Gottfried Wilhelm Leibniz Universität Hannover
Institut für Informationsverarbeitung

Diplomarbeit

Tracing Parallel Line Structures For Land Cover Analysis

Marc Abuin Fernandez

Betreuer:	Torsten Bschenfeld
Erstprüfer:	Prof. Dr.-Ing. Jrn Ostermann
Zweitprüfer:	Prof. Dr.-Ing. Bodo Rosenhahn

Hannover, den 09.11.2010

Ich erkläre hiermit, dass ich die vorliegende Arbeit selbständig angefertigt und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Contents

1 Prólogo

1.1 Introducción

El creciente uso de las imágenes satélite y la mejor calidad de estas permiten a los gobiernos tener un mejor conocimiento de su región. Concretamente los mapas que clasifican las regiones según si son urbanizaciones, industria o zona verde. De este modo se tiene un mayor control de las posibilidades de cambio y desarrollo de cada región. En este contexto, y enmarcado en el proyecto Geoaida, sistema de interpretación automática de datos, se ha desarrollado este proyecto. Su finalidad es, a partir de imágenes de zonas verdes, identificar campos de cultivo, diferenciándolos así de los bosques u otras zonas verdes con estructuras diferentes. Estos cultivos se identifican a partir de las estructuras de líneas paralelas que forman el arado del campo. Más concretamente, se pretende identificar cultivos en África, donde normalmente siguen las irregularidades del terreno. De esta forma los cultivos dejan de ser parcelas con un arado de líineas rectas y pasan a ser un conjunto de líneas curvas paralelas. El objetivo técnico del proyecto es la identificación de estructuras de líneas curvas paralelas. Y para ese fin también se incluye la reconstrucción de líneas que, por motivos de ruido en la imagen o deficiencias en el filtrado, se hallan segmentadas. De esta forma se obtienen líneas más largas que, a la hora de estudiar el paralelismo, ofrecen datos más fiables.

1.2 Método

La imagen se pre-procesa con el filtro detector de bordes de Canny. El contraste del arado da lugar a líneas de borde que son detectadas por este filtro. El resultado es una imagen compuesta por líneas blancas sobre fondo negro. Esta imagen es la que será tratada por el algoritmo desarrollado en este proyecto. Dicho algoritmo se compone de tres fases principales:

- Recolección de líneas
- Reconstrucción de líneas
- Estudio del paralelismo

La idea del algoritmo es recopilar datos sobre las líneas que componen la imagen extraída del filtro de Canny, reconstruirlas con el fin de tener líneas más largas, y, finalmente, buscar estructuras de líneas paralelas para poder decidir si la imagen contiene un campo de cultivo.

Recolección de líneas

El algoritmo explora toda la imagen píxel por píxel buscando los más brillantes (blancos) que son los que pertenecen a las líneas que forman bordes en la imagen original. Una vez hallado el primer píxel blanco se busca en la vecindad otros píxeles blancos conectados. De esta forma, y teniendo en cuenta que el filtro de Canny obtiene líneas de un píxel de ancho, si se encuentran varios pixeles blancos conectados significa que se ha encontrado una línea. Las coordenadas de estos píxeles se almacenan en orden (de extremo a extremo de la línea) y se sigue con la exploración de la imagen en busca de ms líneas.

La tabla 5.3 muestra la importancia del almacenamiento de las coordenadas en orden. Por esta razón el algoritmo utiliza dos funciones diferentes para el seguimiento de la línea. El primero se utiliza para encontrar el inicio de ésta y el segundo para seguir des del primer punto encontrado hasta el final. El orden determina el cálculo del valor de la pendiente de la línea (o de los vectores directores). En la tabla se puede ver como un orden incorrecto del almacenamiento de las coordenadas puede llevar a valores erróneos de dichos vectores directores. Este error puede suponer más errores en las siguientes etapas, que basan su funcionamiento en los datos obtenidos por este primer paso. La reconstrucción de líneas se vería comprometida así como la comparación de pendientes para el estudio del paralelismo. Una vez que se ha explorado toda la imagen, esta primera etapa también hace un filtrado de los segmentos almacenados y se queda con los más largos. Como no existe una longitud estándar de los segmentos recogidos en las diferentes imágenes, se utiliza un umbral variable dependiendo de la longitud media de los segmentos recogidos en cada imagen. Este filtrado es necesario para evitar que segmentos muy cortos puedan actuar como ruido, dificultando el funcionamiento del algoritmo. Un claro ejemplo de ello es la figura 6.20 , donde se observa que la zona verde de la zona original se traduce en una multitud de segmentos cortos sin una orientación clara.

Reconstrucción de líneas

En esta etapa el algoritmo trabaja ya con los datos recopilados en la primera etapa. Trabaja, sobretodo, con las coordenadas de las líneas. Estas coordenadas son importantes para extraer el conjunto de direcciones que toman las líneas en toda su longitud y que se almacenan en vectores. En esta segunda fase se pretende reconstruir las líneas que, por deficiencias en la imagen o en el filtrado, se han convertido en segmentos. Para unir estos

segmentos el algoritmo, que trabaja con cada segmento, busca si podría formar una línea más larga con otros segmentos de alrededor. La búsqueda es en una región triangular orientada hacia la última dirección del segmento con el que trabaja el algoritmo. De esta forma el algoritmo puede encontrar varios segmentos candidatos a ser la continuación. Tras encontrar los candidatos se debe escoger uno entre todos ellos para ser la continuación al segmento procesado en ese momento y pertenecer así a una línea más larga. Este estudio se hace por comparación de direcciones. Se calcula el ángulo que forman la línea que une ambos segmentos con la última dirección del segmento candidato. El ángulo que se busca es el más cercano a cero, como se observa en la figura 5.13. Esta etapa va uniendo los segmentos separados por pequeos espacios vacíos, formando así líneas más largas y reduciendo el número total de líneas en la imagen. Ésta pasa de estar formada por segmentos cortos a contener líneas largas (en el caso que aparezcan en la imagen original).

Este proceso se ejecuta con cada línea hasta que el método no encuentra candidatos buenos para unirlos a la línea. Es entonces cuando el algoritmo pasa a su tercera y última fase: el estudio del paralelismo.

Estudio del paralelismo

El algoritmo puede trabajar en esta fase final con las mejores líneas, aquellas que aportan datos más fiables ya que se ve mejor la evolución de una línea curva. Para estudiar el paralelismo de líneas curvas es importante conocer la mayor parte de las direcciones que toma ésta. El algoritmo, en este tercer paso, también trabaja línea por línea. Esta fase se puede se diferencian dos partes: encontrar las líneas candidatas a ser paralelas y comprobar si las candidatas son realmente paralelas a la línea procesada. El método utilizado para encontrar las líneas candidatas es una línea ortogonal a la línea a la que se le buscan paralelas. Como se ve en los ejemplos 4.13 y 4.12, dos líneas pueden tener la misma curvatura pero según su posición relativa pueden no ser paralelas. En este aspecto, la línea ortogonal descarta como candidatas aquellas líneas que no crucen la línea ortogonal o que, en el punto de cruce, no sean perpendiculares a la línea ortogonal.

Con los candidatos que quedan se realiza una comparación de sus pendientes. Para ello se utiliza la distancia Euclidiana (expresión 5.3) entre los vectores directores de las líneas. Con el fin de salvar las diferencias de longitud entre líneas, la línea más corta se compara con diferentes tramos de la línea más larga tal y como se detalla en la sección 5.3.2. La distancia ideal entre vectores directores para el paralelismo sería cero. Pero, dado que el arado de un terreno no responde a líneas trazadas de forma exactamente paralelas, se utiliza un valor umbral como relajación de los requisitos del paralelismo. La distancia, pues, debe estar por debajo de dicho umbral para poder considerar dos líneas paralelas.

El programa, finalmente, cuenta el número de líneas que pertenecen a grupos de líneas paralelas. Si éstas representan un porcentaje significativo del total de líneas con las que

ha trabajado el programa, el algoritmo acaba concluyendo que la imagen contiene un campo de cultivo.

1.3 Resultados

Cada fase trabaja a partir de los resultados de la fase anterior, por lo que se observa una propagación de errores. Si el algoritmo no logra una buena recolección de líneas con una determinada imagen (obteniendo menos líneas o segmentos más cortos), esto repercutirá directamente en los resultados de las siguientes fases. Es por ello que en este estudio también se muestran resultados intermedios. De esta forma se puede valorar cada parte.

Los resultados que se pueden extraer de la primera fase muestran en la tabla 6.1. Los segmentos recogidos por esta parte del algoritmo pertenecen a las principales líneas de la imagen. Aquellas líneas que luego servirán para determinar si se trata de un campo de cultivo. Para ello se requiere una imagen nítida que se vea claramente el arado. Las imágenes 6.1 y 6.12 son el ejemplo de una imagen nítida con un arado claro y una imagen con un peor contraste y con un arado más borroso. Se observa que en la imagen más borrosa no se obtienen segmentos suficientemente largos como para formar líneas para el posterior estudio del paralelismo. Se observa también que el algoritmo suele mantener alrededor del 10% de los segmentos recolectados con mayor longitud y estos segmentos son los que se deben reconstruir en la siguiente fase del proyecto.

Los resultados de la segunda fase son difíciles de cuantificar, puesto que la relación entre segmentos y líneas reales no siempre es la misma. Por lo tanto se ha contado la reducción del número de líneas para contar el número de líneas que se han conectado (y reconstruido) durante esta segunda fase. Estos datos se recogen en la tabla 6.2 y se observa que en el mejor de los casos se reduce un 14% el número de líneas por ir conectando segmentos entre sí para formar líneas más largas.

