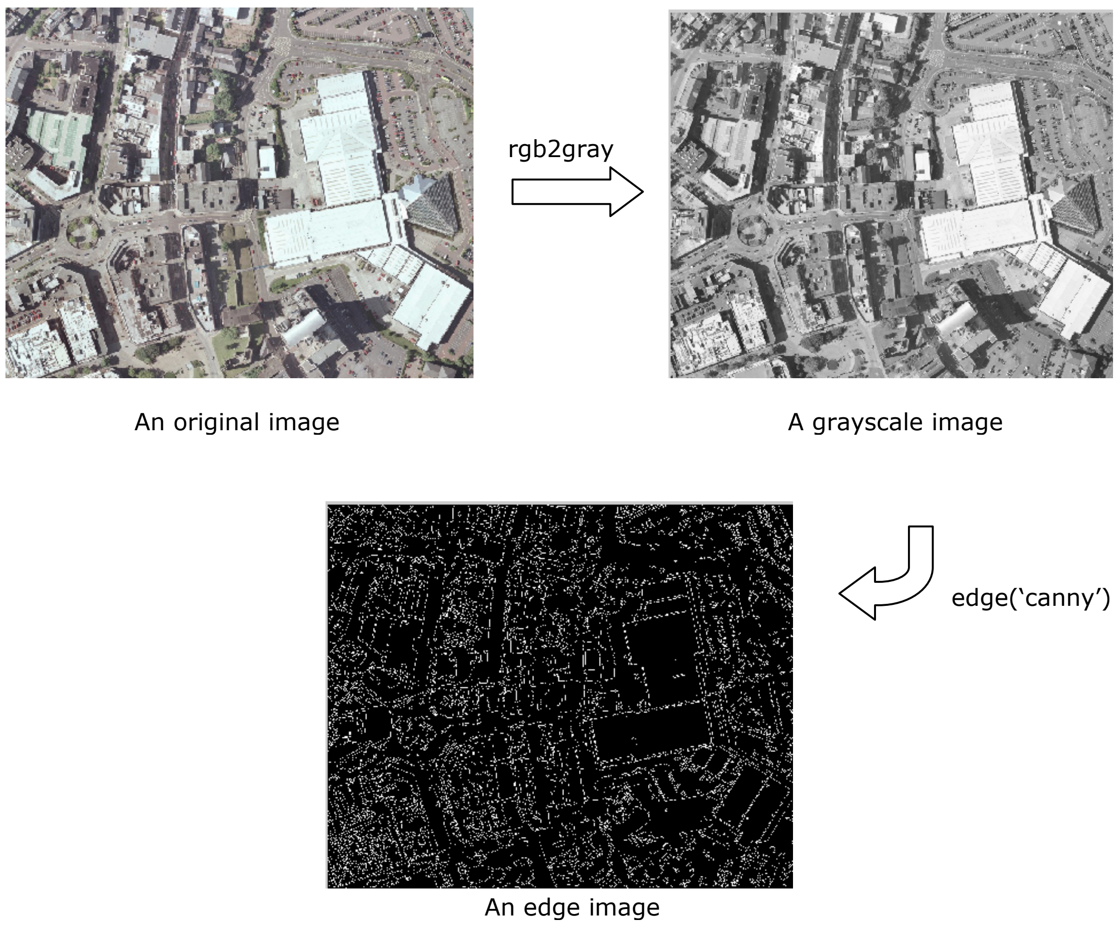


Figure 2.8: Edge detection

2.3.2 Region finding

This process is to transform an edge image into regions as well as reduce unnecessary noise such as spurs to provide a clean region image. Applying the binary invert operator to the binary edge image gives a region image in binary with 1's where a region is found and 0's elsewhere is given. The regions will be emphasized by erosion [6] that shrinks region pixels in size. Erosion also separates regions that used to be connected by small spurs. This advantage allows region labeling to distinguish and label each region correctly; otherwise, all connected regions will be mistakenly grouped into a massive region. The size of regions is now smaller than that of original regions; they will be grown back to their original size later. Then, in order to remove spike noise regions, opening operator [7] using a 3x3 square structuring element is applied to the region image for three times. The result is shown in the figure below.

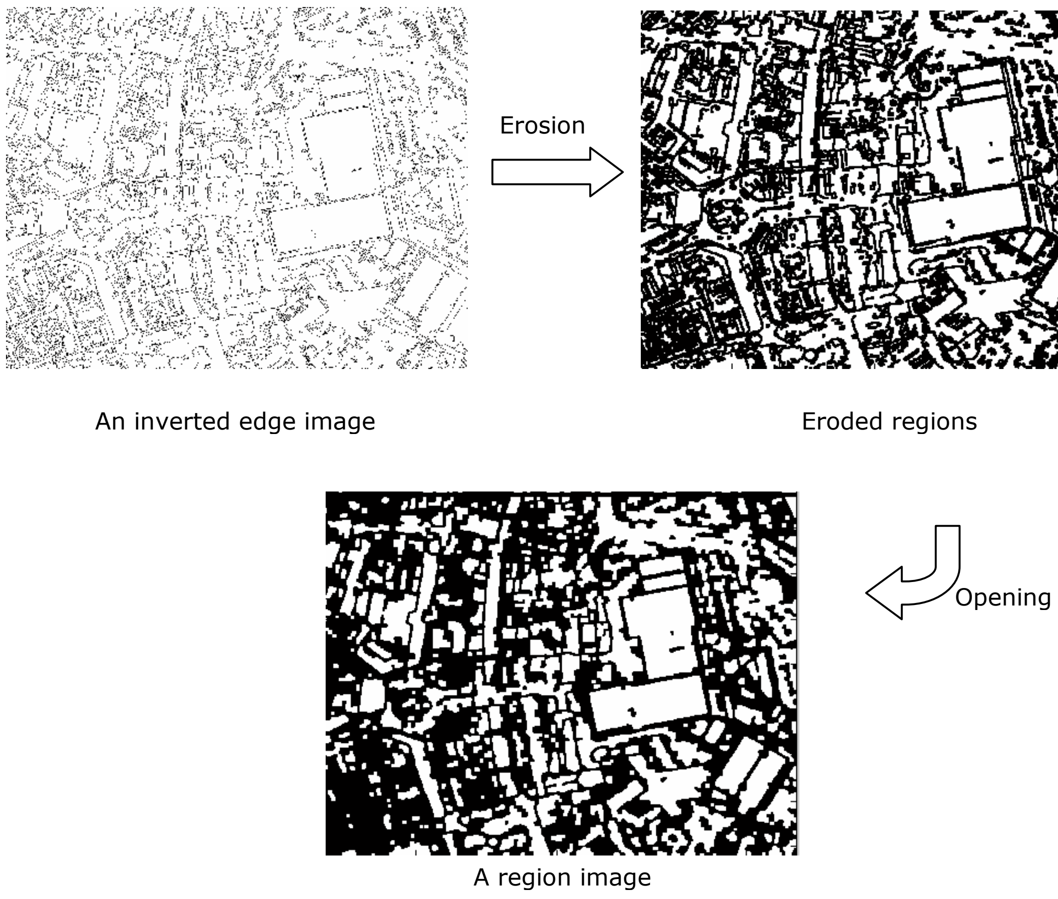


Figure 2.9: Region finding process

2.3.3 Region filtering and enlarging

Although the regions provided from the previous process are tidy and much of noise are removed, a region filter is still needed before performing the rectangle test in order to reduce the number of candidates and remove the noise that ‘opening’ cannot remove. This has been done by size filter. If a region is smaller than a low threshold, the region is considered as a noise region. A noise region is a group of noise pixels that is too big to be removed by opening operator and is too small to be a rectangle. If a region is larger than a high threshold, it is rejected as an irrelevant open area. Even if a massive region such as a lake, a forest, or a meadow, is rectangular, it will be rejected because it is larger than expected rectangle size. This can save a lot of computational time in the rectangle test since the program will not waste time detecting a great number of corners of a massive region, which finally will be rejected. As mentioned previously, the regions are smaller than their original size because of erosion effect. They will be approximately recovered back to their size by simply applied dilation [8], the opposite operator of erosion, to the regions. To do the operation mentioned above, the regions have to be labeled so that each region can be manipulated independently. MATLAB provides a useful function for this task, bwlabel(). It groups connected pixels together and names each group with different number. When the pixels are shown using imagesc() function, it ends up with a unique color for each group. That is the reason for the colorful regions in the figure below.

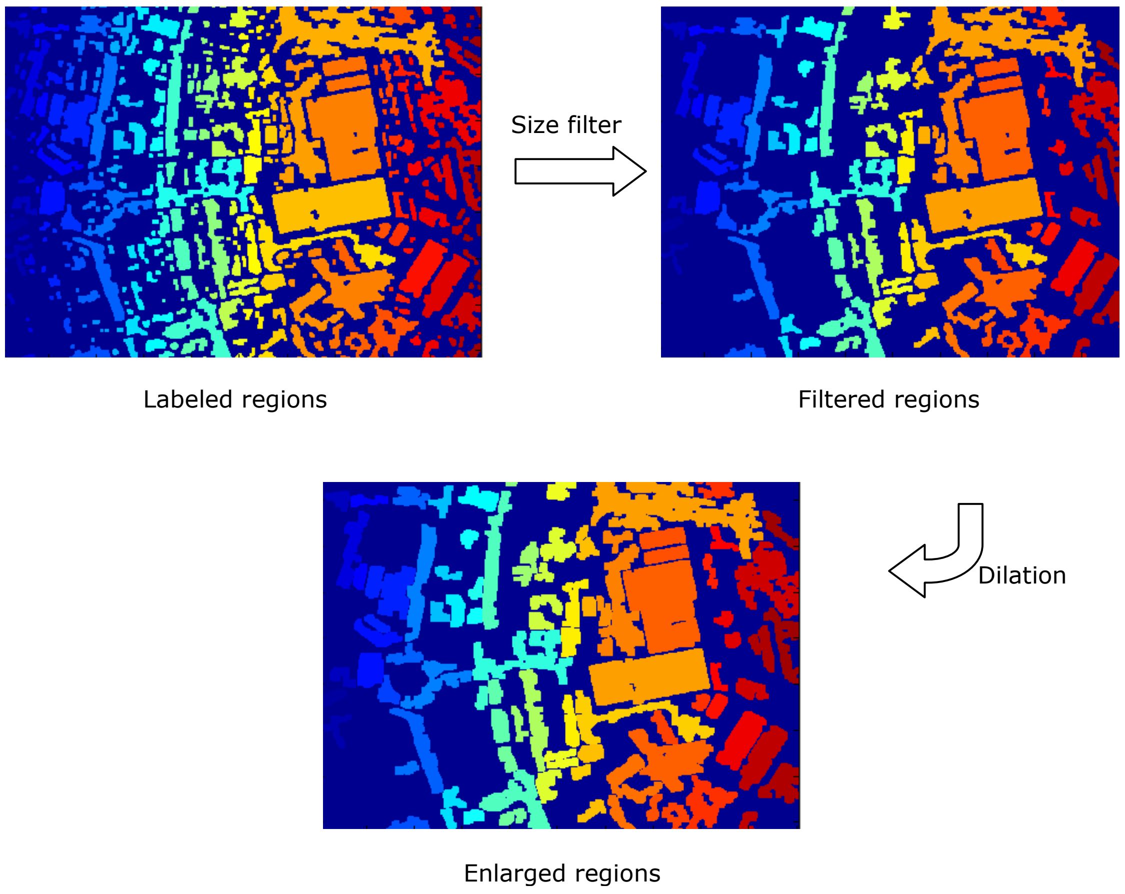


Figure 2.10: Region filtering and enlarging

2.3.4 Rectangle testing

The test iteratively fits a rectangle to a region. If a rectangle fits a region with an acceptable 'inbox' and 'outbox' error, the region will be classified as a rectangle. Each region candidate is an enlarged region from the previous process and each tested rectangle is created from three corners of a region. The test process, therefore, starts with corner finding. Then all selections of three corners will be tested exhaustively until a fitted rectangle is found.

A corner is a pixel on boundary of a region. The boundary points are easily found using 'bwperim()' function in MATLAB. However, the given boundary is rather jagged because of spurs, unnecessary boundary pixels, shown in the figure below. Additionally, bwperim() does not give a consecutive boundary track that is required for the corner finding algorithm.

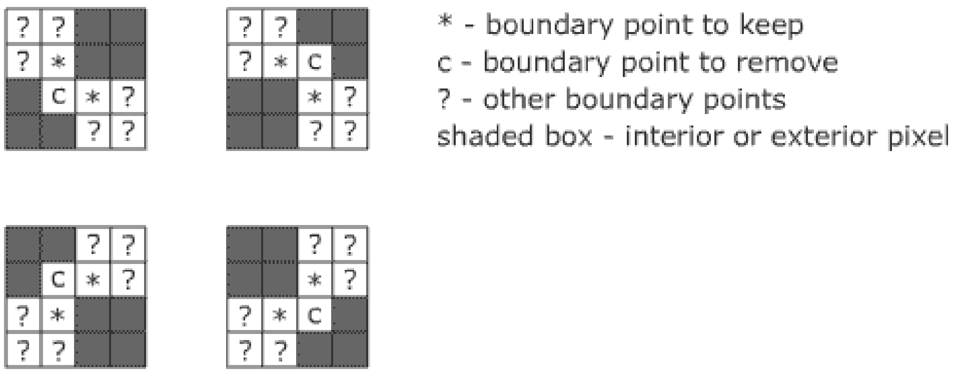

* - boundary point to keep
c - boundary point to remove
? - other boundary points
shaded box - interior or exterior pixel

Figure 2.11: Unnecessary boundary pixels
(Adapted from [9])

Therefore, boundary pixels have to be fed into two preprocessing steps, which are dangling spur removing and boundary tracking [9], before corner finding. The source codes of both are in [9] and can be downloaded from the URL provided in the bibliography.

Corner finding unit can be adapted from linear segment splitting technique [9]; it is a recursive splitting algorithm. The code is also available in the same page as spur removing and boundary tracking code. As shown in Figure 2.12, the algorithm finds the leftmost point A and the rightmost point B on the boundary and creates a line segment through point A and B. If the distance from the furthest boundary point C to the line is less than a threshold, then this segment is finished. Otherwise, the algorithm will create two new segments from A to C and from B to C and recurse [9]. The algorithm was designed to find linear segments. Instead of using segments as the output, it can be transformed to be a corner finder by using segment endpoints, which actually are corners, as the output. The threshold mentioned above controls the sensitivity of the corner-finding algorithm and decides the detail of the corners; it is then called 'corner detail threshold'. The figure below, for example, shows the effect of the corner detail to the number of detected corner.

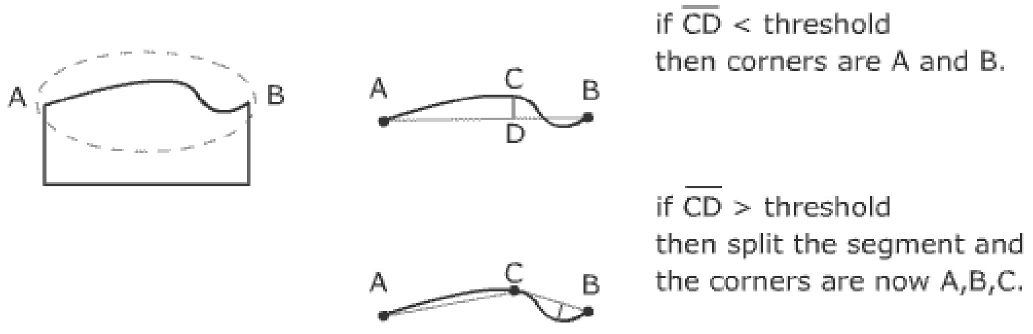

A
B
A
C
B
D
if $\overline{CD}$ < threshold
then corners are A and B.
if $\overline{CD}$ > threshold
then split the segment and
the corners are now A,B,C.
A
C
B

Figure 2.12: Corner detail threshold

Suppose the process of corner finding comes to the segment between point A and point B. If the distance from the furthest point to the segment, CD, is less than the corner detail threshold, the process will be finished and the found corners will be point A and B. Otherwise, the segment will be split into AC and CB; now the found corners are at least points A, B and C and the process still continue recursively. The threshold determines the number of corners. The less the threshold is, the higher the number of corner that are detected.

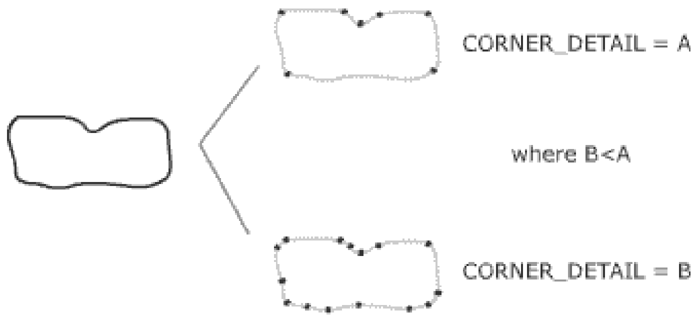


Figure 2.13: The number of corner and corner detail threshold

Since the target is a rectangle, a region must not have less than four corners; otherwise, it will be rejected. To reduce computational time consumption, a region having more corners than a limit will be automatically rejected as a non-rectangle region by the test. Moreover, a three-point selection that does not give a right angle within tolerance will also be ignored and the next selection will be fed to the test. The whole test process is illustrated in the figure and in the pseudo code below.

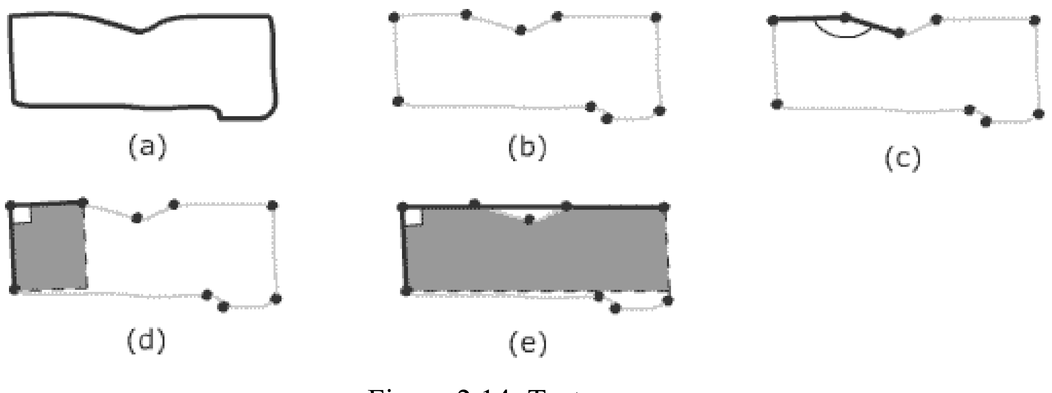


Figure 2.14: Test process

Figure 2.14(b) shows detected corners of the region in 2.14(a). Three points are selected in 2.14(c) but the selection does not provide a right angle; therefore, the three points will be reselected. The shaded areas in 2.14(d) and 2.14(e) are an overlapped area. Although the selection in 2.14(d) gives a right angle, it provides unacceptable outbox error. The reselection process will continue until the selection giving a fitted rectangle with acceptable inbox and outbox error in 2.14(e) is selected. The region will be classified as a rectangle and the next region will be fed to the test.

Pseudo code of the rectangle test:

```
1-    For each region
2-        detect corners;
3-        if the number of corners > a limit threshold or < 3
4-            do next region;
5-        end;
6-        For each three-point selection
7-            if the selection provides a right angle within tolerance
8-                find the fourth point to complete a rectangle;
9-                calculate overlapped area, inbox and outbox error
10-               if inbox error < threshold and outbox error < threshold
11-                   store a region in output variable;
12-                   do next region;
13-               end;
14-           end;
15-       end;
16-   end;
```

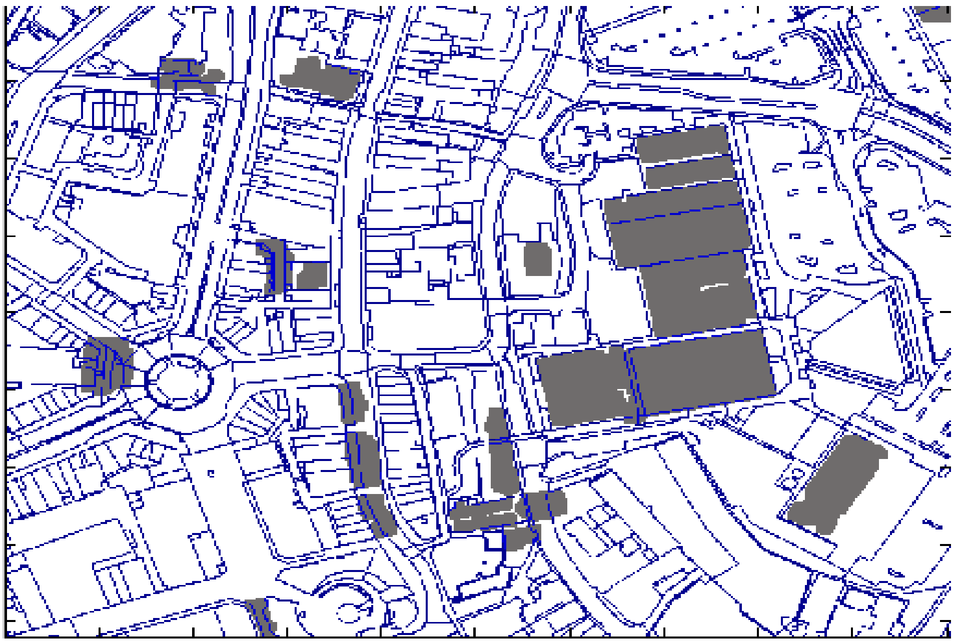


Figure 2.15: Extracted rectangles (dark shaded area) overlaid with ground truth

Figure 2.15 shows the result of the rectangle extraction. As can be seen, most of the significant rectangles were extracted. There are many false negatives and some of the extracted rectangles are false positives caused by occlusions. This will be discussed in the result section.

2.4 Experiment

The aim of the experiment is to measure the accuracy of the rectangle extraction rather than its speed. As a result, only geometrical and positional correctness will be evaluated. The evaluation will be accomplished simply by comparing the extracted rectangles to the expected rectangles and calculating the number of matched rectangles, false negatives and false positives. The false negative occurs when the extraction misses a true rectangle while the false positive occurs when it gives a wrong rectangle. Additionally, the percentage of overlapped area, inbox error and outbox error of hit rectangles are also calculated. The comparison is performed pixel by pixel so that both geometrical and positional correctness will precisely be measured. To find the best result the system can provide, the experiment is performed iteratively with different sets of control parameters. The result will show the best parameter set providing the smallest number of false negatives and false positives.

To set up the experiment, control parameters were selected. A control parameter was a threshold or a tolerance value whose change could considerably affect the extracted output. The control parameters were selected by experiment. Then, using an OS-provided image, ‘swansea_aerial_image.tif’, as an input image, the selected parameters would be manually optimized. Next, parameter range sets are formed. The members of each set are specified to be higher and lower than the optimum value in order to cover both good and bad ranges of the search space. The control parameter sets are shown below.

SMOOTHING	= {0.1, 0.9, 1.7, 2.5, 3.3}
MORPHOLOGICAL_SIZE	= {3, 5, 7, 9}
CORNER_DETAIL	= {3, 5, 7, 9}
ANGLE_TOLERANCE	= {2, 4, 6, 8, 10}

The SMOOTHING set contains sigma values of Gaussian smoothing for the Canny edge detection. The greater the sigma is, the smoother the image is going to be. The MORPHOLOGICAL_SIZE set contains the size of structuring element used in the region finding process. The larger the size is, the more effectively the erosion removes unnecessary noise. However, a large structuring element will also destroy the detail of the region shape. The CORNER_DETAIL set contains the corner detail threshold discussed in the previous section. The smaller, the more detail. The ANGLE_TOLERANCE set contains tolerant angles that one of angles in a rectangle can acceptably be away from a right angle. Every combination of the control parameters was assessed. Eventually, four hundred (5x5x4x4) results were produced from the experiment.

2.5 Result

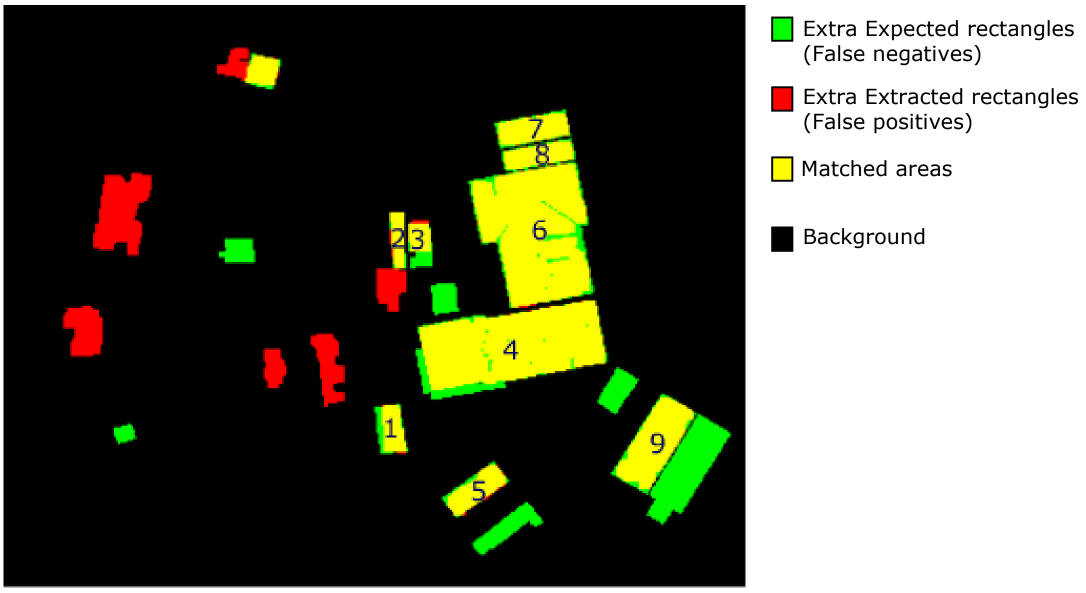

7
8
6
2
3
4
1
9
5
Extra Expected rectangles (False negatives)
Extra Extracted rectangles (False positives)
Matched areas
Background

Figure 2.16: Experiment result, 'swansea_aerial_image.tif' (The labels correspond to the matched pair No. in Table 2.1)

Figure 2.16 shows one of the four hundred results using swansea_aerial_image.tif as an input image. The expected rectangles, the green areas, are manually generated. The extracted regions are red and some of them are false positives caused by occlusion such as rectangle-shaped shadows. The table below displays the percentage of the overlapped areas above. The parameter set used to generate this result is [0.1, 7, 7, 4] where the leftmost is SMOOTHING then MORPHOLOGICAL_SIZE, CORNER_DETAIL and ANGLE_TOLERANCE respectively.

The number of expected rectangles:	16
The number of extracted rectangles:	15
The number of matched rectangles:	9
The number of false negatives:	7
The number of false positives:	6

Matched pair No.	Overlapped percentage	Interior error percentage	Exterior error percentage
1	73.48	26.52	1.6
2	93.89	6.11	4.55
3	61.23	38.77	6.93
4	90.02	9.98	0.12
5	88.39	11.61	2.66
6	92.36	7.64	0.24
7	90.41	9.59	0
8	88.43	11.57	0
9	91.19	8.81	0.19

Table 2.1: Experiment result showing the accuracy of the system; (% of overlapped>60 and % of exterior error < 10)

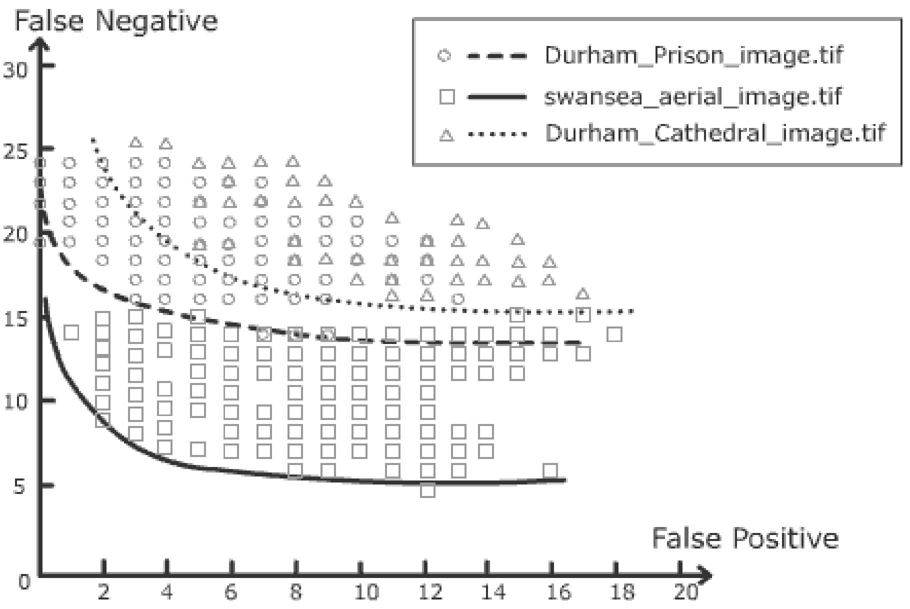


Figure 2.17: The system boundary

Three aerial images were used in the experiment, ‘Durham_Prison_image.tif’, ‘swansea_aerial_image.tif’ and ‘Durham_Cathedral_image.tif’. Figure 2.17 shows the boundaries of the experiment results of the images. The boundary indicates capability of the system; it cannot perform better than this limit. For example, the best extraction the program could perform in swansea_aerial_image.tif provided five false positives and six false negatives. These six false negatives are the expected rectangles that cannot be found by the program. This occurs when the intensity of the expected rectangle is indistinguishable from the intensity of its surroundings. It is then incorrectly rejected in the rectangle verification process. The best parameter sets for both images are shown in the table below.

Image name	False Negatives	False Positives	SMOOTHING	MORPHOLOGICAL _SIZE	CORNER _DETAIL	ANGLE_ TOLERANCE
Durham_Prison_image	15	7	0.1	5	5	8
Durham_Prison_image	15	7	0.1	5	5	10
Durham_Prison_image	15	7	0.1	5	7	8
Durham_Prison_image	15	7	0.1	5	7	10
Durham_Prison_image	14	8	0.1	5	3	8
Durham_Prison_image	14	9	0.1	5	3	10
swansea_aerial_image	6	5	0.1	7	7	4
swansea_aerial_image	7	4	0.1	7	9	4
swansea_aerial_image	8	3	0.1	3	9	2
Durham_Cathedral_image	19	5	0.1	3	3	2
Durham_Cathedral_image	17	11	0.1	3	3	4

Table 2.2: The best parameter sets

2.6 Conclusion

From the results above, a rectangle having constant interior and distinct intensity in comparison to its surrounding pixels was correctly extracted. Most of the matched rectangles are this kind of distinct rectangle. However, there are extracted rectangles that do not look like a rectangle. The reason for this is weak constraints in classifying a region candidate as a rectangle. The acceptable inbox and outbox percentage, for example, might be too weak.

There is a reason for false positives caused by occlusions such as a rectangle-shaped shadow in the figure below. It is not a technical problem; it is the assumption. Since the program is designed to extract constant rectangles, anything rectangle-shaped with a constant interior is extracted. There is no clue to hint which rectangle is expected, which is not. The program has done reasonably well in constant rectangle extraction. If rectangle-shaped occlusions were included in the set of expected rectangles, the results would be better because the current false positives could be considered as matched rectangles.

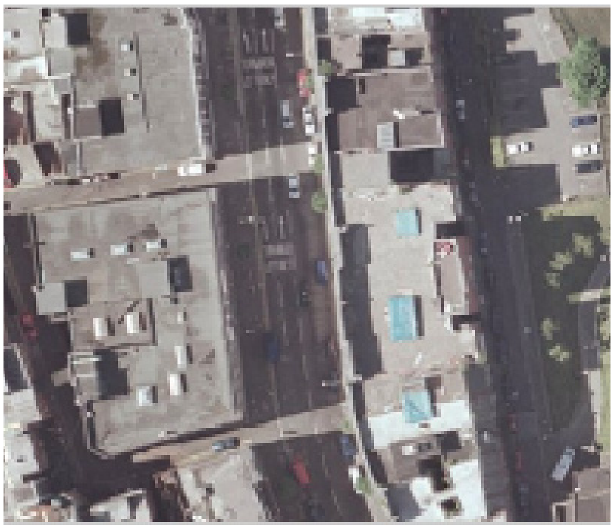


Figure 2.18: A rectangle-shaped shadow

Intensity is everything in this extraction technique. It relies on the intensity of an image in the first place in edge detection. The following steps only deal with the information based on the consequence of the edge detection. This is the reason for false negatives. For example, an expected rectangle with perfect rectangle shape and purely constant interior might be rejected by the extraction if it is located next to another object having almost the same intensity. Human eyes could certainly distinguish these two features. Edge detection, however, might be fooled by the intensity as shown in the figure below.

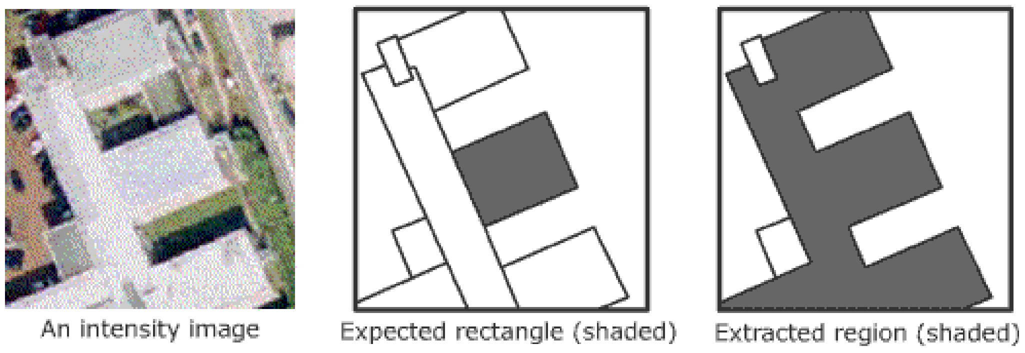


Figure 2.19: Intensity causing error

As can be seen in Figure 2.17, the boundaries of different images with the same parameter sets in the experiment are so different. This leads to a hypothesis that different images might need different sets of parameters. Since each image has its own characteristics, it might need thresholds in a specific range. The valid size of a rectangle, for example, could be different in different images. It might vary from seven hundred pixels to twenty thousand pixels in one image while in another it image might vary from two hundred pixels to five hundred thousand pixels. As a result, parameters would have to be readjusted every time the input is changed. This is not practically convenient.