En la última fase se observan resultados numéricos y gráficos. Éstos son los resultados finales del algoritmo presentado en este estudio. Aunque la decisión se basa en los porcentajes de la tabla 6.3. El umbral que se propone es el 50%. Es decir, si el porcentaje de líneas paralelas respecto al total de líneas en la imagen (en la tercera fase) supera el mencionado 50% el algoritmo concluye que la imagen contiene un campo de cultivo. Las dos imágenes que pertenecen a campos de cultivo tienen un porcentaje muy superior al de las imágenes que no contienen ningún campo de cultivo.

1.4 Conclusiones

Así pues, con el fin de identificar campos de cultivo en fotos satélite, se ha desarrollado un detector de estructuras de líneas paralelas. El objetivo real es la reconstrucción de líneas curvas segmentadas y el estudio de estructuras de líneas paralelas curvas. Para un

trabajo futuro, aumentar el porcentaje de líneas conectadas ayudaría al estudio de líneas paralelas por varios motivos: el programa trabajaría con datos más fiables y habrías más posibilidades de identificar líneas paralelas. Otra posible mejora sería realizar una conexión diferente a una línea recta. Una conexión por interpolación o extrapolación de los segmentos a unir ayudaría a un mejor estudio del paralelismo.

2 Introduction

Earth images from satellites are an ever-growing asset with applications in several fields. The geographic knowledge is important for enterprises or the civil service. It allows a better management of woods and crop lands towards the optimization of natural resources.

IKONOS is one of the satellites that take high-resolution and wide spectrum pictures. From satellite images and Geographic Information System (GIS) data, GeoAIDA manages with its semantic net and control the analysis process. GeaoAIDA is a system for controlling the interpretation of remote data for the purpose of land use analysis [1], [2] and [3]. It uses a semantic net for classifying regions as settlement, industrial, wood or land regions. Agricultural lands show specific structure of parallel lines. In Europe, these lands have structures based on straight lines. But for instance, in Africa, they adapt to the irregular shape of the terrain. Many times they have curved shape, and some methods applied to straight shaped fields do not work. This report is focused on these agricultural land pictures.

In image processing, edges are fundamental to the analysis of shape. This shape is what the program uses in order to analyze the content of the image. An edge detector is an essential tool for pre-processing images based on lines: it extracts main lines and allows the program to work with images based on lines. After edge detection process, the resulting image consists of some line, most of them presenting gaps. That is, the image is composed of segments. When these segments are curved there are three challenges to manage with: detect the lines, close gaps and study their parallelism.

Focused on them, the study is divided in three parts. After a previous processing of the image, it is necessary to search and collect segments. Their location is stored as well as their geometric data such as slope and direction.

The following step is the reconstruction of lines, which is necessary in order to obtain longer lines. That is because the study of parallelism of longer lines is more reliable than a study of short segments. A good collection of segments causes a good reconstruction. So, the order of the data are collected influences the outcome, so the storage step is as important as the analysis. The last part is the study of parallelism. Here, the results follow straight from the outcome of the previous processes. Location and slope of each line allow detecting parallel lines, while the curvature is now not important.

So there are three processes whose efficiency is directly related to each others. That is, error propagation is in all steps. It means that the outcome of the first step becomes

essential for the following processes to work properly.

3 Previous knowledge

3.1 Filters

An Important step in image processing is choosing the best filter. Filters can be classified in two types: point-oriented and region-oriented filters.

- **Point-oriented filter:** every action of this kind of filter changes a pixel value and it is independent of values of pixels around. Neighborhood does not change how this kind of filter transforms the pixel value.

- **Region-oriented filter:** also called spatial domain filter, it works with pixel's neighborhood. This linear filter works with a kernel matrix and does a convolution between kernel matrix and the pixel with others pixels around it.

$$Image = \begin{pmatrix} I_{00} & \cdots & I_{i0} & \cdots & I_{M-10} \\ \vdots & \ddots & \vdots & & \vdots \\ I_{0j} & & I_{ij} & & I_{M-1j} \\ \vdots & & \vdots & \ddots & \vdots \\ I_{0N-1} & \cdots & I_{iN-1} & \cdots & I_{M-1N-1} \end{pmatrix}$$

For a 3x3 neighbourhood:

$$\begin{aligned} I'_{ij} = I_{i-1j-1} \cdot w_1 + I_{ij-1} \cdot w_2 + I_{i+1j-1} \cdot w_3 + I_{i-1j} \cdot w_4 + \\ I_{ij} \cdot w_5 + I_{i+1j} \cdot w_6 + I_{i-1j+1} \cdot w_7 + I_{ij+1} \cdot w_8 + I_{i+1j+1} \cdot w_9 \end{aligned} \tag{3.1}$$

Where I=intensities and w=weights.

3.2 Spatial domain filters

The value of every pixel is multiplied by its corresponding filter coefficient, as shows the expression 3.1. All nine obtained values are added up and the result is divided by the

sum of all filter's coefficients. The result becomes the new pixel value in the image which is in the middle of the filter. When the kernel matrix covers the whole matrix, which represents image grey pixel values in the figure 3.1, the convolution is done. Setting up the parameters (figure 3.2) with different values makes different filters. The figure 3.2 shows an example of high-pass filter. The sum of all its elements is zero. So when the nine pixels have the same value, the result is zero.

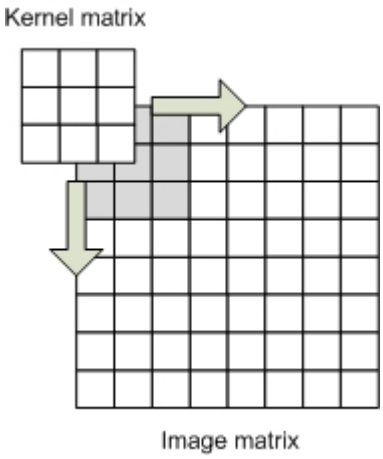


Figure 3.1: Convolution.

w_1	w_2	w_3
w_4	w_5	w_6
w_7	w_8	w_9

Figure 3.2: Parameters kernel matrix.

Smoothing filter, average filter and median filter are low-pass filters. The goal of these filters is blurring and minimizing noise; it is important in an image pre-processing. Noise can lead detection algorithms to an error or false detections.

In a previous image processing is also necessary a high-pass filters. Image details are in high frequencies being possible the detection of jumps in the image intensity. These jumps can represent boundaries of objects or certain delimited regions. Different altitudes or colors in a land region mean noticeable changes on intensity in grey-scaled images.

According to the expression 3.1 and the kernel matrix 3.2, a region uniformly colored becomes a dark region with values approximately zero. On the contrary, when the filter pass by a region where the intensity changes abruptly, the result is different from zero. It is a good segmentation method because in a dark background are white pixels that most of them represent boundaries.

3.2.1 Edge detector

As is remarked before, edge detector filter is an essential segmentation tool. The good behavior of this method will affect efficiency of all algorithms in this study. If there were

-1	-1	-1
-1	8	-1
-1	-1	-1

Figure 3.3: Example of high-pass filter kernel matrix.

too much false detection the interpretation after parallelism study could be completely wrong, even though following algorithms work. The choice of an edge detector (ED) is not trivial. A good result after edge detector depends on images and setting of parameters. There is no one works best with all images and fixed parameters. Because an automatic parameter setting for ED is not possible yet, results through different images will be with variable in their quality. A parameter setting that works well for an image, may not work for another.

This study is about green land images and all of them have some similarities. Working with a certain type of image can help in order to set minimally ED parameters. The previous setting has to be manual, by comparison watching several results with different values for each parameter. Thus, this variability can be minimized obtaining good results with fixed parameters, although the quality of results was lower.

According to the comparative study of several well-known edge detectors [4], Canny edge detector [6] is the best option for this study. Its goodness with fixed parameters is compared with Nalwa-Binford, Sarkar-Boyer and Sobel edge detector. Statistically, Canny and Nalwa-Binford obtain better results. Results of Nalwa-Binford edge detector change less over all images with fixed parameter, but its output is not one pixel wide. For this study is necessary an output image composed of one pixel wide lines. This is because the collection process works with this kind of lines. Wider lines would be harder to collect, put in order and study their properties. So Canny is the edge detector best meets the requirements.

3.2.2 Canny edge detector

Canny edge detector methodology (1986) has three basic ideas.

- **Detection:** in output image has important edges. It is highly unlikely that the detector checks a false edge.
- **Location:** the distance between real location and detected location should be minimal.
- **One result:** The detection of an edge only gets a single result.

The Canny method follows three main steps: gradient calculation, non-maximum suppression and hysteresis loop.