In conclusion, the constant rectangle extraction has done well although it provides some false negatives and positives. In the case of matched rectangles, it shows high accuracy according to the overlapped area and error percentage in table 2.1. As discussed above, some modifications could bring about improvements. Rather than relying only on intensity, for example, a model-based technique could possibly be integrated into the program to give another definition of a constant rectangle in term of geometry to guide the extraction. Moreover, an automatic parameter tuning process employing evolutionary programming technique could also be useful. These ideas will be discussed in chapter 4.

Chapter 3

Constant Parallel Line Extraction

Preview

- Introduction to the problems and the goal
- The strategy and implementation
- ‘SNAKES’ Theory
- The experiment comparing the extracted results to expected ground truth
- The experiment results and conclusion

3.1 The Problems and the goal

This chapter focuses on constant parallel lines as the targeted primitive feature. The constant parallel lines are defined by homogeneous intensity between them whereby the extraction is simplified. As discussed in chapter 1, standard edge detections usually detect too many or too few edges from complicated aerial imagery because the extraction relied mainly on image intensity. It is therefore necessary to incorporate other criteria into the constraints to guide the extraction and avoid errors caused by local anomalies.

This study proposes an alternative technique employing geometric and photometric properties as additional criteria. The alternative technique is investigated through the experiment presented in this chapter.

3.2 Strategy

This approach presents linear feature extraction using purely geometric constraints to track centerlines. The centerlines are then used to initialize hypothesis roadsides, the starting control points for Snakes [29]. Snakes make use of geometric constraints and photometric constraints in the form of energies to optimize control points. The hypothesised roadsides eventually are deformed to correct roadsides.

The general ideas for the extraction are illustrated in Figure 3.1. To begin with, the centerlines are detected by the line tracking process. The centerlines are used to initialize hypothesised side points that will be used in the Snake optimizing process. Finally, the optimizing process produces extracted parallel lines. The details of the three main processes are in the following sections.

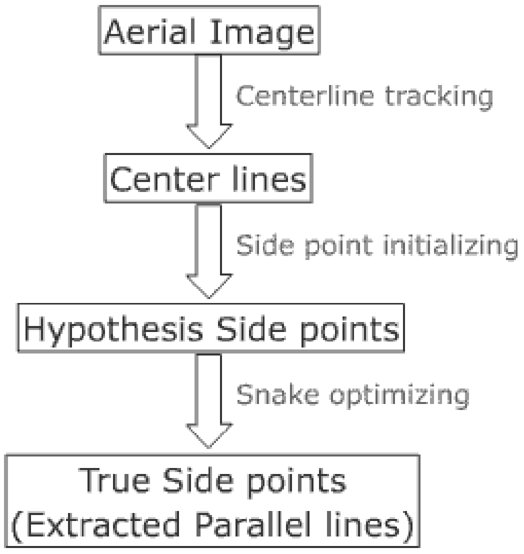


Figure 3.1: Diagram of the strategy

3.2.1 Centerline tracking process

The centerline tracking process begins with edge detection as shown in Figure 3.2. Edges are input to a distance transform [10] to convert a binary edge image into a distance image. A distance image contains intensity levels showing the distance to the closest foreground boundary [10]. The distance transform is illustrated in Figure 3.3.

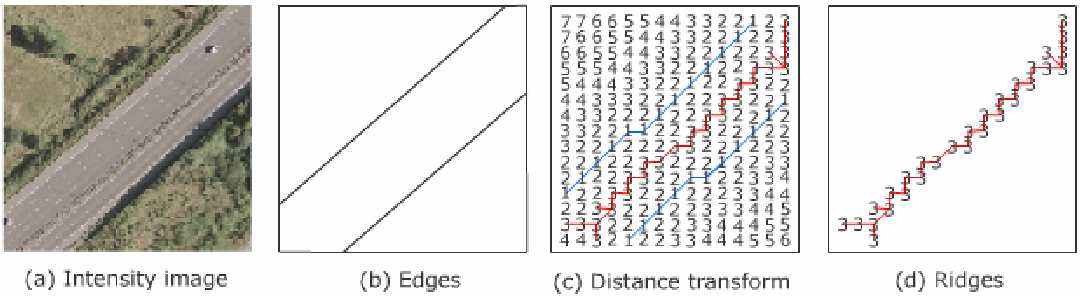


Figure 3.2: Line tracking process

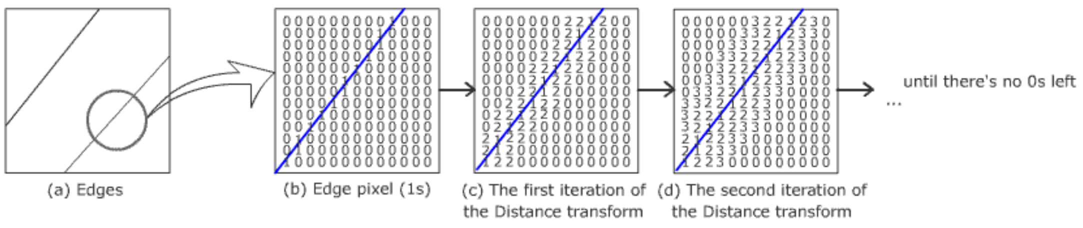

(a) Edges
(b) Edge pixel (1s)
(c) The first iteration of the Distance transform
(d) The second iteration of the Distance transform
until there's no 0s left
...

Figure 3.3: Distance transform

From the binary edge image, the distance transform considers only zero pixels. If a zero pixel has at least one adjacent non-zero neighbour, the zero's value will be changed to be one greater than that of the smallest neighbour's in the next iteration. All the zero pixels are checked in each iteration. It continues changing any zero pixels until all of them are changed. The transformation creates ridges, so called skeletons, between edges. A ridge pixel is a transformed pixel where the value changes from low to high and from high back to low again in any direction at least one direction [16].

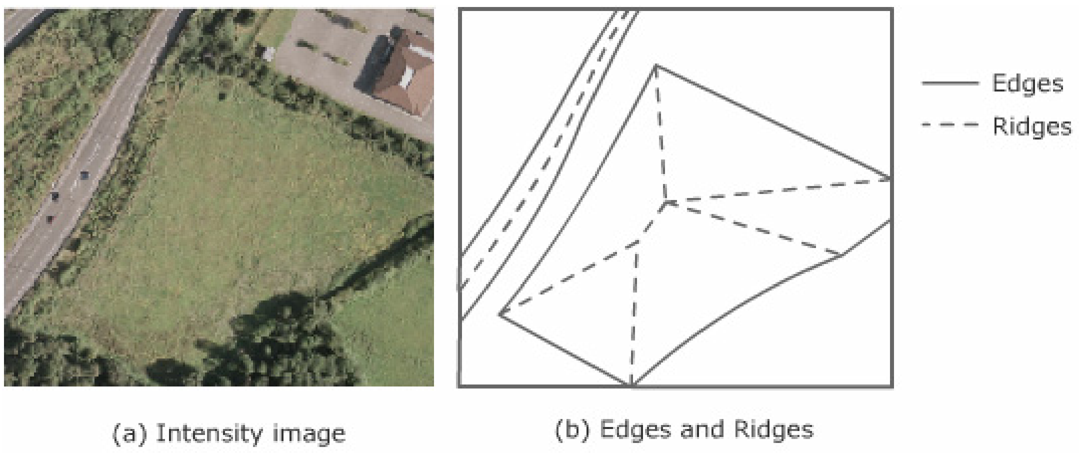


Figure 3.4: Ridges in an open area

Hence, the distance transform tracks zero pixels and increases the pixels' values, symmetrically from both sides of parallel edges. Theoretically, the ridge should be located exactly in the middle of the two edges. Connected ridge pixels are, therefore, the centerline between two parallel edges. Some ridges are however not on the centerlines. Ridges in open areas, for instance, are not a target centerline as shown in Figure 3.4. But, these pixels are so far away from edge pixels; their transformed values are very high. Consequently, this characteristic can be used to decide whether a ridge should be accepted as a centerline or not. In addition, some ridges are too low to be a centerline because of unnecessary noise or texture pixels in intensity images. The unwanted ridges caused by noise usually are short; they can be filtered out easily using length constraints. Additionally, most noise ridges can be removed by the thresholds, which allow the program to accept only the ridges having values in a valid pass range. The threshold's values depends largely upon the width of targeted parallel lines. If the parallel lines are rather wide, the thresholds must be defined in a high range and vice versa. A noise ridge, which has a value in the same range as a true ridge, cannot be removed by means of the mentioned constraints. The noise ridge is normally caused by significant texture on targeted features such as road markings. Figure 3.5 shows a filtered ridge image. Although most noise ridges have been removed, there are still some ridges to be deleted.

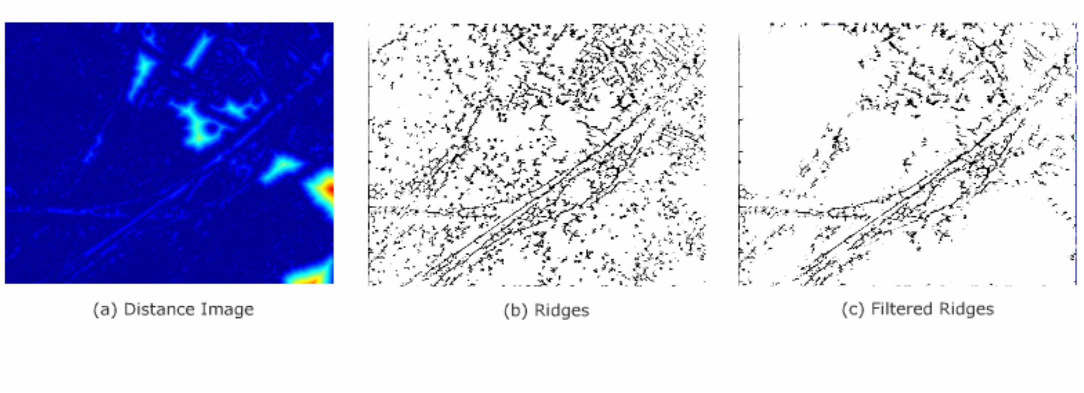


Figure 3.5: Filtered ridge image

As can be seen, the remaining noise mainly appears in irrelevant areas e.g. forest, fields, and wasteland. This noise can be easily filtered out using color constraints in the first place. This method is highly effective to reduce such noise; however, the selection of the target color has to be verified by a human operator in order to achieve a certain degree of precision. To obtain the initial information, the program will simply request the user to select color samples in the target area and creates a color filter based on the information. Nevertheless, the color filters may obtain holes caused by any objects which have significantly different colors such as cars, shadows and road markings. To amend the holes and remove salt noise, morphological operators, dilation and closing, are applied to the filters as shown in Figure 3.6.

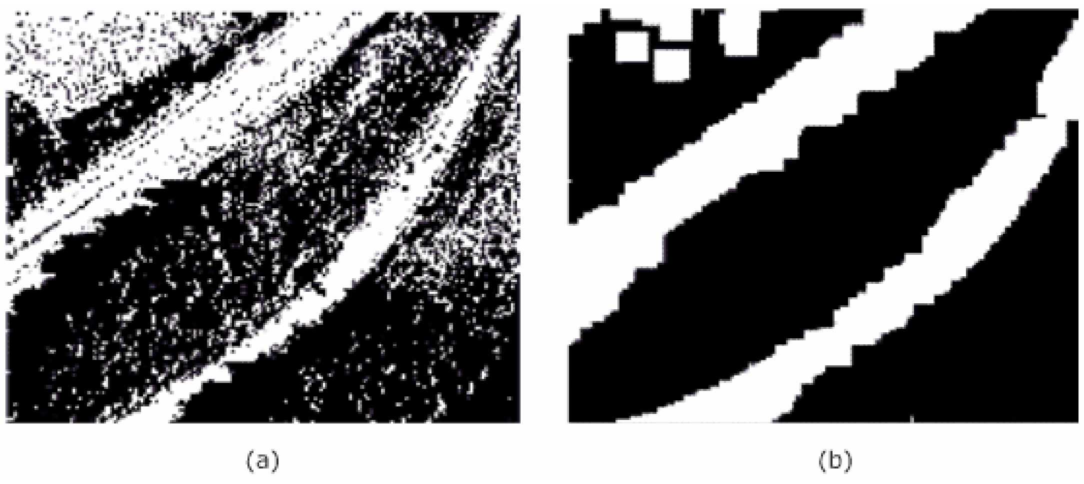


Figure 3.6: Fill holes and remove salt noise in the color filter in (a) using dilation and closing operator

The color filter must be integrated into the centerline tracking process (Figure 3.2) to screen irrelevant pixels in RGB mode. The most effective position for the integration was identified by a series of experiment undertaken previously. The experiments demonstrated that to apply the filter at the position (a), (b) and (c), one of which would have given the best ridges (Figure 3.7). The identified position, somehow, had not given the expected result because unwanted ridges had not been removed. A hypothesis is that the photometric constraint would give a better result if it is used as the input of the edge detection process.

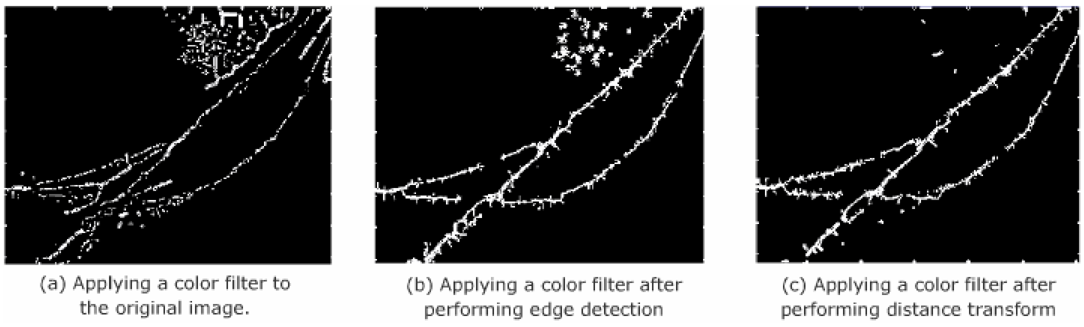
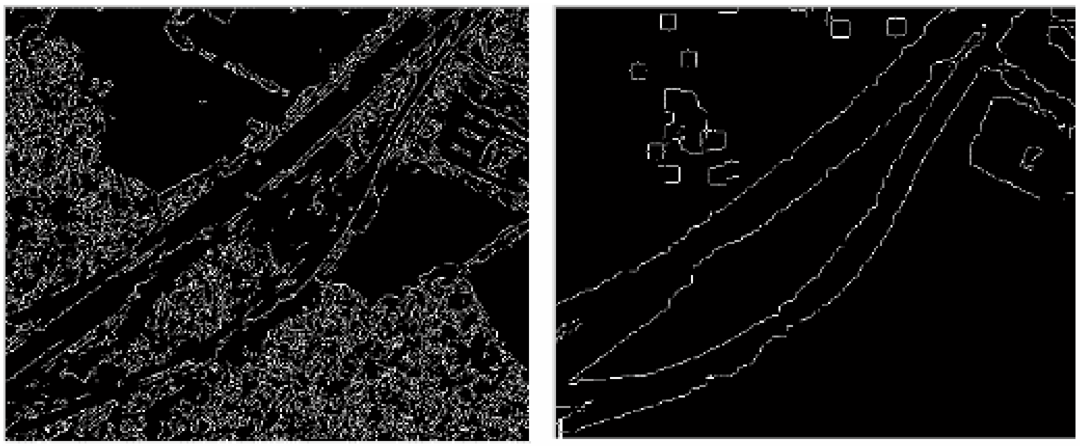

(a) Applying a color filter to the original image.
(b) Applying a color filter after performing edge detection
(c) Applying a color filter after performing distance transform

Figure 3.7: Ridge images after applying color filter to different places in the main process of centerline tracking

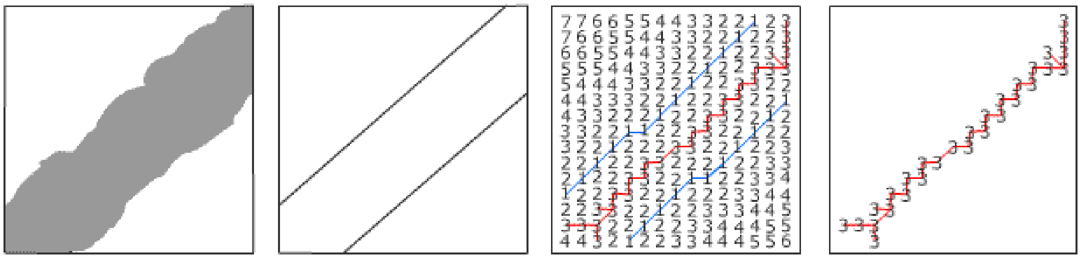
Because the shape of the color filter is closely similar to the target feature shape, the color filter can be used as a representation of focused areas. The color filter is, therefore used before the edge detection instead of an intensity image. The detected edges are almost noise-free as shown in Figure 3.8. Therefore, the new entire centerline tracking process is illustrated in Figure 3.9.



(a) Detected edges from an original image

(b) Detected edges after a color filter

Figure 3.8: Edges from a color filter



(a) Color filter (b) Edges (c) Distance transform (d) Ridges

Figure 3.9: Line tracking process with a color filter

This process produces rather jagged centerlines. Preprocessing procedures are required to remove spurs and track consecutive pixels. The preprocessing units will be discussed in section 3.4.

3.2.2 Side point initialization

The side point initializing process requires a referent centerline. The unmodified centerline contains a large number of pixels, which causes high computational expense. The initializing process creates a series of control points by selecting represented-pixels. As a result, the modified centerline contains only the represented pixels that act as the control points. This method reduces the computational expense in the snake optimizing process. This study found that the method reduced the calculation time from 240 minutes to 30 minutes while the accuracy remains. The control points are shown in Figure 3.10 (b). A control point is located every four pixels along the consecutive pixels of the centerline.

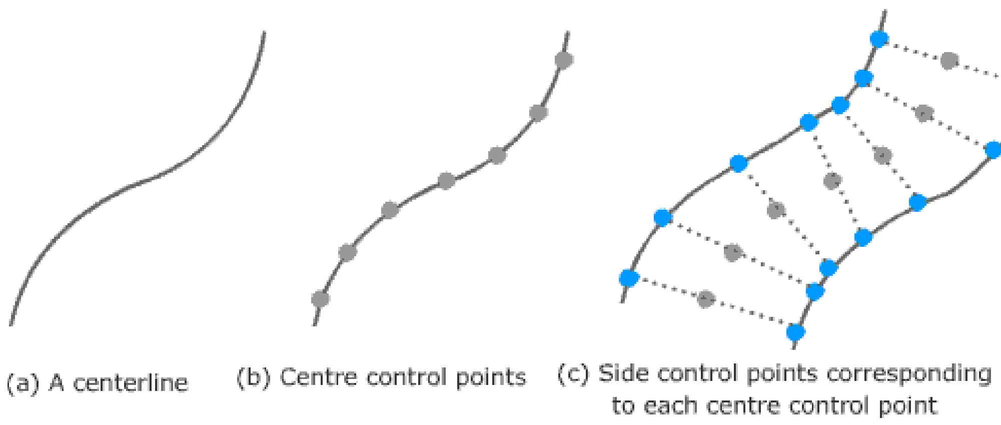

(a) A centerline
(b) Centre control points
(c) Side control points corresponding to each centre control point

Figure 3.10: Control points from a centerline. (a) centerline, (b) control points, (c) side control points

The strategy of side point initialization is to project each center control point in the perpendicular direction to establish references called the side control points at the two side curves. The side control points are located at high gradient magnitude positions from the centerlines as shown in Figure 3.10 (c). The process performs a gradient transform and calculates a perpendicular direction to the centerlines at each center control points. More details will be discussed in the implementation section.

The color filter discussed in the centerline tracking process limits a search space for detecting the initial side control points. The search space is actually the width of the color filter. The width is slightly bigger than that of true parallel lines because of the effect of dilation. This suggests that the true sidelines are closer to the center than the edges of color filters. It is, therefore, not necessary to calculate the gradient magnitude of a pixel having a further distance from a center control point than the width of the color filter.

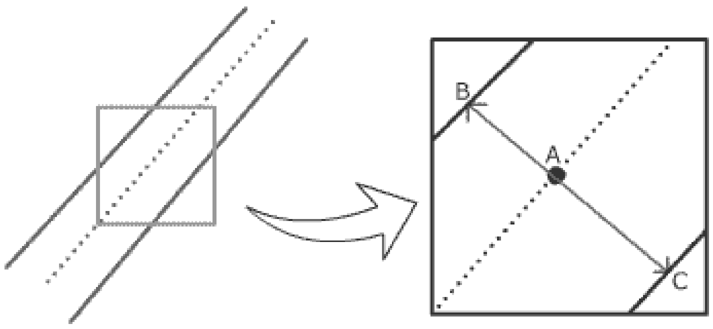


Figure 3.11: Edges of a color filter limits the search space

In Figure 3.11, for example, B and C are the points on the edges of the color filter in the perpendicular direction to the center control point A. To locate the initial side control points on the left side of A, the initializing process calculates the gradient magnitude for every pixel from A to B and locates the left control point at the pixel giving maximum gradient magnitude. The right control point is located in the same way. As a result, the left and right control points can be located in the positions shown in Figure 3.12 (b). Since the constraints depend only on the image gradient, it is vulnerable to occlusions such as shadows. To improve the initialization, the side points are relocated to new positions where the distance from side points to the centre point is constant on each side. The constant distance is called left width and right width. The left width in Figure 3.12 (c), for instance, is the median of AE, BF, CG and HD. Because the median ignores the peak noise displayed in Figure 3.12 (a) and (b), it always gives a reasonable width for both sides. The side control points are then initialized as shown in Figure 3.12 (c). The initial points are now located close to the true sidelines. The side control points will then be optimised to better fit the real points by the snake optimizing process.

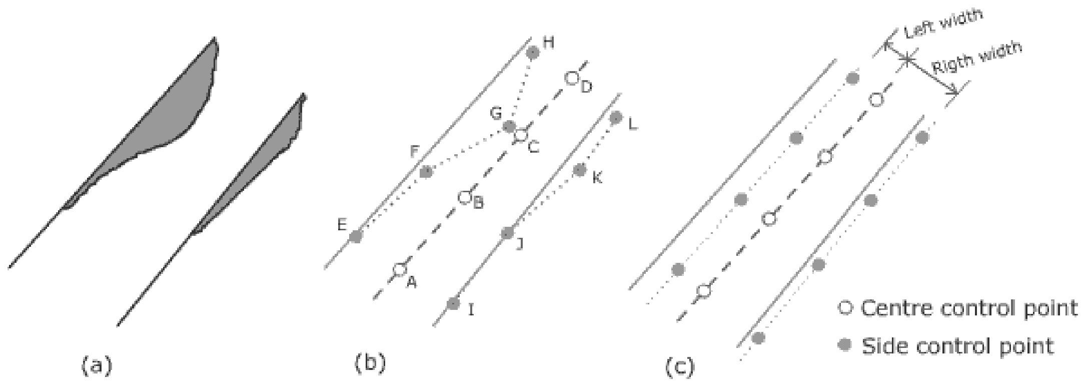


Figure 3.12: Initial side control points

3.2.3 Snake optimizing process

A Snake is a curve consisting of control points. In the original Snake [29], it is completely specified by the number and coordinates of its control points. The optimizing process deforms snakes to shapes where both geometric and photometric constraints are balanced. The two constraints are embedded in the optimization in terms of energies. The energy functions and behaviours of snakes are discussed in the theory section. Three optimizing techniques have been tested and the results are as follows.

The first technique was implemented following the suggestion in [17]. Energies of adjacent neighbors of a control point were calculated and the control point was forced to move to the neighbor giving the lowest energy. This was a simple and straightforward technique. It however could easily be stuck in a local minima. Thus, the search space was extended to avoid the local minima. The optimizing process searched the lowest energy in a larger area such as in a 5x5 square rather than 3x3. However, the size of the search space needed to be adjusted carefully. If the size was too large, it would take a lot of time to calculate energy at every position in the search space and the lowest energy might actually belong to another snake as shown in Figure 3.13. If the size were too small, the local minima problem would occur.

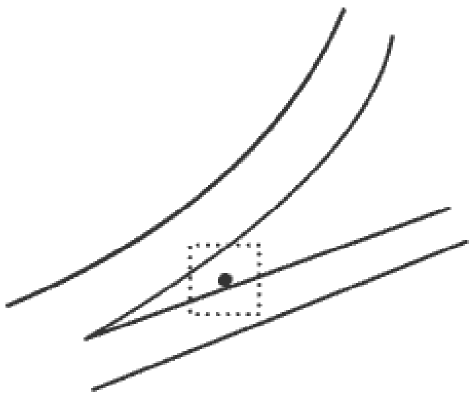


Figure 3.13: Scope of search space overlaps two edges

The second technique introduced a navigating matrix. The matrix guided the search of the optimizing process. To reduce the size of the search space in the first technique, the navigating matrix limited the pixels to which a control point could move. As shown in a navigating matrix size 7x7, the center of the matrix is a control point. In the optimization, only the edge pixels (shaded) of the navigating matrix were considered and the control point would be moved to one of those pixels only. This technique gave two advantages. First, since the size of the search space was reduced, it then reduced computational-time consumption. Second, if the size of the navigating matrix kept shrinking during the optimizing process, the search space would get smaller. As a result, the move that the optimizing process could make had been gradually changed from a coarse scale to a fine scale. This would avoid overshooting when the snake became close to the optimum positions.

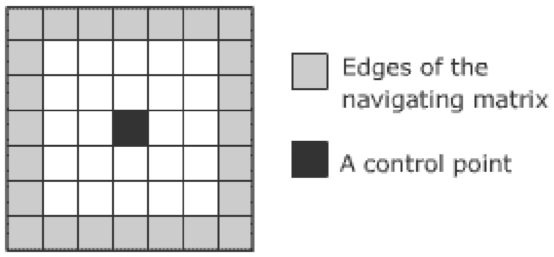


Figure 3.14: A 7x7 navigating matrix

This technique, however, still had a problem in the initial size of the search space that occurred in the first technique. The navigating matrix could erroneously cover pixels in another feature if its size were too big. Moreover, if a control point moved to a wrong position in the beginning of the process when the navigating matrix was still big, it would not be able to move back to a correct position when the navigating matrix became smaller because the correct control point would not exist within the navigating matrix anymore.

The last technique uses the fact that the distance between the two snakes on each parallel line feature should be kept constant. The standard deviation of the width of snakes is the included in the energy function to penalize the snakes when their moves cause an inconstant width. This technique limits the search space by forcing the optimization to search the pixels in the perpendicular direction to the snake curve only as shown in Figure 3.15. This project selected this technique because of its simplicity and efficiency.

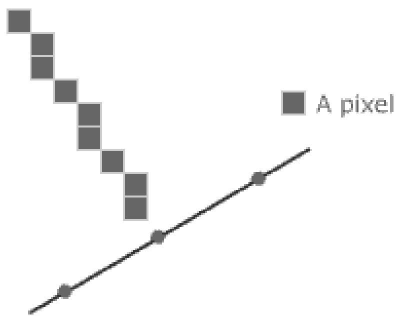


Figure 3.15: The pixels in the perpendicular direction to the snake curve

3.3 Theory: Snakes

Active Contour Models, well known as Snakes, extract contour features, called snake curves, in input imagery. Both photometric and geometric constraints are embedded in the active contour models to guide the search in an optimizing process. During the process, a snake curve is optimized locally close to its initial position to satisfy both constraints. The geometric constraints control smoothness and stiffness of snake curves and the photometric constraints fit the curves [23] onto target features exploiting image properties of the features. Snake techniques optimize curves using energy functions that can be analogous to physical systems [11].

The energy function for snakes is divided into three parts as shown in equation (3.1)

$$Esnake = Eimage + Einternal + Eexternal \qquad (3.1)$$

The image energy responds to photometric constraints. Image properties such as intensity or image gradient are included in the image energy. The internal energy represents geometric constraints. Properties of curves such as length and curvature are included in the internal energy. The external energy represents other constraints such as a gravity force pulling a curve in the direction of the gravity. In this extraction, the external energy is the standard deviation of the width between a pair of curves. It is used as a penalty function to keep the curves parallel. These energies make snakes flexible and adaptive for various feature shapes [23] by including suitable properties of the features to the energies or even suppressing some energies to satisfy the character of the features.

The original representation of snakes introduced in [29] is shown below.

$$v(s,t) = (x(s,t),y(s,t)) \quad 0 \le s \le 1 \qquad (3.2)$$

'**v**' is a snake curve in two dimensional space. '**s**' is proportional to the length of the curve and '**t**' is the current time in optimizing process. '**x**' and '**y**' are the curve's coordinates [23]. The total energy of a curve is calculated. The curve is then deformed to a new shape according to the direction and magnitude of the sum of an image force, an internal force and an external force. The image force pushes the curve to image features while an internal force ensure the smoothness of the curve [23].

Since the representation of snakes presented in [29] is a continuous form so are the form of the provided equations. In a computer system to move the snake curves to a desired position, a discrete form of the Snake representation and energy equations is required. The representation of snakes is discretized into '**n'** control points, (as discussed in 3.2.3). The discrete representation of snakes is shown in (3.3) and the continuous and discrete equations of each energy are discussed below.

$$v(i) = (x(i),y(i)),\quad i = 1,2,3,\ldots,n \quad \text{where curve } v \text{ consists of } n \text{ control points} \qquad (3.3)$$

The total energy in (3.1) can be rewritten in (3.4) where a, b and c are coefficients used to adjust the balance of the energies.

$$E(v) = aEimg(v) + bEint(v) + cEext(v) \qquad (3.4)$$

[23] provides both continuous and discrete equations for the image energy as shown in the equation (3.5) and (3.6), respectively.

$$Eimg(v) = -\int_0^1 P(v(s,t))\, ds \qquad (3.5)$$

where P is a function with high value proportional to the interest features.

Converting to discrete form:

$$Eimg(v) = -\sum_{i=1}^{n} P(v(i)) \qquad (3.6)$$

where curve v consists of n control points

The function P can include both the magnitude of the image gradient and image intensity at a control point. In this study, only the magnitude of the image gradient is included.

[23] also provides the equation for the internal energy in a continuous version (3.7). The discrete version (3.8) is given by [17].

$$Eint(v) = \frac{1}{2}\int_0^1 \alpha(s)\, |\partial v(s,t)/ \partial s|^2 + \beta(s)\, |\partial v^2 (s,t)/ \partial s^2|^2\, ds \qquad (3.7)$$

where α(s) and β(s) are arbitrary functions controlling the elasticity and stiffness of the curve. They can be specified as constant values.

$$Eint(v) = \sum_{i=1}^{n} (\alpha|v(i) - v(i-1)|^2 + \beta|(v(i+1) - v(i)) - (v(i)-v(i-1))|^2)$$

$$= \sum_{i=1}^{n} \alpha((x(i) - x(i-1))^2 + (y(i) - y(i-1))^2) + \beta((x(i+1)-2x(i)+x(i-1))^2 + (y(i+1)-2y(i) + y(i-1))^2) \quad (3.8)$$

A problem occurs when the internal energy of the head and the tail control points (i=1 and i=n) of a curve are calculating as can be seen in the equation (3.8). For both control points, the approximations of the internal energies can be calculated by the equations proposed by Laptev [23]. This study adapted the equation in accordance with the requirement of the problems.

The external energy is rather arbitrary. To calculate the external energy, firstly the representation of the snake curve used in this paper has to be introduced. Since the target feature is a parallel line, a snake curve consists of two sets of control points. One set is for the right side; the other is for the left side as shown in the figure below.

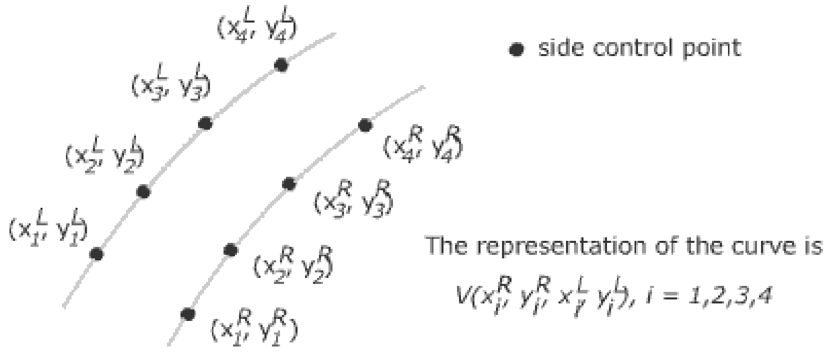


Figure 3.16: Representation of snakes

The external energy in this paper is used to control the width of snake curves to be constant. The external energy is the standard deviation of the width of a curve as shown in the equation (3.9).

$$Eext = STD(\sqrt{((x_i^R - x_i^L)^2 + (y_i^R - y_i^L)^2)}) \quad (3.9)$$

Consequently, the internal energy and the image energy have to be changed to be the sum of the energy of both sides as in shown in the equation (3.10) and (3.11) below.

$$Eint = EintR + EintL \quad (3.10)$$

$$Eimg = EimgR + EimgL \quad (3.11)$$

The snake optimizing process searches for low snake energy positions close to an initial curve. The curve is then deformed until the optimizing process cannot give a lower energy in its surrounding area or the number of the loop limit is reached.