Gradient calculation

Prior to the gradient calculation it is necessary to minimize the noise on the image with a Gaussian filter. This low-pass filter smoothes the image depending on a σ parameter. The bigger σ is, the greater the smoothing is. It is important to strike a balance between the noise reduction and details to preserve because edges extraction is difficult from a too blurred image.

$$f(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} \tag{3.2}$$

Gradient magnitude and orientation of every pixel is solved after Gaussian filter has minimized the noise. It supposes to have two images: The value of each pixel is equal to the gradient magnitude (3.4) on first one and it is equal to the gradient orientation (3.5) on the other one.

$$G[f(x,y)] = \begin{bmatrix} G_x \\ G_y \end{bmatrix} = \begin{bmatrix} \frac{\partial}{\partial x} f(x,y) \\ \frac{\partial}{\partial y} f(x,y) \end{bmatrix} \tag{3.3}$$

$$|G| = |G_x| + |G_y| \tag{3.4}$$

$$\phi(x, y) = tan^{-1}\frac{G_y}{G_x} \tag{3.5}$$

Non-maximum suppression

Every pixel of magnitude image is compared with two neighbors. These neighbors are located following the angle value of the same pixel on gradient orientation image. If the magnitude of the pixel is lower than some of these two neighbors the final value of the pixel on output image is equal to zero. Otherwise, the value remains as the value of the gradient magnitude.

Hysteresis loop

Hysteresis loop has two thresholds, one greater than another. After Non-maximum suppression this method looks for pixel values greater than the greatest threshold. Then it follows the direction perpendicular to the orientation of the gradient and checks if these pixels have greater values than the lower threshold. Only these connected peaks are listed in the outline they belong to.

The figure 3.4 shows the process of Canny edge detector. There are the original image and the output images of the different steps mentioned above.
After Canny edge detector the image is composed by with lines on a black background as seen in the figure 3.5. So the image is ready to be processed by the algorithm of the first part of this study.

3.2.3 Curve tracing and curve extraction.

The human eye is able to identify lines in an image. Lines which allow to detect contours and shapes that help the person to identify objects. These lines and these objects are critical for the image interpretation. But there are contour lines and lines that are part of the texture of the object observed. The field of image processing researches how to interpret complex images. Among other things, it allows to know how to differentiate lines of contour in a texture, since some textures may act like noise to detect the edges (in most cases, the most important thing).

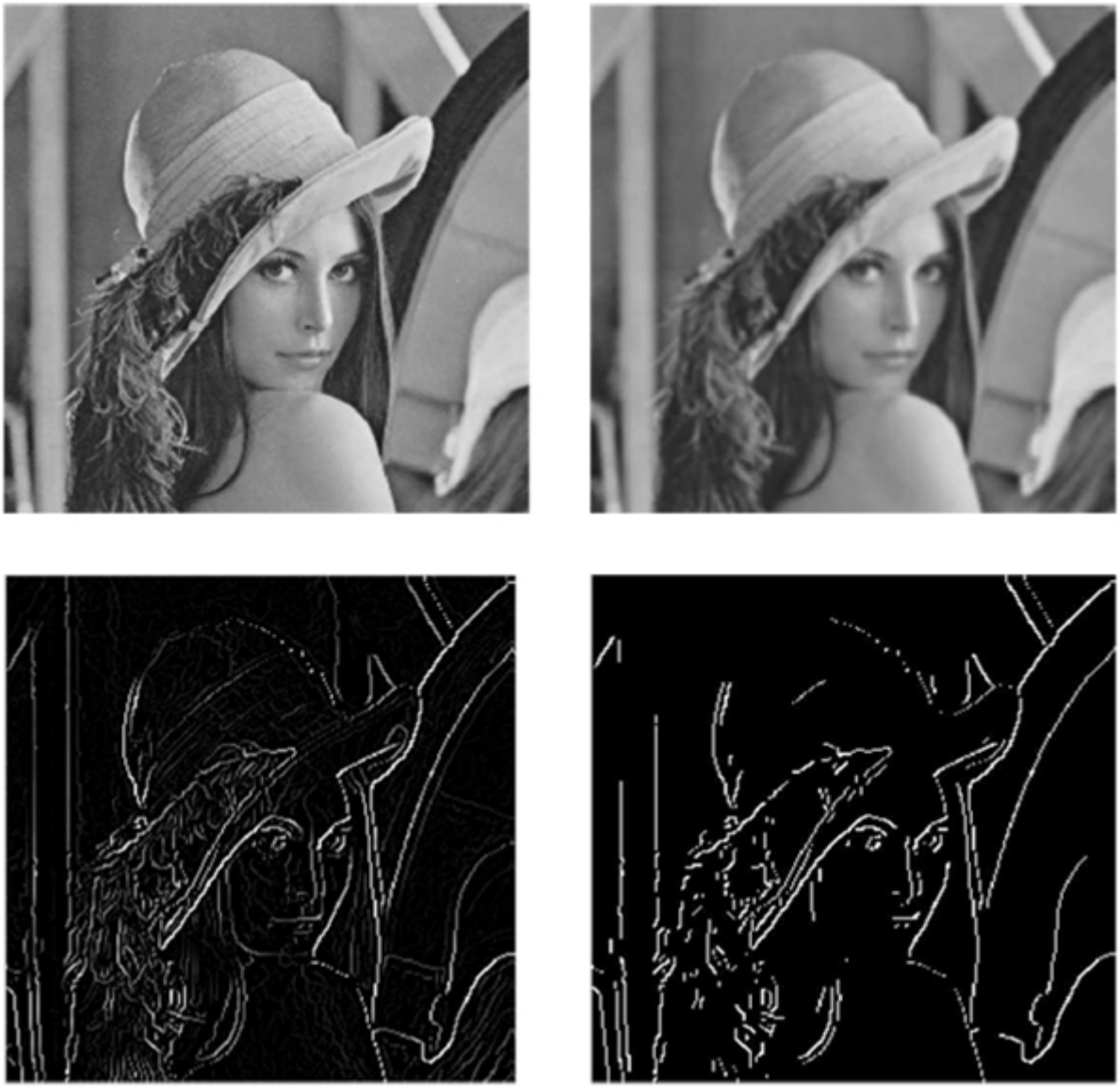


Figure 3.4: Different steps in Canny edge detector. Top left original image. Top right image after the Gaussian filter. Bottom left image after Non-maximum suppression. Bottom right output image after hysteresis loop.

To study the lines, the first step is to extract the lines and then detect them as lines. The first operation is called line extraction and the edge extraction filters performs this function. Canny edge detector filter is chosen in this study. Its main function is to pick pixels that belong to lines and contours up, and isolate them in another image. This new image, in this study, is a binary image.
There are several filters, as mentioned above and different methods to extract curves such as curvelets. This method extracts curves and segments, locations, orientations

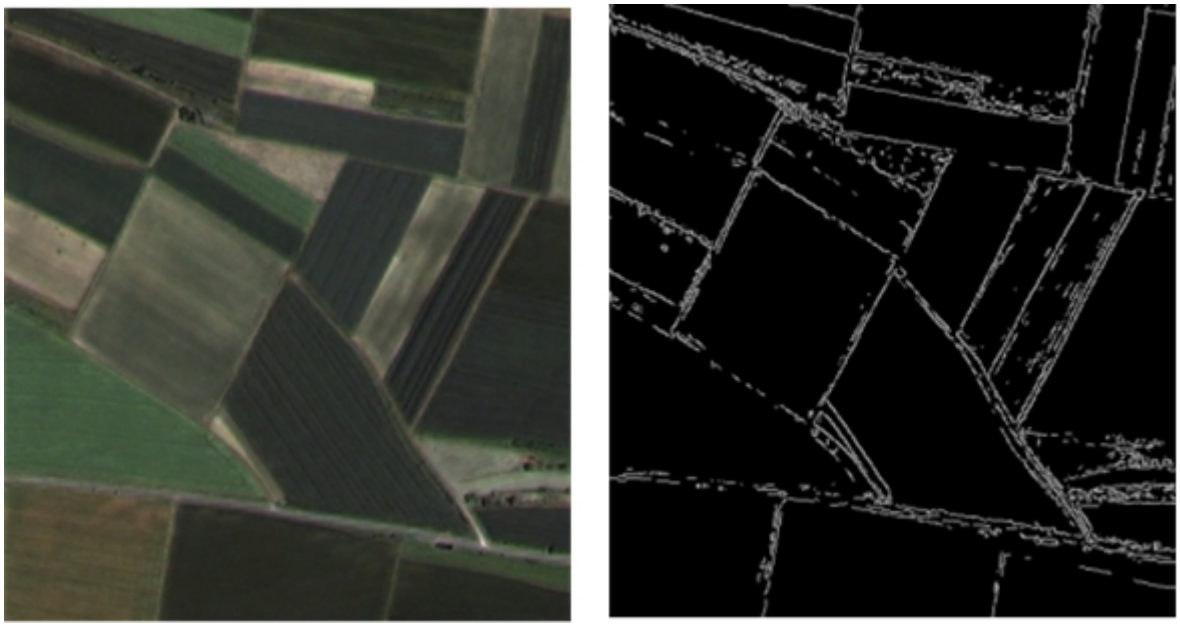


Figure 3.5: Original image of agricultural land on the left. Output image after Canny edge detector on the right.

and scales, which are useful for compressing images or representing objects with edges. These methods are very appropriate to draw the lines, but not labeled. The line does not appear in their data as a unit itself. This is part of the curve tracing, which makes a local study of isolated pixels by extracting lines. Interconnected pixels form a line with length and a certain form.

4 Tracing parallel line structures

4.1 Introduction

This study is divided in three steps. First one is the collection of segments. It works with Canny edge detector output image. It collects main lines on the image and it stores them with line class variable. The main goals of this step are the collection of larger lines and their duly ordered and structured storage for further study.