3.3 Implementation

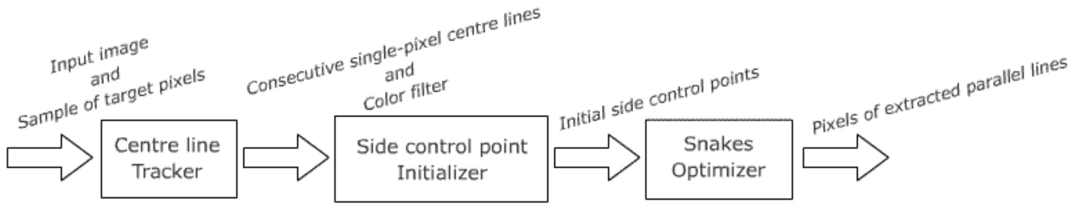

Input image
and
Sample of target pixels
Centre line Tracker
Consecutive single-pixel centre lines
and
Color filter
Side control point Initializer
Initial side control points
Snakes Optimizer
Pixels of extracted parallel lines

Figure 3.17: The constant parallel line extraction process

The entire process of the constant parallel line extraction is shown in Figure 3.17. It will be discussed independently below.

3.4.1 Centerline tracker

The process of the centerline tracker is depicted in Figure 3.18. The tracker receives a color image and a sample of target pixels as its input and it produces consecutive single-pixel centerlines as its output. Two output centerlines are displayed in Figure 3.19.

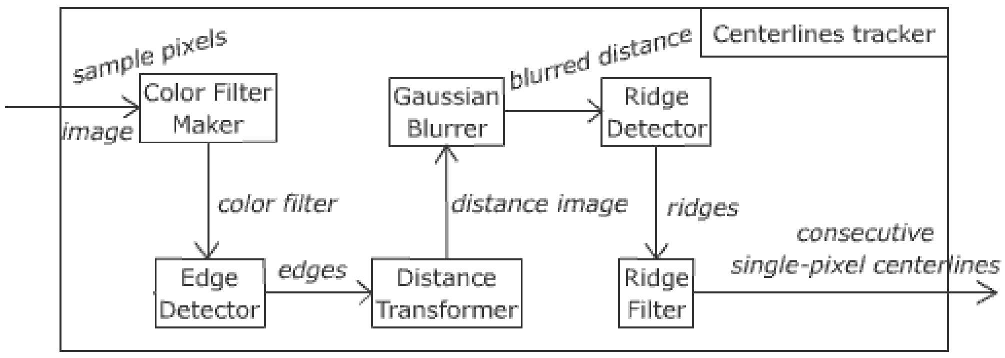


Figure 3.18: Centerline tracker

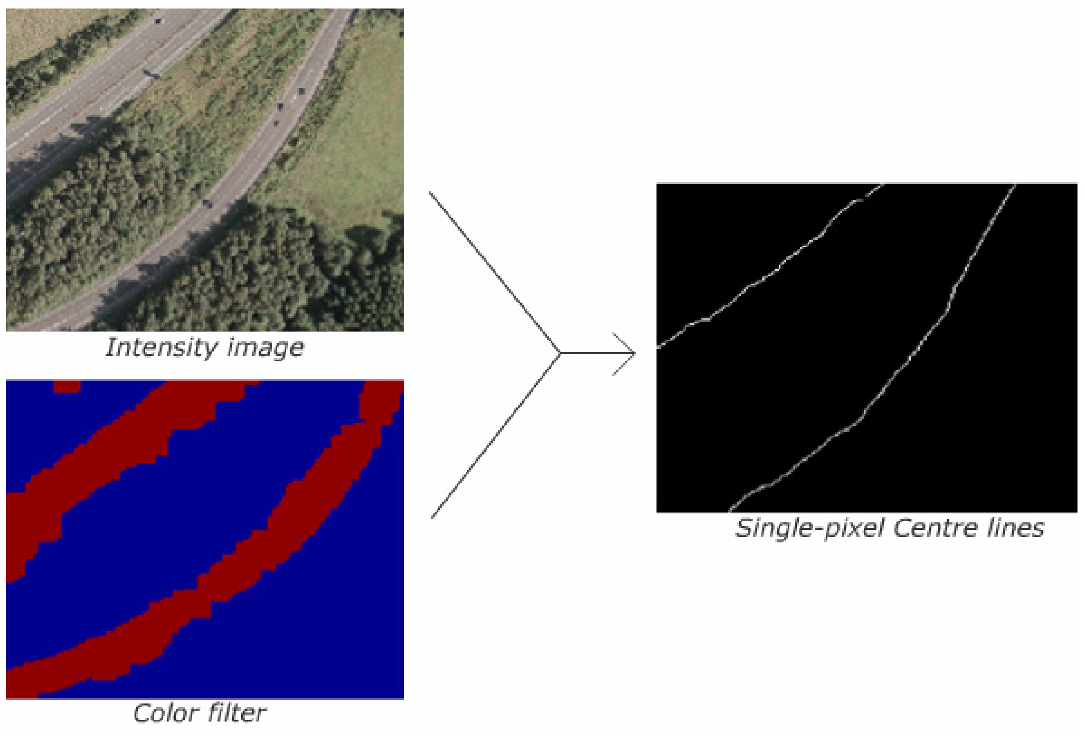


Figure 3.19: Two output centrelines from the tracker

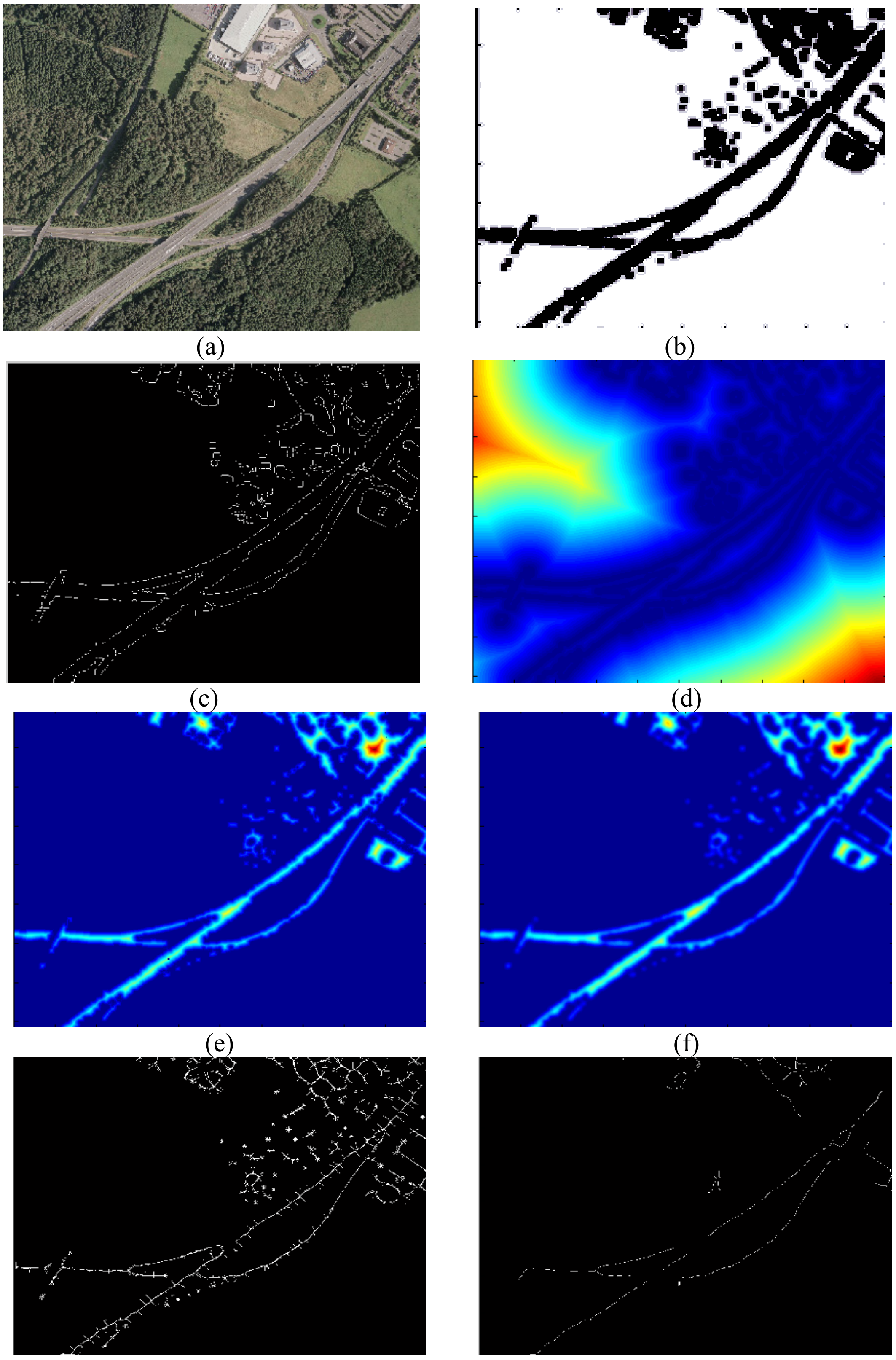

(a) (b)
(c) (d)
(e) (f)
(g) (h)

Figure 3.20: Outputs from each sub process of the centreline tracker

Figure 3.20 shows the output of each sub process of the centerline tracker. To begin with, the color filter maker calculates the mean intensity in RGB of the sample pixels and filters out the area having significantly different intensity from the mean. The tolerant intensity used in this project is (+/–) 30 for the all three colors. The remained area is a color filter. A color filter is shown in Figure 3.20 (b). The edges of the color filtered raw image are detected by edge detection; the edges are shown in Figure 3.20 (c). This edge detector is the same as the one discussed in chapter 2.

MATLAB provides a built-in function for the distance transform, bwdist(). The distance transform assigns a number to each pixel in the edge image. The number is the distance between that pixel and its nearest nonzero pixel. [14]. The distance image is shown in Figure 3.20 (d). Figure 3.20 (e) illustrates the distance image after applying the color filter to remove irrelevant area. The different colors in the image represent the different distance values. The brightness of the color is directly proportional to the distance value. In Figure 3.20 (f), the distance image is then blurred to suppress sharp edges between two different distance values. The sharp edges would cause messy branches in the ridge image. The ridge detector first filtered out too high and too low distance values. It then finds ridges from the remaining pixels. A ridge pixel is a pixel where the distance value changes from high to low and from low back to high in some direction. A sharp edge can be incorrectly detected as a ridge if its distance value is in the same range as that of the ridge. The suppression of the sharp edges significantly reduces the length of sharp-edged ridges. Therefore, the short sharp-edged ridges can be filtered out using a length constraint. The detected ridges are shown in Figure 3.20 (g) and those that are long enough are shown in Figure 3.20 (h).

The ridges (Figure 3.21 (a)) are thinned (Figure 3.21 (b)) using a thinning operator [15]. The thinning operator, however, cannot remove groups of pixels at the junctions as shown in Figure 3.21 (b). To remove those junctions, pixels having only two neighbors are selected; the rest are removed as shown in Figure 3.21 (c). Then, any two closest tips are connected using the Bresenham algorithm [12]. Up to this point, a single-pixel ridge is produced as shown in Figure 3.21 (d). To remove branch noise, the junctions are detected and removed, then the length constraint is applied to delete too short lines. The remaining pixels are then reconnected using the Bresenham algorithm as shown in Figure 3.21 (e)-(g).

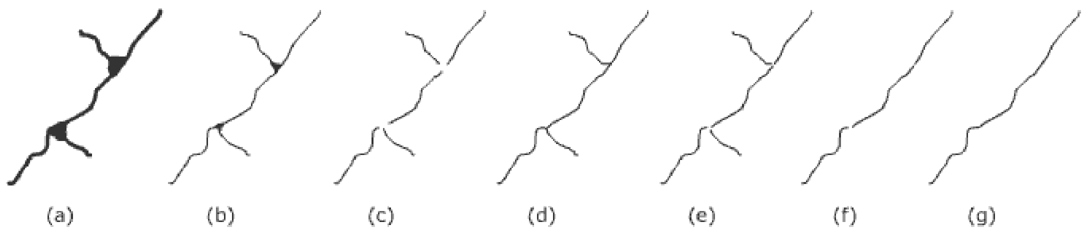


Figure 3.21: Ridge filtering

3.4.2 Side control point initialization

In the perpendicular direction to the centerlines, the width finder detects edges within the color filter to find the approximate width of the parallel line features. The approximate width is then used to guide the search for the initializing process of side control points as shown in Figure 3.22. The initialization locates initial control points on both sides of the centerlines within the range of the approximate width.

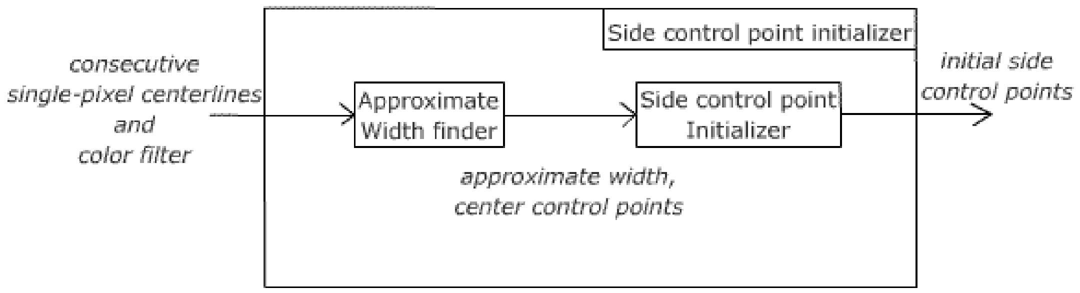

consecutive single-pixel centerlines and color filter
Side control point initializer
Approximate Width finder
Side control point Initializer
approximate width, center control points
initial side control points

Figure 3.22: Side control point initializer

To begin with, the width finder defines centre control points on the centerlines. A control point is located every four pixels along the consecutive pixels of the centerline. The width finder calculates the perpendicular direction to the centerline at each control point employing two vectors that point to its adjacent neighbors, U and V as shown in

Figure 3.23. The perpendicular direction is the direction of vector W in Figure 3.23. The vector W has θ/2 degree from U where θ is the angle between U and V. θ can be calculated effortlessly from the dot product of U and V. Then, connected pixels are created using the Bresenham algorithm [12] in the direction of vector W to find the edges of the color filter as shown in Figure 3.24. The edge is the furthest position within a limit threshold where pixels change from 1 to 0 in order to avoid occlusion pixels on the color filter as shown in Figure 3.25.

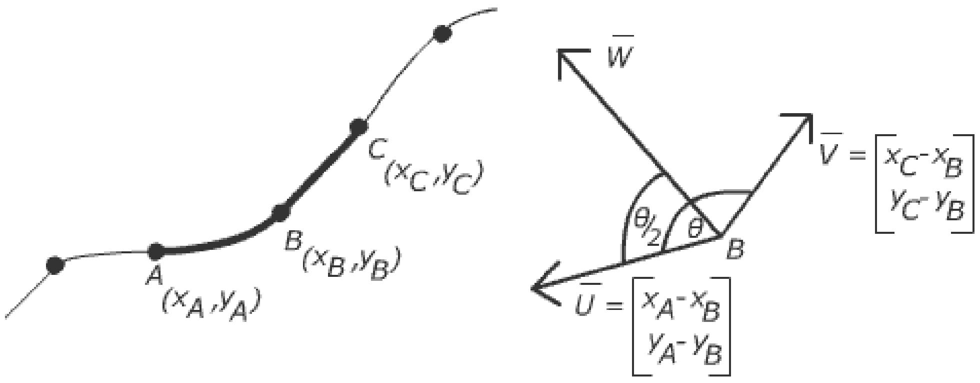

C
(x_C, y_C)
B
(x_B, y_B)
A
(x_A, y_A)
$\overline{W}$
$\overline{V} = \begin{bmatrix} x_C - x_B \\ y_C - y_B \end{bmatrix}$
$\theta/2$
θ
B
$\overline{U} = \begin{bmatrix} x_A - x_B \\ y_A - y_B \end{bmatrix}$

Figure 3.23: The perpendicular direction to the centreline

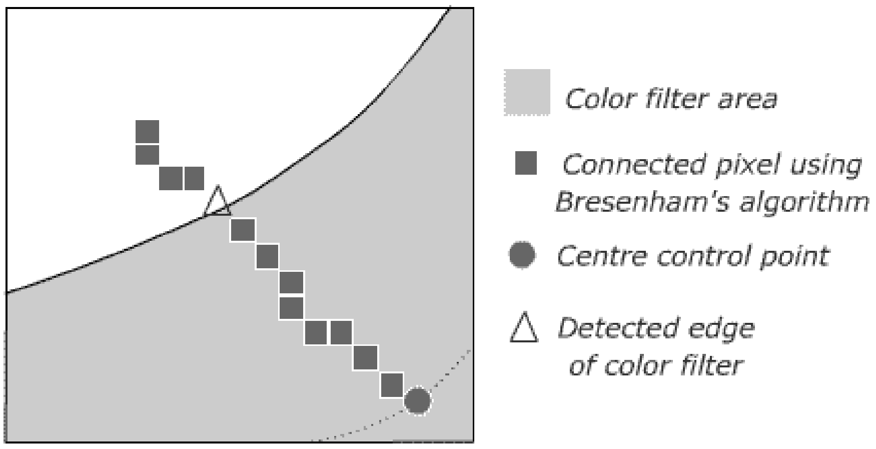

Color filter area
Connected pixel using Bresenham's algorithm
Centre control point
Detected edge of color filter

Figure 3.24: Detect the edge of a color filter

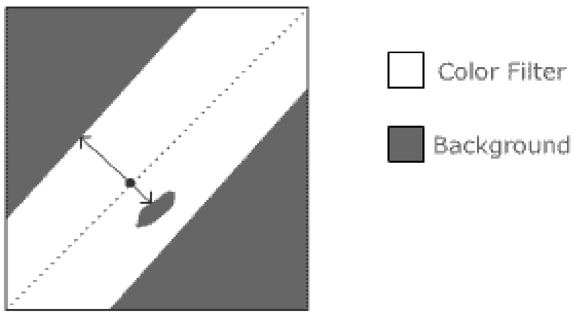


Figure 3.25: Occlusion in a color filter

The side control point initialization required the approximate width, which is a distance from the center control point to the edge. Instead of using the coordinate measuring system, the pixel is used as a unit of measurement to identify the distance. The pixel unit has an advantage over the coordinate measurement because the pixels are already delineated by Bresenham's algorithm, in other words, the pixels are already positioned in the perpendicular direction. Therefore, the pixel unit identifies not only the distance but also the location. The approximate width at each control point for both sides is calculated iteratively and then sent to the initialization.