Second step is the reconstruction of lines. Despite de good performance of Canny edge detector, even there are some broken lines. For the parallelism comparison, the larger the lines are, the better results are because a comparison is more reliable when there are more samples. These lines are studied and its geometric properties are stored in line class variable. Their ends location, their coordinates in the picture among other properties are stored and the slope values along the lines are solved. In this second step the algorithm estimates a line that follows the last slope of the curve. The algorithm uses this line to compare with other estimated lines of other curves around so the algorithm can decide which lines should be connected. Also it uses this estimated line to find possible segments that might belong to a line originally larger. Thus, this step is based on the calculation and study of the properties of the lines so it can reconstruct the basic lines of the image (the most important for the image interpretation) from the segments detected by the previous step.

The third and last step is the parallelism study. It looks for the lines around a line and examines whether these are parallel to the line. This algorithm measures the Euclidean distance to compare the curvature of both lines. In this step there are two important goals: how to find the lines that can be parallel and how to compare them with the line treated by that part of the algorithm. In the first case is significant to take with large or little lines that can be located along the line. Two parallel lines may be that do not start at the same point. Especially because in first step are only segments of lines. But looking for lines in an extended region is inefficient. The second case is related with the first one. If lines do not start at the same point the parallelism study have to be very careful. Two lines can have the same curvature but they can be located so that they are not parallel.

4.2 Step 1: collection of segments

In this step the image has been pre-processed by Canny edge detector. So it is a grey level image where the needed segments are white. The goal of image scanning is to check all the pixels in the image and to detect white connected pixels as a possible segment. The coordinates of these pixels need to be stored in an orderly way. The idea is to work with segments instead of working with single pixels. But this first step has to work to pixel level in order to build segments.

4.2.1 Image scanning and tracking segments

In this step, two operations are performed simultaneously. First is looking for a white pixel. In the image obtained after applying the Canny edge detector, pixels have different intensities of gray. A high intensity above a threshold is considered a white color. When a white pixel is found, the first search stops and starts the formation of a segment.

The second search is performed in the neighborhood of the white pixel in order to find more white pixels connected to the first one. And then it keeps searching in the neighborhood of the new pixels found. The formation of a segment finishes when no more white pixels connected are found. Then the first search continues in order to find another white pixel. So, the first operation looks for a white pixel and the second operation looks for the segment where this pixel may belong.

Thus, it collects all the pixels with a required minimum intensity. Then it does a comparison of the segments lengths. The segments too short are rejected. This rejection has two purposes:

The first is to study the edges which are best detected. If the program has longer segments, it is easier to form the original lines. The second purpose is to remove noise. In the reconstruction lines step, the segments too short do not help at all in the process. The attributes of these segments are not reliable because they are based on very little data. These segments hinder the reconstruction step, and it is why they are considered as noise and they are rejected.

The figure 4.1 shows a picture after Canny edge detector. There are pixels with different intensity values. As shown in the image 4.2, the pixels collected by the program are the brightest. It can be seen that this outcome is free of shorter segments of the image. It shows the selection of the longer segments above mentioned.

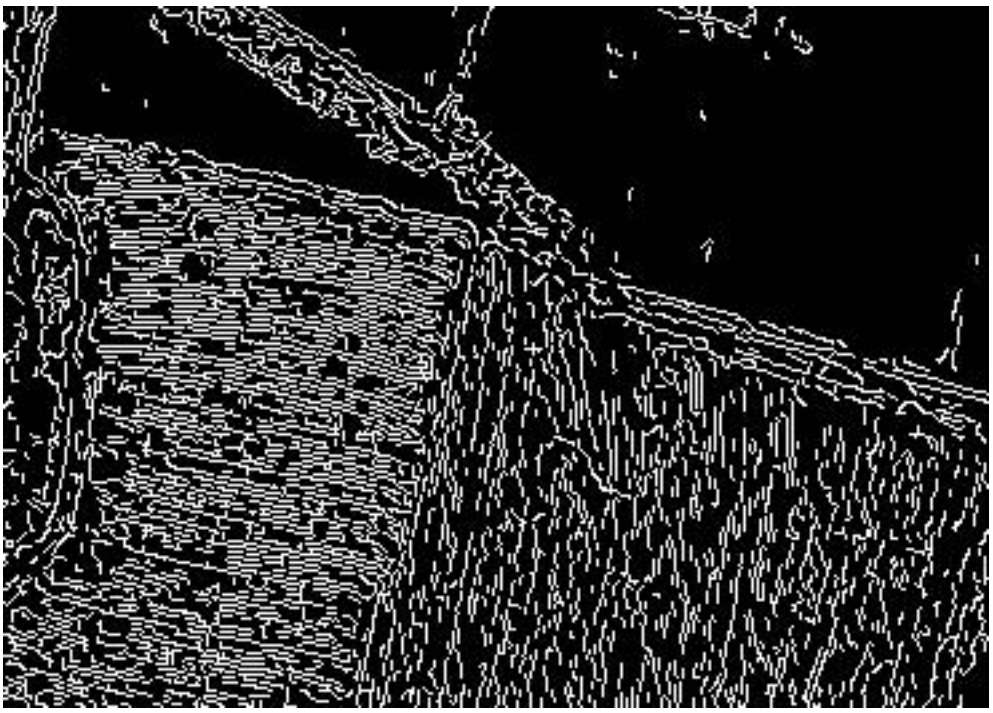


Figure 4.1: The outcome of the edge detector is the input of the first step: line collection.

The program stores the coordinates of the pixels of each segment. The attributes of the segments are obtained from these coordinates. These attributes are used by the next step: the reconstruction of lines.

4.3 Step 2: reconstruction of lines

Once the data that defines the lines are well structured, the next step is to reconstruct the lines. The goal of this step is obtain longer lines from the segments collected before. As is mentioned above, the segments collected may be little segments of long lines. And the study of parallelism is more reliable if it is based on long lines. So this step tries to connect different segments of the same line between them. The problem is that the image is full of segments of different lines. And the program has to decide which segments form a single longer line. There are two important parts in the reconstruction step: make a search region and choose the tight candidate between all possible segments that may be the following part of the line.

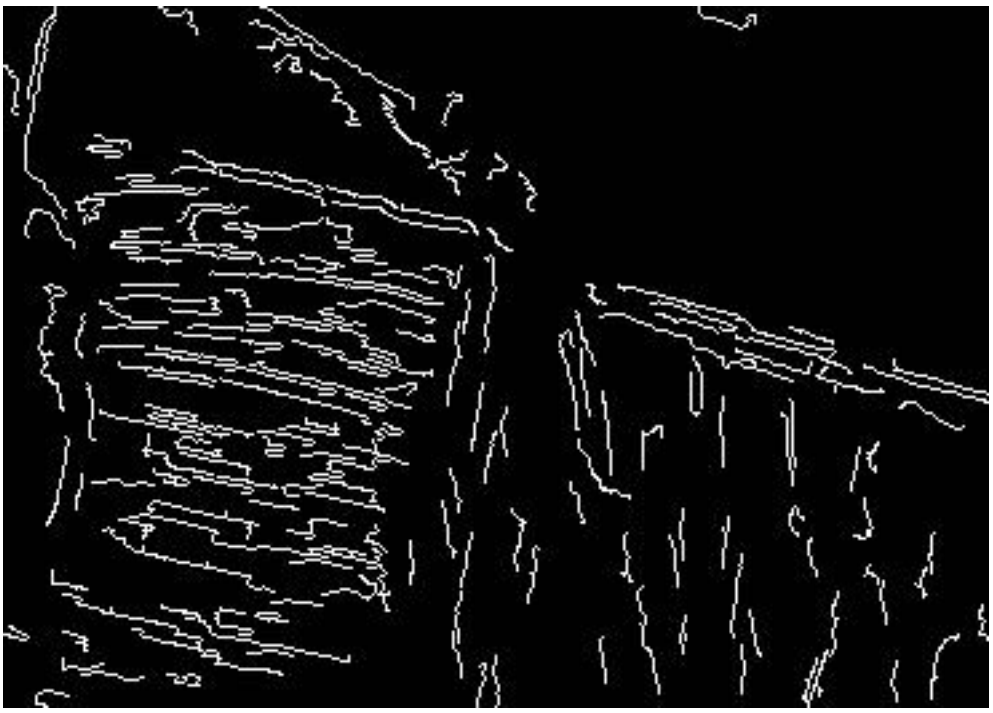


Figure 4.2: The result of the first step. These lines have passed through a filter of length: the second step works with the longest lines.

4.3.1 Direction vectors: parameter d

The slope is needed for curvature comparison, since the main goal of this study is find regions with parallel lines. Every line needs to have 2 estimated lines in the reconstruction step of this study. The slope of these lines has to be equal to the slope at the endpoints of the lines because this slope indicates the direction to the search region. The slope is extracted from the direction vectors. These vectors are normalized to 1. From one pixel to the next are there 8 possible directions with 45 degrees of resolution, but it is a margin too big. With the aim of increasing the resolution, the direction is calculated with more distant coordinates, instead of a point with the closest one.

$$\begin{aligned} vectorofdirectionX[i] &= coordinateX[i+d] - coordinateX[i-d] \\ vectorofdirectionY[i] &= coordinateY[i+d] - coordinateY[i-d] \end{aligned} \tag{4.1}$$

The greater the d parameter is, the greater the resolution is, as shown in figures 4.3 and 4.4. But the parameter d has limits because the higher the d parameter is, the lower the number of directions obtained is. The algorithm looks for several directions, because the parallelism between 2 curve lines is not indicated by a single direction as in straight line case. But it can simplify with a reasonable number of sections. Thus, this parameter depends on the length of the lines collected.

When parameter d is big, it can mean that the slope found at one point of the line

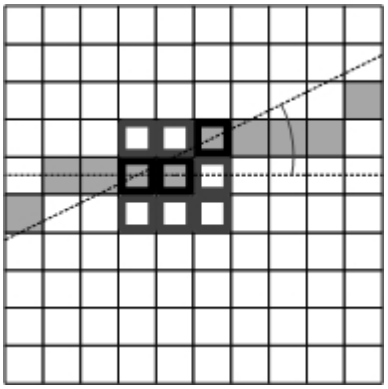


Figure 4.3: $d = 1$ resolution= 30

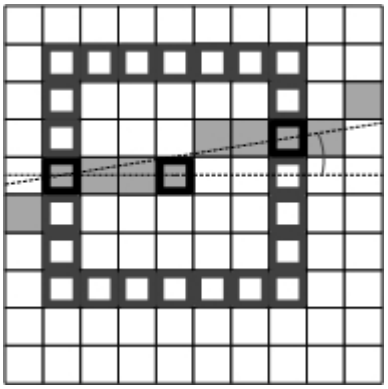


Figure 4.4: $d = 3$ resolution= 9.6

maybe is not related with the physical tangent direction in this point. But is a direction that the line has in one point between coordinates $i - d$ and $i + d$.
According to the expression of 4.1 the directions of the d points of the ends of the line have to be solved changing the parameter, decreasing it by 1 every point nearer the end. Is well-known the last point has infinite number of slopes. So the last value of direction vector is calculated with the penultimate coordinate where d is equal to 1. It has the worst resolution. So the slope in the ends of the line is an average of these last d values of X and Y coordinates.

4.3.2 The search region and candidate assessment

The first is related with the region where it has to search. The following part of the line may be anywhere but there are some areas more probable than others. The region is triangle shaped because if the following segment is near, its location cannot be far of the last direction of the line. This location can vary depending on the distance where it is located the other segment. It is important to note that there are probably several segments in this region. All of them are stored as candidates to be the following segment of the line. Beforehand, it is unknown which is the segment that belongs to the original line. That is why the program must do a study after obtaining several segments which are candidates to follow the line.

The figure 4.5 shows the typical example of searching candidates for the line reconstruction. The line named "line" in the figure is the main line in the algorithm. Is the line which the process wants to make longer. The rest of the lines collected before, while the process is working with this line, are possible candidates. The endpoints of all lines are clearly marked with a dot. The algorithm searches for lines and it detects the endpoints

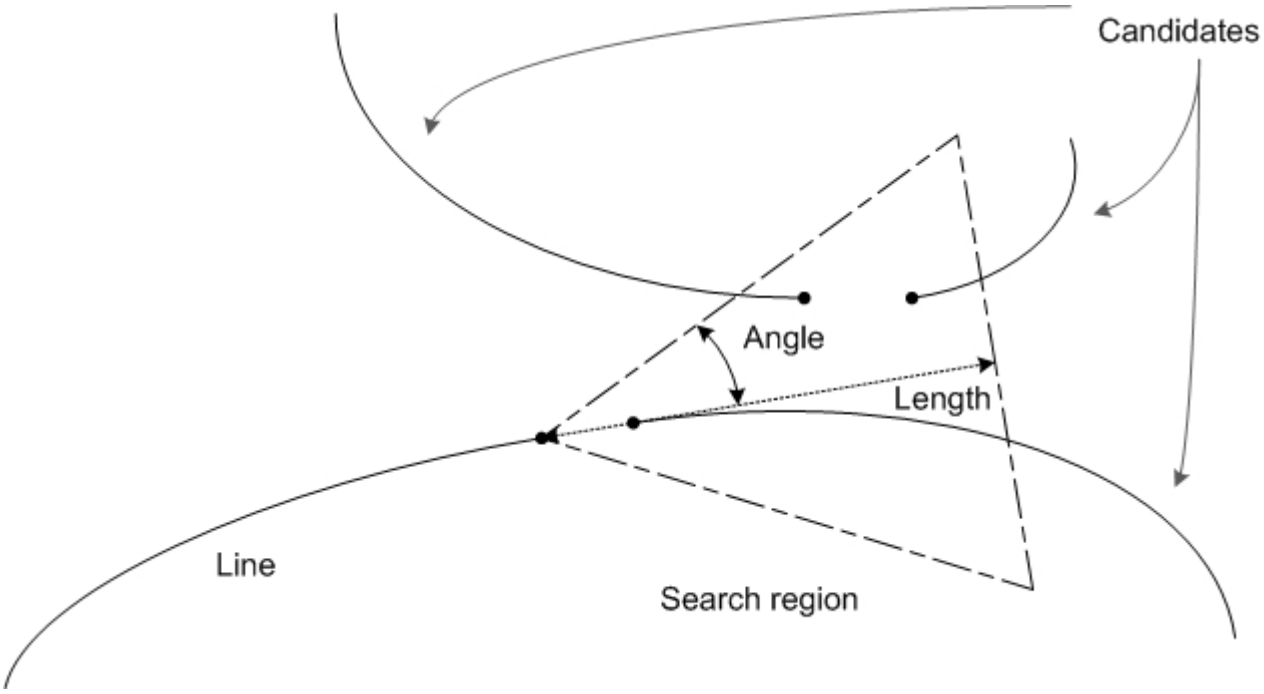


Figure 4.5: Example of a segment ("line") looking for the following segment among several possible candidates in order to form a longer line. The triangular region is the searching region. The program looks for the endpoints of these lines within the region. Then, the program has to decide which segment is the best to follow the "line".

of the lines. Put another way, the algorithm searches for candidates through the endpoints of these lines.

The algorithm searches for lines and it detects the endpoints of the lines. Put another way, the algorithm searches for candidates through the endpoints of these lines. The lines which become candidates are those lines whose endpoint is in the search region.

The figure 4.5 shows the triangular shape of the search region. Its orientation follows the estimated line in its endpoint. The length and the angle are parameters to set up. And these parameters change the search region.

Enlarging the opening angle and shortening the distance of search, for example, may be a change. This could be useful in cornering lines. Although straight sections with long gaps could not be rebuilt. A large length and a short opening angle would allow the program to rebuild these straight sections with long gaps. But the sharp curves could not be rebuilt. Only if the gaps are really short. Because if the gap is short, the segment

is not in a large angle related to the estimated line.

This study, the second part, focuses on the direction that the segment would follow if it was a straight line. The most of the lines collected are short lines and the search region is small compared to the image. Otherwise it would take too many operations and the program would become a very slow process. So in a short distance, the most probable direction is the more similar to the last direction of the last few pixels of the segment.

The process starts with a single segment, the longest one. It searches candidates and chooses the best of them. The program searches in the both endpoints of the segment. When a new segment is connected to the first one, the resultant line is treated as a new segment, because it has a new endpoint and a new direction. The search begins for both endpoints and it finishes when no valid segments are found. Then the process continues with another single segment. The reconstruction of lines finishes when all segments are processed.

Setting the search region

As the figure 4.6 shows, two orthogonal lines are made from the end of the estimated line which becomes the central line of the triangle. The length of these two lines is the aperture of the triangle. And it depends on the angle parameter. Thus, it can detect a segment in a curve shaped line. But also can detect more lines around it, even if their directions are completely different. That could cause false reconstructions, because the candidate assessment would be harder as higher is the number of the possible following segments. The program builds two lines because, as the program makes lines, it is easier than make a centered one.

In this case was there a problem related with vertical line. The slope of the orthogonal lines is solved with the expression 4.2.

$$b = \frac{-1}{a} \tag{4.2}$$

And here, the program has to distinguish 2 cases: when the slope of the estimated line is equal to 0 and the rest.

In the first case, this line may be vertical or horizontal. When it is vertical the expression 4.2 is useless for the computer because $a = 0$ and it is not the real value. The routine checks that it is vertical, puts the orthogonal slope equal to 0 ($b = 0$ in the expression