For each center control point, the side point initialization locates two initial side control points at the maximum magnitude of the image gradient within the approximate width and in the perpendicular direction to the centerline as shown in Figure 3.26. The image gradient is calculated using gradient() function, a built-in function in MATLAB. It provides the numerical gradient in x and y direction of an input matrix [13]. The magnitude of the gradient is calculated from this.

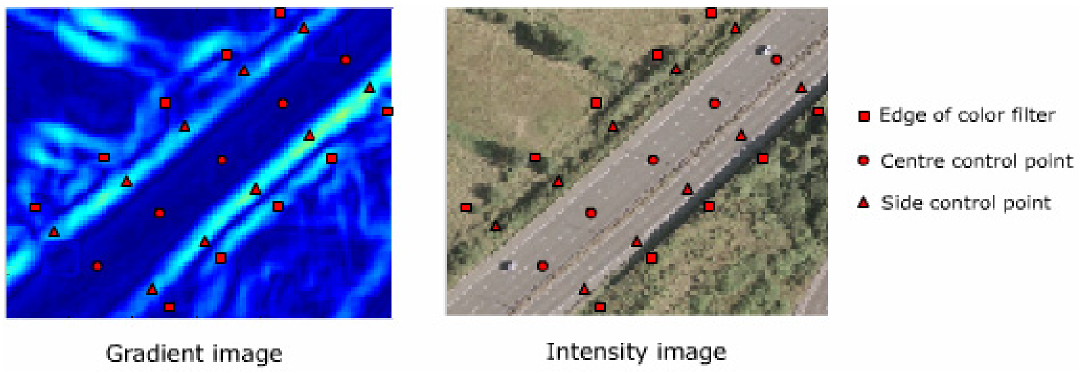


Figure 3.26: Initial side control points

As mentioned in the strategy section, the initial control points on both sides might provide jagged curves because the width of the parallel curve at each vertex could vary significantly. The program relocates the side control points using the median of the distance between any pair of control points. The new curves of the side control points are forced to be parallel by setting the width to the median.

3.4.3 Snake optimization

The snake optimizer receives an initial snake curve, $V(x_i^R,y_i^R,x_i^L,y_i^L)$, $i = 1,2,3,\dots,n$ where **n** is the number of control point on the curve. The curve V consists of two parallel lines. The control points on each side represent each line. For example, x_i^R and y_i^R are the control points on the right side; they are used to specify the right line of the curve. The optimizer also requires the image gradient to calculate image energy.

The optimizing process starts with changing the position of a control point on one side to be one pixel closer to the centerline and within the perpendicular direction. Then the total energy of a curve with the new control point is calculated. If the calculation suggests that the change of the position would lower the energy, the new control point is treated as a candidate control point. The process then returns to change the original control point by moving it one pixel further from the centerline in the same direction then the total energy is test. If this test shows that the energy of this control point is lower than that of the candidate, it will be kept.

The process performs this calculation iteratively for all control points of both sides on the curve. The change that gives the lowest energy is used to update the curve. This process uses the updated curve to repeat the optimization from the beginning. The process iterates until no new move gives lower energy or the limit loop is reached. The last updated curve is considered as the optimum curve.

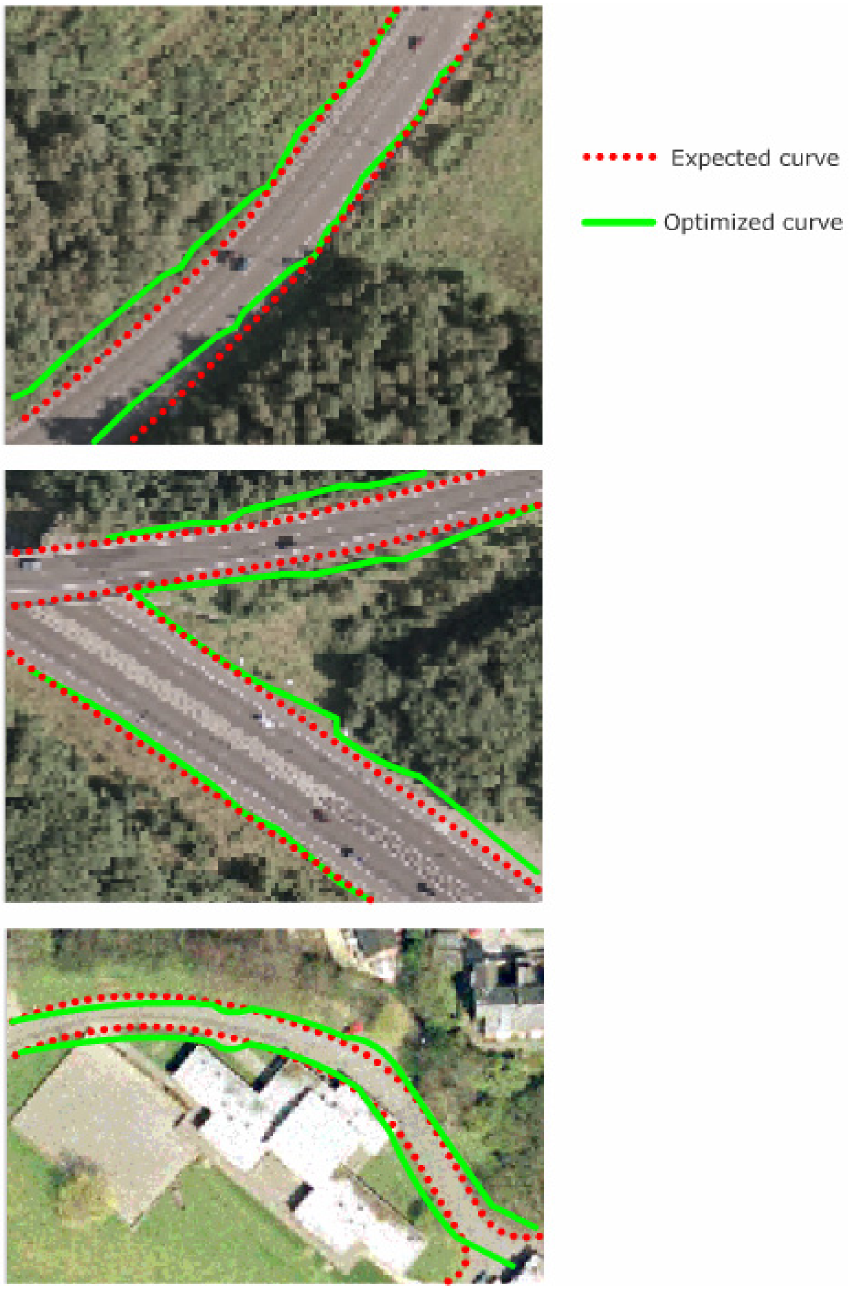


Figure 3.27: Examples of optimized curves

3.5 Experiment

The experiment measures the accuracy of the approach by comparing extracted parallel lines to ground truth. The ground truth is manually generated. It includes significant target features as shown in the figure below. Because the constant parallel line extraction does not have adjustable parameters like the constant rectangle extraction in chapter 2, consequently, the number of experiments is then much smaller than that of the first experiment.

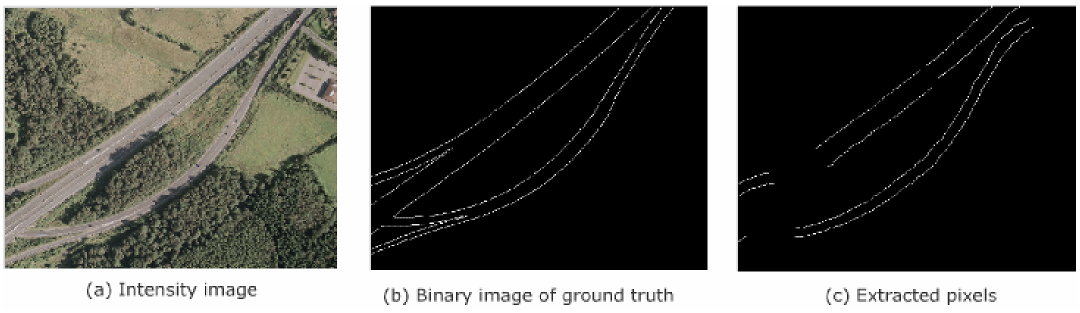


Figure 3.28: Binary image of ground truth

The experiment matches an extracted pixel to a true pixel where the extracted pixel is the nearest pixel to the true pixel and the distance between them is less than a tolerance delta. A true pixel is considered as a false negative when it does not have a matched extracted pixel. An extracted pixel is a false positive when it cannot be matched to any true pixel as shown in the figure below. The results of the experiment are the percentage of matched pixels, false negative, false positive, and the average of the error distance. Since the extraction requires a sample of target pixels from a user, to test the reliability of the methods, three selections of sample pixels are experimented for each input image. The experiment is performed by varying the tolerance delta to examine the accuracy of the extraction and the suitable value of delta. Delta varies from zero to twenty stepping by one. The result of this method is presented in the next chapter.

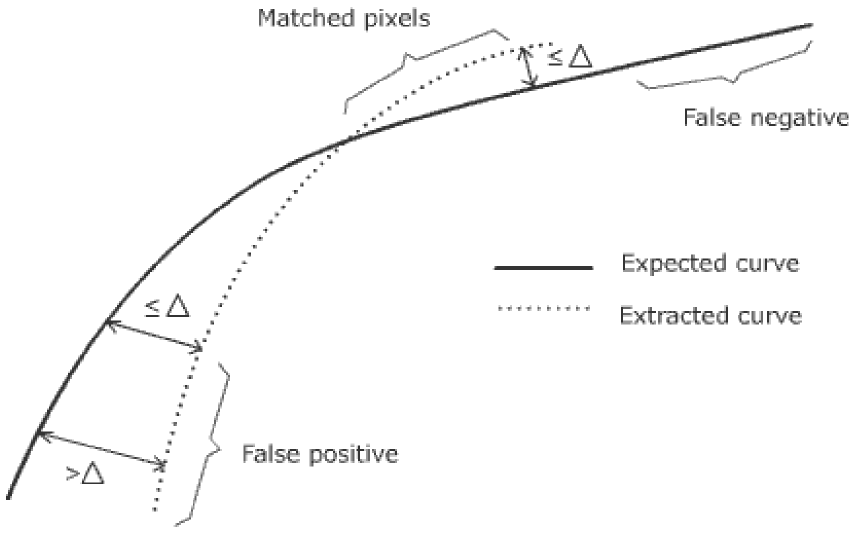
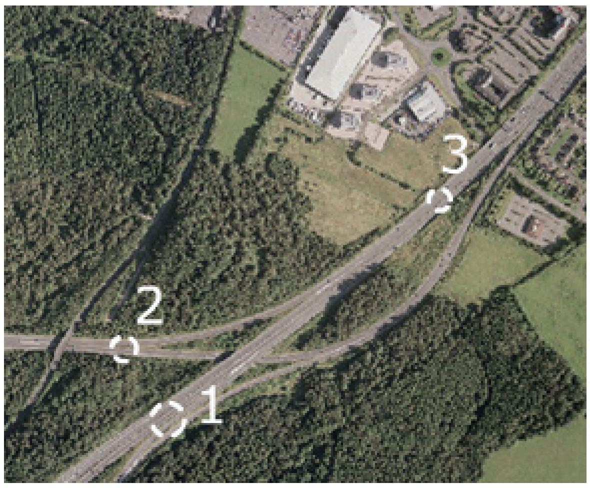


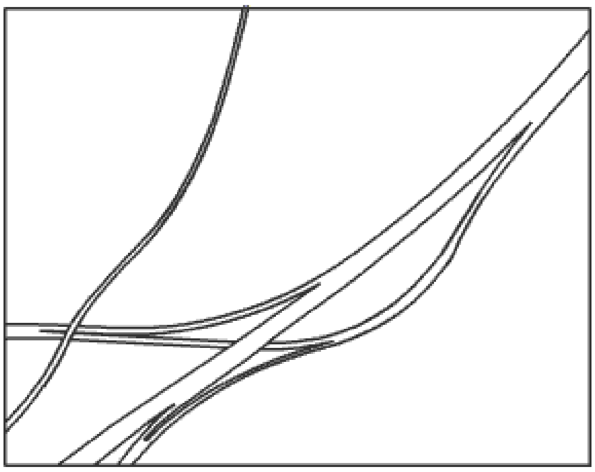
Figure 3.29: Matched pixels, false negative and false positive.

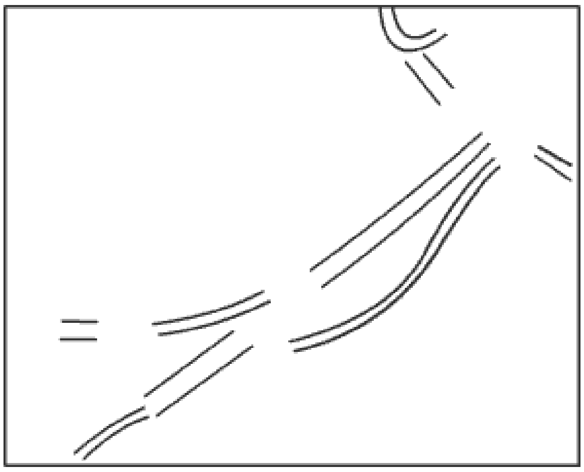
3.6 Result

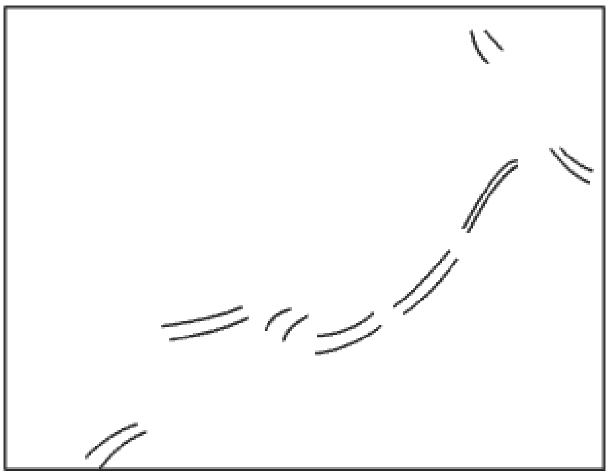
Figure 3.30, 3.31 and 3.32 show the results for ‘m27_aerial_image.tif’, ‘M27_Junction_image.tif’ and ‘Durham_Residential2_image.tif’, respectively. I ran the experiments three times on each image, using different sample target-pixel regions to initialize the colorfilter. The results are shown in the figures below.