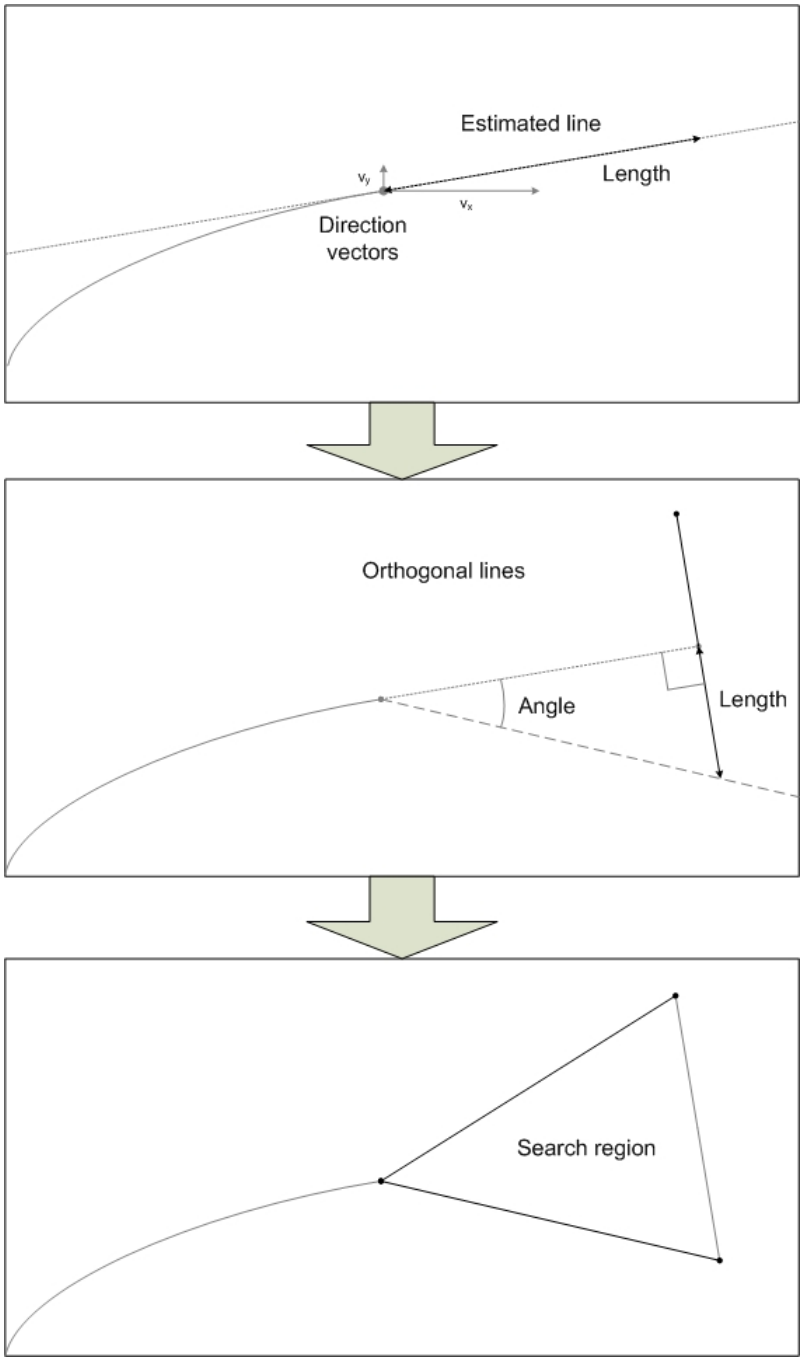


Figure 4.6: Steps of the search region construction. The first step is to build the estimated line in the right direction. This line has a certain length. The second step is to build the opposite side of the triangular region. That side is an orthogonal line to the estimated line. Its length determines the aperture angle of the region. Finally, the vertex is connected with the endpoints of the oppsite side. The pixels inside this triangle are the search region.

4.2) and makes a horizontal line. When the estimated line is horizontal, $a = 0$ is its true value but it does not work either with the expression. The algorithm keeps the slope equal to 0 but put the attribute, which decide whether the line is vertical or not, a *true* value in the orthogonal line. So, the orthogonal line is vertical.

In the second case are also necessary the attributes of state. These attributes depend on the endpoint, where the estimated line begins. This line can be built from left to right or vice versa. So the sign of the slope at this point is important. Otherwise, the orthogonal lines have not the correct slope and the region is not well done. As the figure 4.7 shows, the sign of b depends on the "dir" attribute because the b sign depends on the direction of the line.

Finally, in the third square of the figure, the program builds two more lines from the ends of these to the beginning of the central one. The program searches inside this region for the candidates to be the following segment of the line.

Making lines: parameter dir

: This parameter helps to create a straight line following the last slope of another line. The last slope which can be calculated from this line is stored in the direction vectors. And from these vectors the program obtains the parameter "dir". This parameter helps to make the line, setting parameters for each value. The figure 4.7 shows the values of "dir" parameter for each angle or range of angles.

$$y = a \cdot x + b \tag{4.3}$$

$$y[i] = y[i-1] + a \tag{4.4}$$

$$x[i] = x[i-1] + c \tag{4.5}$$

In the case of a vertical line:

$$y[i] = y[i-1] + c \tag{4.6}$$

The expressions 4.3, 4.4, 4.5 and 4.6 shows how the program makes a line. It adds the slope value (a) and c is used to indicate whether the line goes to the left or to the right. The old x and y values ($x[i-1]$ and $y[i-1]$) are used too.

For instance, if "dir" is 4, the program sets the slope to 0 and it creates the line from right to left. So, in the expression 4.5, c is -1. The direction 3 has a positive slope value

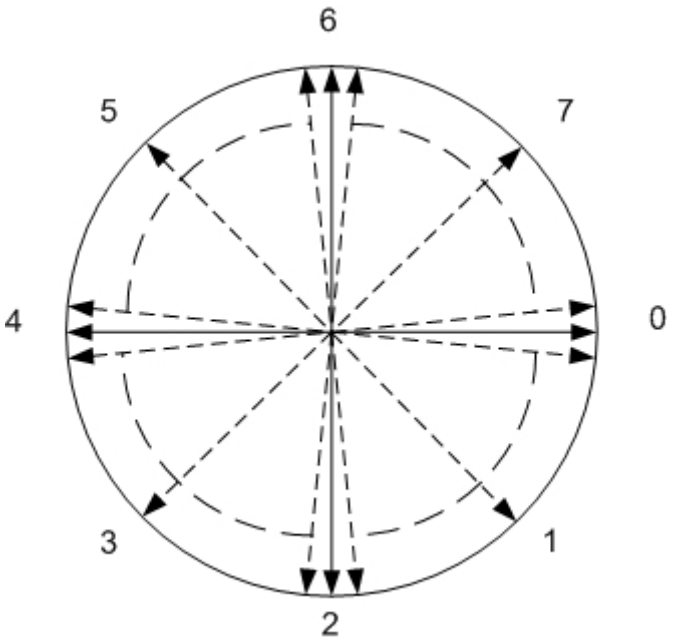


Figure 4.7: Angles for each dir value: 0 [0], 1 ($\frac{3\pi}{2}$,0), 2 [$\frac{3\pi}{2}$], 3 (π, $\frac{3\pi}{2}$), 4 [π], 5 ($\frac{\pi}{2}$,π), 6 [$\frac{\pi}{2}$], 7 (0, $\frac{\pi}{2}$). This parameter helps to set the parameters for making a line.

but "c" is also negative. The direction 5 has both parameters (slope and "c") negatives. And the direction 6 also has "c" negative.

4.3.3 Connecting lines

To connect lines the program uses a function which makes straight lines from the start point to the end point. It only needs the coordinates of two points. The algorithm makes a line that connects the main line with the chosen candidate. To do this, it must take into account the order of the coordinates stored of two segments. Then, the program calculates the slope needed to build this line in order to connect the two endpoints.

This algorithm is focused on lines with little gaps. A short straight sector in a long line is not the best reconstruction. But it has not to affect in a measurable way. And it allows the program to spend less operations than other interpolation methods. The aim is not the best reconstruction possible, but it is the obtainment of longer lines for a better study of parallelism.

The figure 4.8 shows segments collected after the first step. There are lined segments. The human eye can detect a structure of five lines but the program only detects seven-

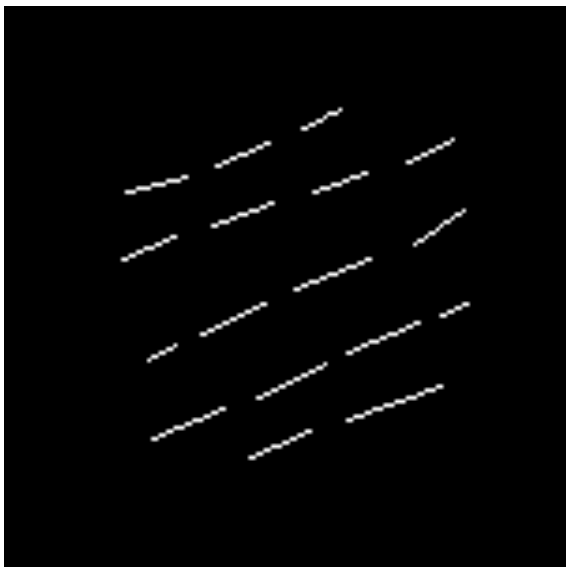


Figure 4.8: Lines with gaps. Segments collected in the first step.

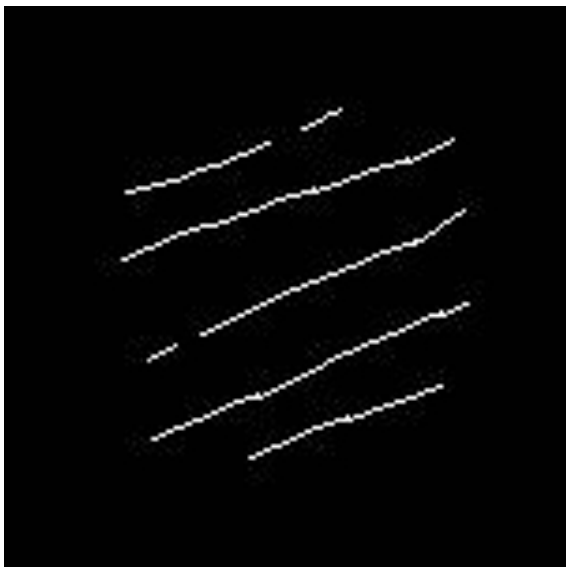


Figure 4.9: Lines after the second step. In this case, this step has closed the most of the gaps.

teen segments. In the figure 4.9 can be seen five long lines and two short segments. The reconstruction is not completely performed but the program obtains longer lines.
At this point, the program has the best data it can obtains, so it is ready for the next step: the study of parallelism.

4.4 Step 3: study of parallelism

In this third and final phase of the study, the program has a vector with lines collected and reconstructed. This allows study fewer lines, these ones are longer and the results they obtain are more significant. Now the aim is to study whether some of these are parallel among them.

Here there are several difficulties. The first is that the program has the lines and data related these lines. But the program has no geographical perception of what lines are close or far in relation to the image. Where should it search all possible parallel lines?

At first, the endpoint coordinates seems again the reference to search the possible parallel lines. But there are parallel lines which do not have the endpoints close each other. Maybe a line is not as well reconstructed as another one, so one is longer than another. There is no assurance that the reconstruction has been completed. This is the error propagation. A mistake in the previous step causes difficulties in this task. Like the errors in the first step makes harder the success of the second step.

The region to search should be short enough to save operations and time execution. But it should be large enough to check relatively distant locations to a line. The orthogonal straight line to the main line is the region chosen. More specifically, the region is formed by the orthogonal line and its neighboring pixels. The line which crosses through this region is a good candidate to be a parallel line to the main one.

Then, when the candidates are found, there is another challenge: how to determine whether they are really parallel to the main line or not.

In figures 4.10 and 4.11 is seen as the same curvature of non-straight lines is not enough to say that the lines are parallel. That is also dependent on its position in space. The left figure shows two curves with the same slope. But they are not parallel because of their relative position. Based only on the comparison of their slope vectors, the program could have concluded erroneously that they are parallel.

The orthogonal line used as search region, now is useful again. The comparison of the direction vectors of the candidate with the direction vectors of the main line cannot be the only process, as is mentioned above. Thus, the program checks whether the candidate is orthogonal to the orthogonal line. The figures 4.12 and 4.13 show the positive and negative case. The negative case shows two lines with the same slope variation but they are not parallel. The positive result, when A and B are equal, is needed to consider a line as a candidate to be a parallel line. Then, the comparison of direction vectors is done.

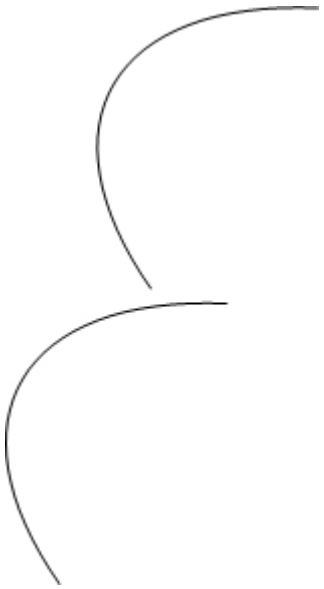


Figure 4.10: Lines with same slope variation but non-parallel. A orthogonal line which crosses the middle of a curve line would not cross the second one.

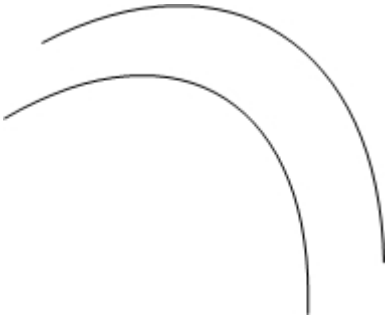


Figure 4.11: Parallel lines with the same slope variation. A orthogonal line would cross both lines and would be orthogonal to both lines.

Maybe two curves in a field are not completely parallel. Their slopes may differ slightly. But they have to be considered as parallel lines. Even though the program has found an agriculture land, maybe the lines do not fulfill these requirements. And the reconstruction of the lines cannot be considered as a strict one. Therefore, a relaxation of the requirements is necessary. The comparison of direction vectors uses a threshold in order to make the requirements more flexible. This relaxation also is needed because the errors propagation.

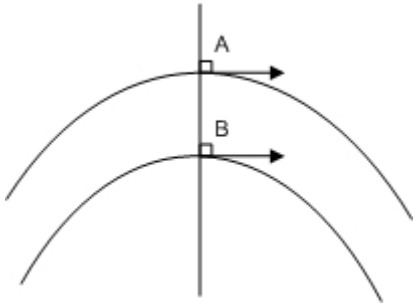


Figure 4.12: Parallel lines with the same slope variation. The orthogonal line crosses both lines and it is orthogonal to both lines. A and B angles are equal to 90 degrees.

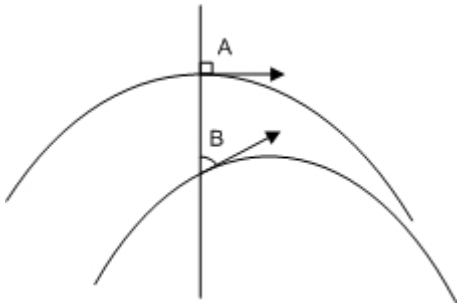


Figure 4.13: Lines with the same slope variation are not parallel. A is a 90 degrees angle, but B is not. The bottom line would not be considered a candidate.

5 Implementation

5.1 Step 1: collection and storage of segments

Three main processes compose this first step: the image scanning, the line tracking and the storage of data. The goal of these processes is to obtain segments as objects. Each segment becomes a variable of line class. This variable contains properties and information about the line. Thus, at the end of this step, the program has a vector with as many variables as lines have been found.

5.1.1 Image scanning

This scanning is done pixel by pixel from left to right and top to bottom. Every grey intensity value of the pixel is compared with a threshold. If the value is over the threshold the method stores the coordinates of the pixel in a vector. In this way, the method has to scan the whole image searching all possible lines which are made up of lined bright pixels. As important as get the coordinates of pixels is to get the coordinates of the pixels that form a line all together and well-ordered. For this reason, once a bright pixel is detected, is necessary to track other points connected to this one. Otherwise the process could get all coordinates but it could not distinguish which of them are lines and which others are single isolated points. Thus the data are useless because it does not let understand the meaning of them.

But the necessary tracking of the line poses a difficulty: it changes the order in the scanning. This is because when a white pixel is detected starts the tracking of the line in order to store all the lined and connected white pixels as a single line. After that, there is no specific order in the scanning and it depends on the newly found line.

The order is fundamental for not missing any point or line. The figure 5.1 shows the normal order. The dark grey squares are the scanned pixels. This image is the scanning state before it finds a white pixel. The arrows in figure 5.2 show two options: to continue to right following the scanning normal order or go back to the next pixel as if the tracking function has not existed. If the process follows the second option the problem above mentioned becomes real and many regions may stay unexplored, as shows the

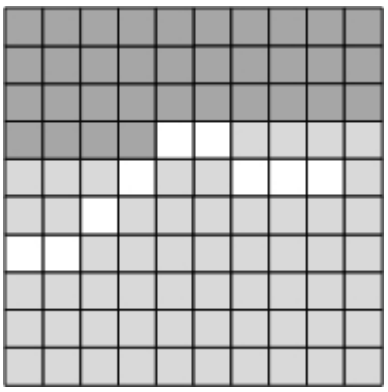


Figure 5.1: Standard scanning. The dark grey squares are the scanned pixels. The grey pixels are dark pixels which have not been explored.. The white pixels are pixels which belongs to a line.

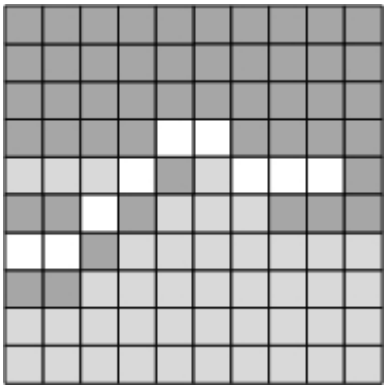


Figure 5.2: Image after tracking. In the region explored are a few pixels unexplored.

figure 5.1.1. Clearly, the first option has to be chosen to ensure the exploration of the whole image.

So the algorithm must know which pixels have already been scanned and it has to be able to back to the position of the last scanned pixel before finding the line. For this purpose the algorithm uses a matrix as big as the original image, shown in the figure 5.5. Each position in the array represents a pixel of the image. These values are Boolean and have value 1 when a position has already been scanned and value 0 for the positions of unexplored pixels. The first step in the scanning process is to check the location state in the Boolean matrix. This matrix is used as a map to the tracker to avoid exploring areas several times and to get the same line more than once. Checking Boolean matrix is not a heavy process. This makes the algorithm more efficient and also the algorithm becomes more reliable because it scans the whole image and the program does not miss a line.

5.1.2 Tracking of the line

The algorithm that tracks the line is called when the scanner finds a pixel with a value over the threshold. This means that it has found a pixel of a possible line.