(a) Sample areas in ‘m27_aerial_image.tif’


(b) Expected pixels of ‘m27_aerial_image.tif’


(c) Extracted pixels using the sample area 1 (shown in (a))


(d) Extracted pixels using the sample area 2 (shown in (a))

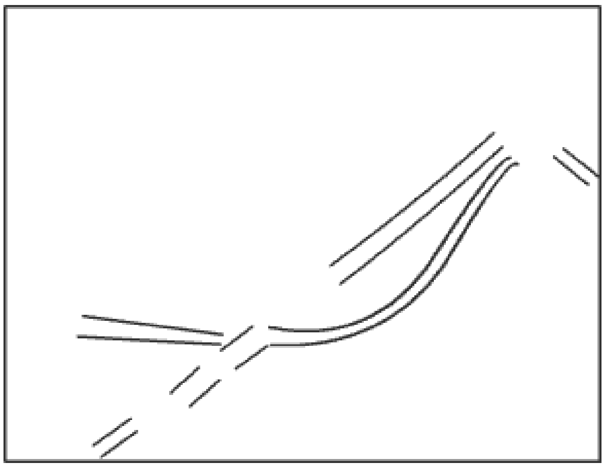
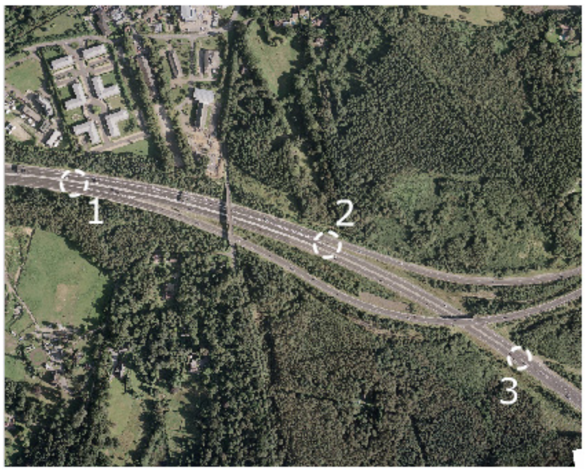
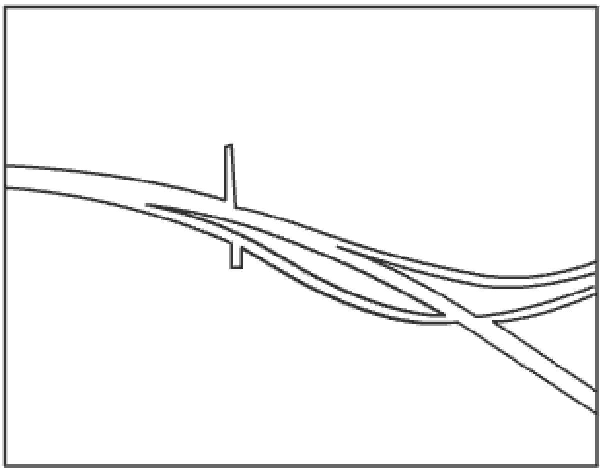
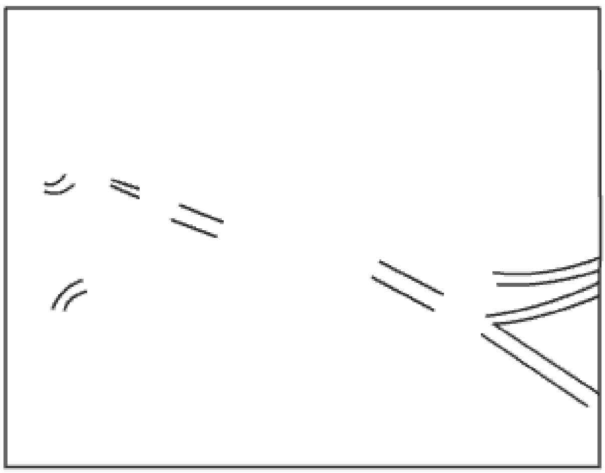

(e) Extracted pixels using the sample area 3 (shown in (a))

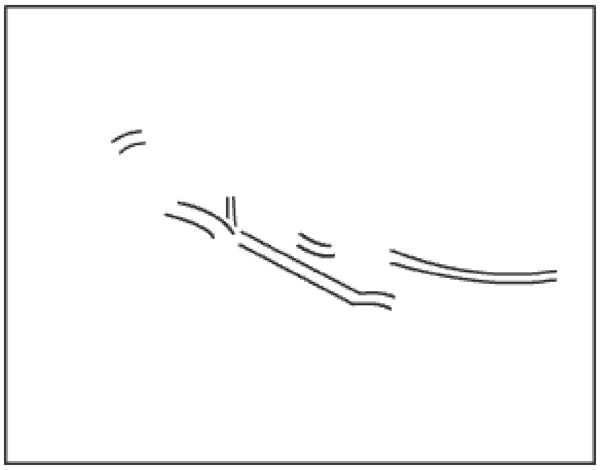
Figure 3.30: The experiment results of ‘m27_aerial_image.tif’



(a) Sample areas in ‘M27_Junction_image.tif’


(b) Expected pixels of ‘M27_Junction_image.tif’


(c) Extracted pixels using the sample area 1 (shown in (a))


(d) Extracted pixels using the sample area 2 (shown in (a))

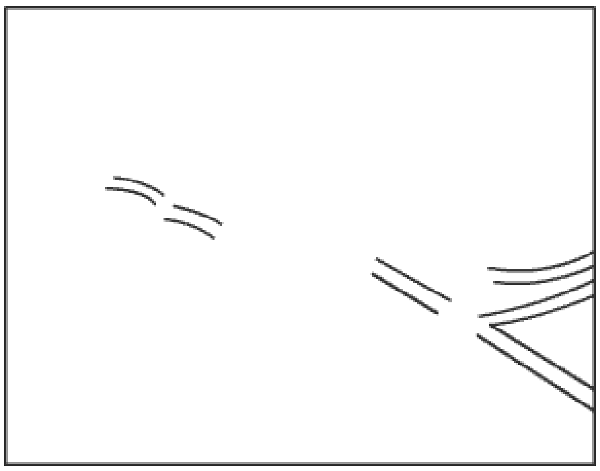
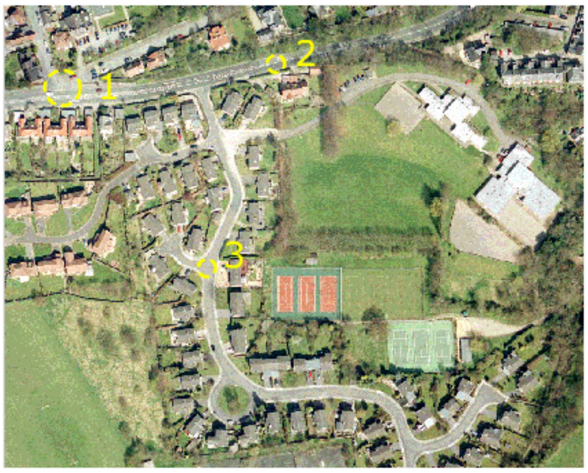
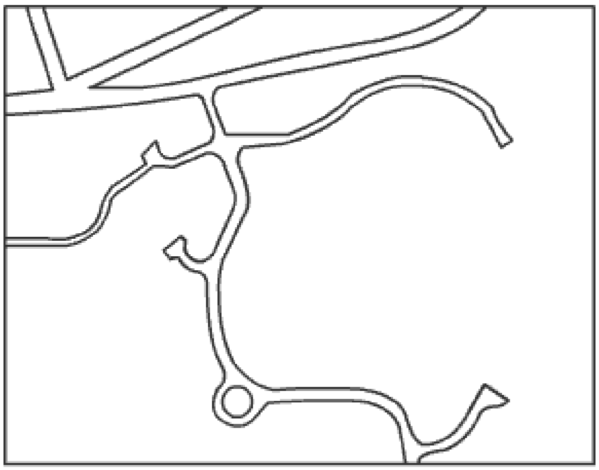

(e) Extracted pixels using the sample area 3 (shown in (a))

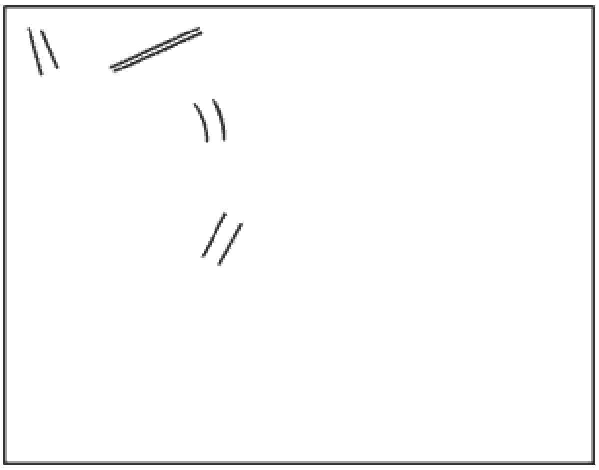
Figure 3.31: The experiment results of 'M27_Junction_image.tif'



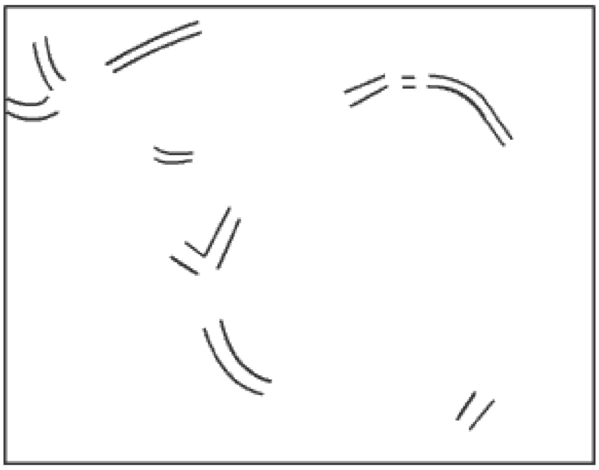
(a) Sample areas in 'Durham_Residential2_image.tif'



(b) Expected pixels of 'Durham_Residential2_image.tif'



(c) Extracted pixels using the sample area 1 (shown in (a))


(d) Extracted pixels using the sample area 2 (shown in (a))

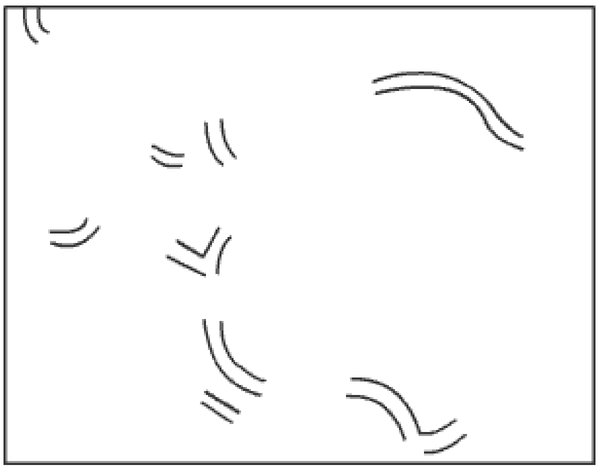
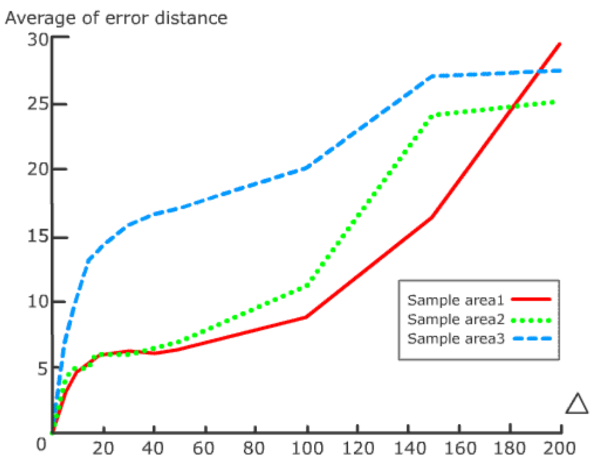
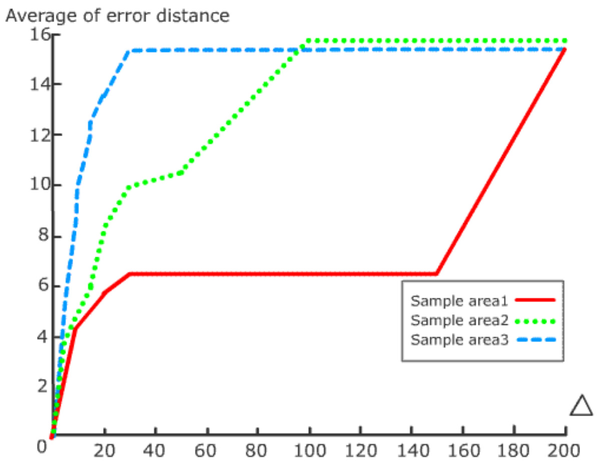

(e) Extracted pixels using the sample area 3 (shown in (a))

Figure 3.32: The experiment results of ‘Durham_Residential2_image.tif’

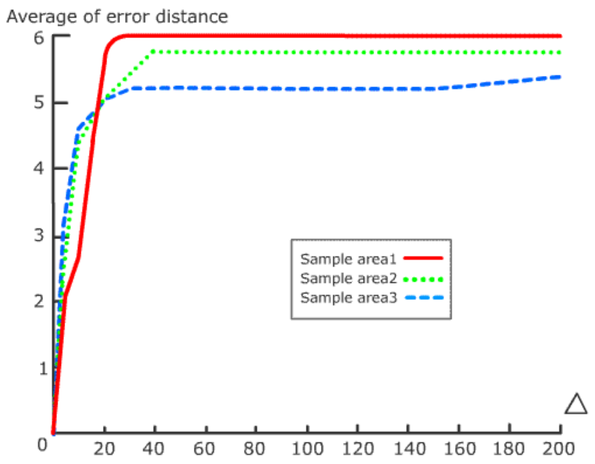
As can be seen from Figure 3.30-3.32, the sample areas of the target features significantly affect the results of the extraction.



(a) 'm27_aerial_image.tif'

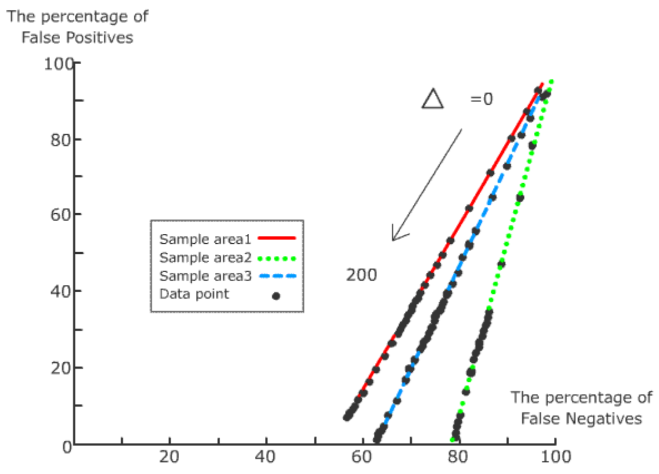


(b) 'M27_Junction_image.tif'

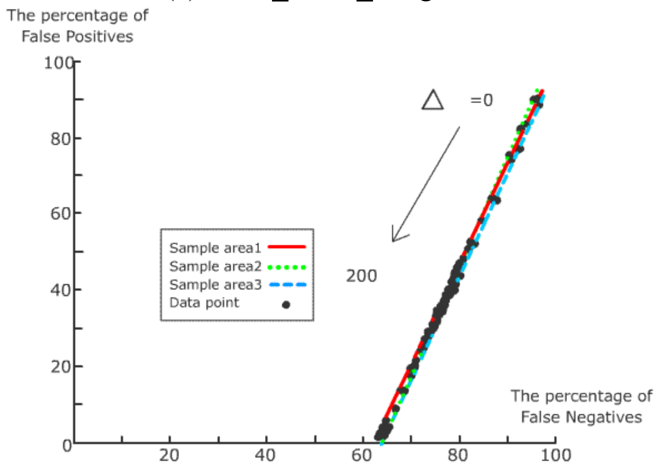


(c) 'Durham_Residential2_image.tif'

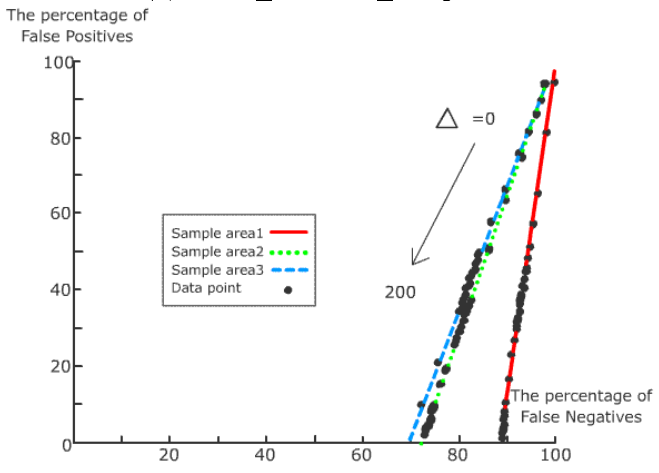
Figure 3.33: The relationships between the average of error distance and Δ



(a) 'm27_aerial_image.tif'



(b) 'M27_Junction_image.tif'



(c) 'Durham_Residential2_image.tif'

Figure 3.34: The relationships between the percentage of false positives and the percentage of false negatives

Figure 3.33 shows the average of error distance and Δ for each experiment in the order of image name. The error distance is the distance between the true lines to the extracted lines as shown in Figure 3.29. As mentioned in the experiment section, the delta varies from zero to twenty stepping by one. However, the average of error distance has not reached its steady state, the state where the average of error distance stays constant, at Δ = 20. To find the range of Δ that gives the steady state, additional experiments are needed. The Δ varying from 20 to 50 stepping by 10 and varying from 50 to 200 stepping by 50 are selected for the additional experiments. The results are shown in Figure 3.33. Figure 3.33 (b), for example, shows the results for 'M27_Junction_image.tif' with three different selections of the sample areas. The range of Δ giving the first steady state is approximately from 20 to 40 and the average of error distance at the steady state is different depending on the selected sample area. The minimum of the average of error distances at the steady state in Figure 3.33(a), 3.33 (b) and 3.33 (c) is about 5 to 6. This could lead to the suggestion for the accuracy of the program that, given a suitable sample area indicating the color of the target features, the program could accurately (the error distance 5 to 6) delineate the extracted lines close to the true lines.

Although the accuracy of the program is high when it correctly extracts the features, Figure 3.34 shows that the program produces a large percentage of false negatives. There are parts of the road that are missed. Figure 3.34 shows the relationships between the percentage of false positives and the percentage of false negatives for each experiment. The graphs provide the information about the percentage of the true pixels that cannot be found by the program and the percentage of the pixels that the true pixels and the extracted pixels are exactly matched (Δ = 0). For example, in Figure 3.34 (b), the former percentage for all sample areas is about 63 while the latter percentage is about 5 (where Δ = 0 and the percentage of false negatives ≈ 95). This means that, for 'M27_Junction_image.tif', 63 per cent of the true pixels will never be found and only 5 per cent of the extracted pixels are precisely located on the true pixels.

3.7 Conclusion

The program provides accurate results when the true pixels of the features are extracted although it misses a large number of the true pixels. This accuracy leads to the success of the Snakes optimization in delineating the hypothesis curves to the true curves; however, surrounding features such as shadows sometimes mislead the optimizer as shown in Figure 3.27.

The reason for a large number of the false negatives is the strong constraints. The aim of the strong constraints in the program is to limit the number of the false positives. As can be seen in Figures 3.34 (a) – (c), the percentage of the false positives reaches zero easily. Because of this, the program incorrectly filters out good candidates. As a result, a large number of the false negatives are produced. For example, if the constraint in the color filter is too strong, it will ignore many of good pixels and this is also the reason for the sensitivity of the sample areas.

If the constraints are weaker, the greater number of the candidate pixels will be provided to the extraction. This will help reduce the number of the false negatives. However, with the weak constraints, the number of the false positives will be increased. The goal to improve the program is to lower the number of the false negatives while the number of the false positives is not affected. The solution for this is discussed in chapter 4.

Chapter 4

Conclusion

The aim of this project is to develop a feature extraction system for two target features, constant rectangles and constant parallel lines. The extraction of these low-level shapes can be integrated into a geographic object recognition system as a primitive system. The feature extraction system deals with the details of its low-level imagery inputs to identify the target features. It then provides the low-level features to an object recognition system. With high-level information about the objects, the object recognition system classifies the features to the suitable object classes.

Because the problems of extracting the two target features are different, they are implemented in the different ways. They have led to different conclusions which are, therefore, discussed separately below.

4.1 Constant rectangle extraction

The goal of the constant rectangle extraction is to identify rectangles having constant or nearly constant interior from aerial imagery. Any objects in the imagery that have this characteristic will be extracted. For example, car parks, buildings, or even shadows can be extracted.

This approach integrates low-level techniques of computer vision to detect and verify the target rectangles. The edge detection is the most important process in this approach because the other processes rely on the detected edges. The edges are detected from an image input. They are transformed to regions. Morphological operators are applied to remove the noise regions. Each remaining region will be verified if it can be considered as a rectangle. This is done by fitting a rectangle onto each region. If the percentage of the overlapped area and the percentage of the error areas are acceptable, the region will be classified as a rectangle.

Although the experiments show that the approach produces a number of false negatives and false positives, accurate results are also provided when the extracted regions are matched to the expected rectangles. A reason for the false rectangles could be the sensitivity of the edge detection. Because it is the most important part in the system and it relies on nothing but the image intensity, edge detection tends to encounter the problem related to the ambiguity of the intensity.

Further work would be to improve edge detection. Edge detection should be locally adaptive to the different areas in the imagery. For example, a region that had a very large size and was rejected by a size constraint would be re-examined. With a new parameter set, the adaptive edge detection would then find edges in the area of the region again. Because the region was large, it needed the edge detector to find the edges more closely in details. The parameters for this could be adjusted by reducing the Gaussian blur value and increasing the range of the high and low threshold of Canny edge detector. This method would provide more potential candidates to the rectangle verification process.

In addition, evolutionary programming techniques such as genetic algorithms (GAs) could be integrated into the approach to adjust the parameters. Given the fact that sets of the valid parameters and a fitness function are provided, a GA is an effective tool for the optimization. GAs could find the optimum parameter set for each image input. The fitness function could be proportional to the number of region candidates. Moreover, if a general model of the features or more information about the features was available, the qualities of the candidates with respect to the information could also be embedded in the fitness function. For example, if the intensity of the target features were known, the fitness function would penalise the parameter set giving a large number of region candidates with the different intensities. Because edge detection was a pivotal factor, GAs could be first applied only for adjusting the parameters of the edge detection to test the feasibility of this idea.

4.2 Constant parallel line extraction

The goal of the constant parallel line extraction is to extract long parallel lines having constant or almost constant intensity between them. The inputs of the extraction are high-resolution aerial images. In contrast to object extraction systems, it is not necessary to include contextual information into the designed approach because the target features are the primitive shapes, which are parallel lines. On the other hand, due to the similarity of shape properties between roads and the parallel line features, the techniques designed for road extraction can be applied in the approach.

The approach starts with detecting centerlines between parallel lines. Most processes for this purpose are designed to reduce noise in a complicated image. The processes are discussed in chapter 3. The approach is designed to be parameter-adjusting-free based on general assumptions of features. The maximum and the minimum width of a parallel line, for instance, are fixed based on the information of the general width in the imagery. Different input images, however, need specific values for particular parameters. The strategy for this is to assign the parameters to be strong to limit the number of false positives although the number of false negatives will be increased. The detected centerlines are used as referent lines to initialize hypothesised parallel lines on both sides.

The image gradient is employed in the initializing process and the median of the width is used to control the width of the hypothesised lines to be constant. The hypothesised lines are initialized very close to the true lines when the lines have constant intensity between them and the intensity is distinct from their surroundings. The initializing process is sometimes fooled by high gradient values of the surroundings. The false hypotheses will be forced to the true lines by the Snakes optimization. The Snakes deforms the hypotheses based on the constraints of geometry and photometry. When the optimizing process reaches a compromise between the two constraints, the deformed curves are delineated close to or on the true lines.

The experiment shows that the approach gives accurate results when the true lines are correctly extracted. It, however, produces a significant number of false negatives (missed portions of the true roads). The false negatives occur due to the fact that the parameters such as thresholds and tolerant values are so strong to avoid false positives.

Further work is to increase the number of the extracted lines and the increased number must not produce more false positives. The technique to connect two extracted parallel curves is proposed. The two parallel curves could be connected if and only if their curvatures, their directions and their widths were close to each other and the gap between them was small. The two parallel curves would be now considered as one parallel curve. Because the connected part between them was initially located based on the image gradient and their widths, the new curve would be optimized again to fit the connected part to the true features. This further work, if successfully, would dramatically decrease the number of false negatives by filling the gaps between two parallel curves and it would not produce more false positives.

Bibliography

[1] R. Fisher, S. Perkins, A. Walker, E. Wolfart, “Feature Detectors – Canny Edge Detector”, The HIRP2 team, September 2000, Retrieved June 10, 2003 from the World Wide Web: http://www.dai.ed.ac.uk/HIPR2/canny.htm

[2] R. Fisher, “The RANSAC (Random Sample Consensus) Algorithm”, May 2002, Retrieved May 27, 2003 from the World Wide Web: http://www.dai.ed.ac.uk/CVonline/LOCAL_COPIES/FISHER/RANSAC/

[3] R. Fisher, S. Perkins, A. Walker, E. Wolfart, “Morphology - Closing”, The HIRP2 team, September 2000, Retrieved May 15, 2003 from the World Wide Web: http://www.dai.ed.ac.uk/HIPR2/close.htm

[4] The MathWorks, Inc., MATLAB Help, Keyword: “edge”, May 2001.

[5] R. Fisher, S. Perkins, A. Walker, E. Wolfart, “Spatial Filters – Gaussian Smoothing”, The HIRP2 team, September 2000, Retrieved June 10, 2003 from the World Wide Web: http://www.dai.ed.ac.uk/HIPR2/gsmooth.htm

[6] R. Fisher, S. Perkins, A. Walker, E. Wolfart, “Morphology - Erosion”, The HIRP2 team, September 2000, Retrieved May 15, 2003 from the World Wide Web: http://www.dai.ed.ac.uk/HIPR2/erode.htm

[7] R. Fisher, S. Perkins, A. Walker, E. Wolfart, “Morphology - Opening”, The HIRP2 team, September 2000, Retrieved May 15, 2003 from the World Wide Web: http://www.dai.ed.ac.uk/HIPR2/open.htm

[8] R. Fisher, S. Perkins, A. Walker, E. Wolfart, “Morphology - Dilation”, The HIRP2 team, September 2000, Retrieved May 15, 2003 from the World Wide Web: http://www.dai.ed.ac.uk/HIPR2/dilate.htm

[9] R. Fisher, “Lecture 7: Boundary Extraction and Segmentation”, slide 5-10, 12, Advanced Vision, January 2003, Retrieved March 14, 2003 from the World Wide Web: http://www.dai.ed.ac.uk/homes/rbf/AVAUDIO/lec7.pdf

[10] R. Fisher, S. Perkins, A. Walker, E. Wolfart, “Morphology – Distance Transform”, The HIRP2 team, September 2000, Retrieved July 2, 2003 from the World Wide Web: http://www.dai.ed.ac.uk/HIPR2/distance.htm

[11] David Young, “Active Contour Models (SNAKES)”, Sussex Computer Vision: TEACH VISION7, March 1995, Retrieved July 15, 2003 from the World Wide Web: http://www.dai.ed.ac.uk/CVonline/LOCAL_COPIES/YOUNG/vision7.html

[12] Foley, van Dam, Feiner, Hughes, “Computer Graphics: Principles and Practice Second Edition in C”, pp. 72 – 81, July 1997.

[13] The MathWorks, Inc., MATLAB Help, Keyword: “gradient”, May 2001.

[14] The MathWorks, Inc., MATLAB Help, Keyword: “bwdist”, May 2001.

[15] R. Fisher, S. Perkins, A. Walker, E. Wolfart, “Morphology – Thinning”, The HIRP2 team, September 2000, Retrieved July 20, 2003 from the World Wide Web: http://www.dai.ed.ac.uk/HIPR2/thin.htm

[16] R. Fisher, S. Perkins, A. Walker, E. Wolfart, “Morphology – Skeletonization/Medial Axis Transform”, The HIRP2 team, September 2000, Retrieved July 20, 2003 from the World Wide Web: http://www.dai.ed.ac.uk/HIPR2/skeleton.htm

[17] Department of Electronic Engineering, The Chinese University of Hong Kong, “Tutorial Notes 5: Hints on MATLAB Assignment – Snake”, February 2003, Retrieved July 21, 2003 from the World Wide Web: http://www.ee.cuhk.edu.hk/~yywai/ele4430/tutorial/tutorial05.pdf

[18] H. Mayer, I. Laptev, A. Baumgartner, “Multi-Scale and Snakes for Automatic Road Extraction”, 1998, Retrieved July 21, 2003 from the World Wide Web: http://www.photo.verm.tu-muenchen.de/dfg_roads/references/eccv98-paper-43.ps.gz

[19] Mayer, I. Laptev, A. Baumgartner, C. Steger, “Automatic Road Extraction Based On Multi-Scale Modeling, Context, and Snakes”, 1997, Retrieved July 21, 2003 from the World Wide Web: http://www.photo.verm.tu-muenchen.de/dfg_roads/references/haifa97.ps.gz

[20] Meir Barzohar and David B. Cooper, “Automatic Finding of Main Roads in Aerial Images by Using Geometric – Stochastic Models and Estimation”, IEEE Transactions on Pattern Analysis and Machine Intelligence, 18(7), pp. 707 – 721, July 1996.

[21] P. Fua and Y. G. Leclerc, “Model Driven Edge Detection”, SRI International, Machine Vision and Application, 3, 1990, Retrieved July 21, 2003 from the World Wide Web: http://www.rfai.li.univ-tours.fr/rousselle/docum/CA102.htm

[22] Stefan Hinz, Albert Baumgartner, “Automatic Extraction of Urban Road Network from Multi-View Aerial Imagery”, Preprint submitted to Elsevier Science, February 2003, Retrieved July 22, 2003 from the World Wide Web: http://www.photo.verm.tu-muenchen.de/site_lpf/site_new/download/pub/2003/hinz_isprs03.pdf

[23] Ivan Laptev, “Road Extraction Based on Line Extraction and Snakes”, Master thesis, Computational Vision and Active Perception Lab (CVAP), Royal Institute of Technology, Stockholm, Sweden, May 1997, Retrieved July 21, 2003 from the World Wide Web: http://www.nada.kth.se/~laptev/abstracts/mscthesis.html

[24] Amnon Meisels and Samuel Bergman, “Finding Objects on Aerial Photographs: A Rule-Based Low Level System”, Dept. of Math. & Comp. Sci., University of the Negev, Isarael, IEEE, 1988, pp. 118 – 122.

[25] Yuan Hsieh, “SiteCity: A Semi-Automated Site Modelling System”, Digital Mapping Laboratory, School of Computer Science, Carnegie-Mellon University, IEEE 1063-6919/96, 1996, Retrieved June 5, 2003 from the World Wide Web: http://www.maps.cs.cmu.edu/RCVW/papers/96Hsieh.htm

[26] Fischler, M., Tenenbaum, J. and Wolf, H. (1981), “Detection of Roads and Linear Structures in Low-Resolution Aerial Imagery Using a Multisource Knowledge Integration Technique”, Computer Graphics and Image Processing **15:** 201 – 203.

[27] McKeown, D. and Denlinger, J. (1988), “Cooperative Methods For Road Tracking In Aerial Imagery”, Computer Vision and Pattern Recognition, pp. 662 – 672.

[28] Quam, Lynn H. (1978), “Road Tracking and Anomaly Detection in Aerial Imagery”, Image Understanding Workshop, pp. 51-55.

[29] Kass, M., Witkin, A. and Terzopoulos, D. (1987), “Snakes: Active Contour Models”, International Journal of Computer Vision **1**(4): 321 -331.

[30] Bajcsy, R. and Tavakoli, M. (1976), “Computer Recognition of Roads from Satellite Pictures”, IEEE Transactions on Systems, Man, and Cybernetics **6**(9): 623 – 637.

[31] McKeown, D. and Denlinger, J., (1988), “Cooperative Methods For Road Tracking In Aerial Imagery”, Computer Vision and Pattern Recognition, pp. 662 – 672.

[32] Vosselman, G. and de Knecht, J., (1995), “Road Tracking by Profile Matching and Kalman Filtering”, Automatic Extraction of Man-Made Objects from Aerial and Space Images, Birkhäuser Verlag, Basel, Switzerland, pp. 256 – 274.

[33] Zlotnick, A. and Carnine, P., (1993), “Finding Road Seeds in Aerial Images”, Computer Vision, Graphics, and Image Processing: Image Understanding 57, pp. 243 – 260.

[34] Ruskoné, R., (1996), “Road Network Automatic Extraction by Local Context Interpretation”, Application to the Production of Cartographic Data. Thèse de doctoral, Université de Marne-La-Vallée, Noisy-Ie-Grand, France.

[35] R. Nevaia and K.R. Babu, “Linear Feature Extraction and Description”, IEEE CGIP, 1980, pp. 257 – 269.

[36] W. Kestner and H. Dazmierzak, “Semiautomatic Extraction of Roads from Aerial Photographs”, Research Inst. For Information Processing and Pattern Recognition, 1978.

[37] R.O. Duda and P.E. Hart, “Pattern Classification and Scene Analysis”, New York: John Wiley, 1973.

[38] D. Cooper, “Maximum Likelihood Estimation of Markov Process Blob Boundaries in Noisy Images”, IEEE Transaction, Pattern Analysis and Machine Intelligence, October 1979.

[39] D.B. Cooper and F.P. Sung, “Multiple-Window Parallel Adaptive Boundary Finding in Computer Vision”, IEEE Transaction, Pattern Analysis and Machine Intelligence, 1983.

[40] S.S. Vincent Hwang, “Evidence accumulation for spatial reasoning in aerial image understanding”, Ph.D. thesis, University of Maryland, College Park, 1984.

[41] P.G. Selfridge, “Reasoning about Success and Failure in Aerial Image Understanding”, Ph. D. thesis, University of Rochester, NY, 1982.

[42] A. M. Nazif and M. D. Levine, “Low level image segmentation: An expert system”, IEEE Transaction, Pattern Analysis and Machine Intelligence, 1985.

[43] W. J. Mueller and J. A. Olson, “Model-based feature extraction”, Integrating Photogrammetric Techniques with Scene Analysis and Machine Vision, vol. 1944, pp. 263 – 272.

[44] Canny J., “A computational approach to edge detection”, IEEE Transaction on Pattern Analysis and Machine Vision, 8(6), pp. 679 – 698.