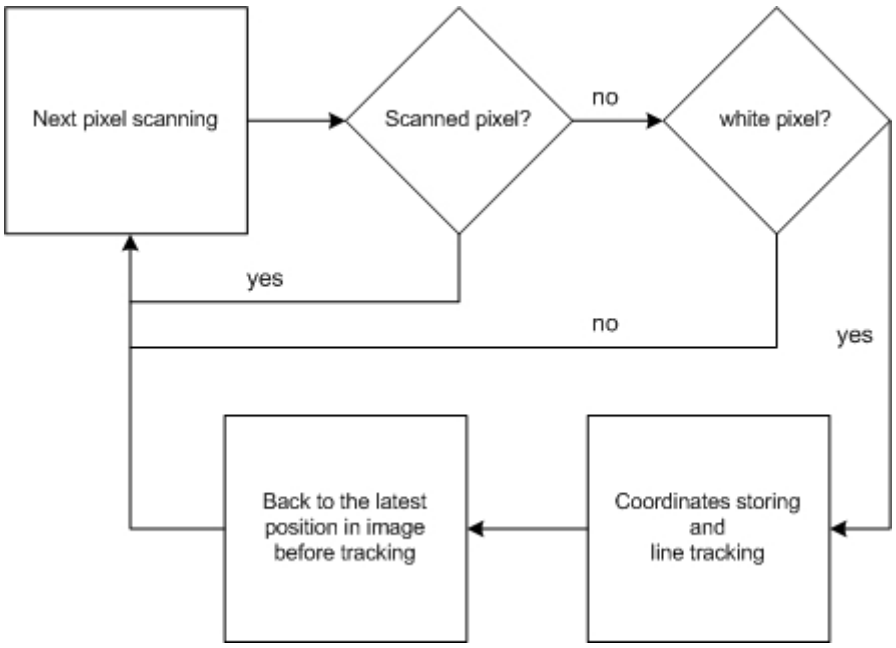
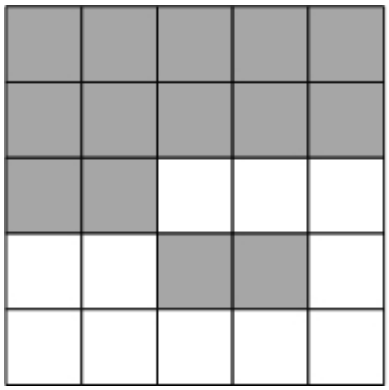


Figure 5.3: Process scheme of image scanning.



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

Figure 5.4: The dark cells are scanned positions. The white cells are unexplored positions.

Figure 5.5: The Boolean matrix. The value 1 is a scanned position and the value 0 is an unexplored position.

But the first pixel that the algorithm finds maybe is not an endpoint of the line, as shown in figure tracking backward. And for an ordered storage of the coordinates of the line is necessary an ordered tracking. A vector with coordinates arranged allows the algorithm to extract some properties of the line.

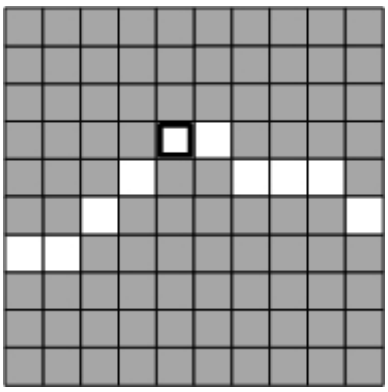


Figure 5.6: Line which needs to be tracked backwards. The marked position is the first white pixel of the line which has been found.

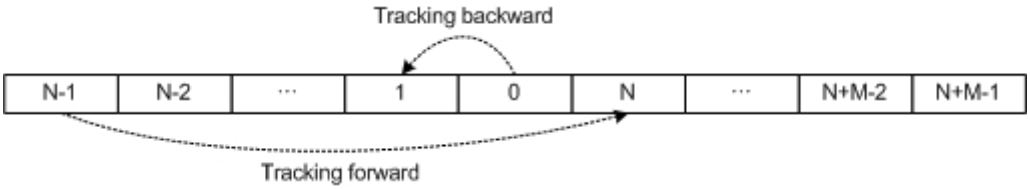


Figure 5.7: Storage of coordinates in a vector in tracking process. The N-first coordinates are stored in reverse way. The M following pixels are stored in a normal way.

Thus, a tracking backwards begins when the first white pixel has been caught. The goal is to find the start point, the point which has to be the first one stored in the vectors of coordinates of the line. The coordinates of the pixel position are stored in two vectors of integers, one for X coordinates and one for Y coordinates. In the tracking backwards of the line, every pixel position is stored in the first position of these vectors. When the line tracking ends means that the beginning of the line has been found. Then the normal storage of the coordinates in vectors continues. The figure 5.7 shows how the N-first coordinates are stored in reverse. That is, the first position of the vector contains the coordinate of the last pixel tracked by the tracking backward process. Then are there M pixels, after the aforesaid N pixels, stored in a normal way. Last position of this vector contains the coordinate of last pixel obtained by tracking forward method.

After the tracking backward finishes, the tracking forward algorithm runs at the first white pixel of the line found. It does not run at the starting point which has just been found. Then, in every pixel, it scans the eight neighbors in clockwise direction. The starting position is the number zero on the figure 5.8. This is the first because the scanning order makes that the next white pixel has to be located on right or underneath.

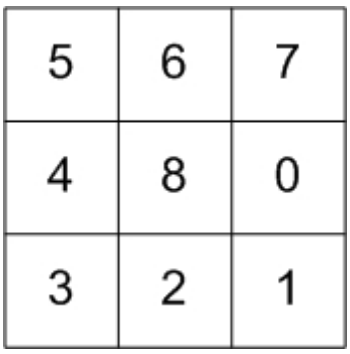


Figure 5.8: 9 squares with positions from 0 to 8 according to tracking order.

When the algorithm finds a new white pixel, it stores its coordinates. Then it marks the location as scanned in the Boolean matrix and starts again the scanning of the new pixel's neighborhood. This tracking ends when it does not find a new white pixel.

When the algorithm scans the neighborhood of the pixel, it has to check that the pixels are really inside the image. The pixels round the picture, out of limits of the matrix, have a random value and they can produce false detections.

The difference between the two tracking methods is the scanning of neighborhood of the white pixel. When the process works forward, the first scanned position of the neighborhood is always the same. The position number 0 is the starting position and the process follows scanning the other positions in the clockwise direction, ie that it goes from 0 to 7 in the figure 5.8. This is an orderly way since this method scans the array by rows from left to right.

However, in the tracking backwards, the starting position in the scanning of the neighborhood is not fixed. This position depends on the latest position where a white pixel has been found. If k is the counter of the scanned position, this process scans firstly the position $k - 1$ and follows in clockwise direction. Thus, the algorithm scans first the most probable positions, where the algorithm can find the pixels which continue the line, because these are in the nearest angle.

5.1.3 Storage of lines: line class

Every line is a single variable object. The aim of this section is not to do an extended explanation about the details of the line class. But there are some aspects about this class which have to be known and understood with the aim of a better comprehension

of what it can be done and how it can be done.

In the first step, the coordinates of every line are stored in objects of the vector class. This allows an easy management of data. The algorithm can insert elements in any position of the vector without moving the rest of the elements. This is an important advantage that the algorithm of line tracking uses all the time. Another important characteristic of this class is the flexibility of the vectors' size. Usually the program has to know the size of vectors before using them. But in the image scanning, the program does not know neither how lengthy will be the line nor how many lines there will be. With this class, the program can store the coordinates and lines with no idea of the number. And vector class has an additional advantage related with the size of vectors. It has a method to get the size easily without having to program specific algorithms to do it.

Several attributes of the line class are vector class such as the coordinates of line, the coordinates of estimated straight lines and the values of the direction vectors in every point of the line. The coordinates are stored in 2 vectors, one for the X coordinate and one for the coordinate Y. And it does the same for direction vectors.

The identifier of the lines is the position in the main vector which is an object of the vector class too. And every line has the information (identifier) as an attribute of which of the lines are parallels to it. This attribute is a vector with the position in the main vector of every line parallel to it.

Related to the coordinates, there are vectors which contain the coordinates of the end points of the line. But these are not vector class above mentioned. They are normal vectors with 2 positions (X and Y coordinates). These endpoints will be important in the next step for locating lines.
The size matters because the scanning process collects many lines too short. And most of them have no real significance for image interpretation and they cause that the algorithm makes mistakes. The program makes an initial selection according to the size.

The state vector along with other attributes is designed to solve the problem of the vertical line. These are Boolean variables which contain different features of the line, such as vertical position and the direction of it (according to the order of its coordinates).

5.1.4 Limitations

Vertical line

The line is defined by the coordinates of its points stored. The calculation of the derivative value depends on the position of every coordinate. These are ordered from left to right in order to obtain real values of the derivative. In the case of a vertical line, it is stored from top to bottom.

A line can be totally or only partly vertical. There are flags attributes in line variable for any case: totally, only the left end part, only the right end part. The curve can be vertically at both ends. In this case the left and the right flags variables would have the true value.
The value of the derivative of a vertical line is difficult to extract. The real value of its slope is infinite and the computer gets this result. The problem is to use it. In order to estimate lines following the slope at the ends of the line, the derivative value can't be infinite because it can't be useful for calculations.

It is important to know that, in the program, the slope is 0 in a horizontal and a vertical line. Thus, when the derivative is infinite, there is an specific number and it cannot be confused by an calculation error. The lines objects have a Boolean variable which is true only when the line is vertical. This attribute is necessary in order to put a value in slope variables. When a line has its slope value equal to 0 the program always checks this attribute.

The slope is needed for curvature comparison, since the main goal of this study is find regions with parallel lines. Every line needs to have 2 estimated lines in the reconstruction step of this study. They are necessary to decide which segment should be the following part of a certain line. But the algorithm makes these lines adding the slope value every X step (expressions 4.3, 4.4, 4.5 and 4.6)and filling vertical gaps when a (in 4.4 is bigger than 1.

The slope is extracted from the direction vectors. These vectors, as explained in Chapter 3, are the variations of the lines in the coordinates X and Y.The problem exists if X value is 0, because the slope would be infinite again. If the algorithm detects X value is 0 or it is below a threshold, the algorithm classifies it as a vertical end and the Boolean variables above mentioned have *true* value. Then, a specific algorithm is necessary for making vertical lines. This algorithm only can make vertical lines, but doesn't need the slope value. It just goes through the Y coordinates for the same value of X and it only has to choose the right direction, upwards or downwards.

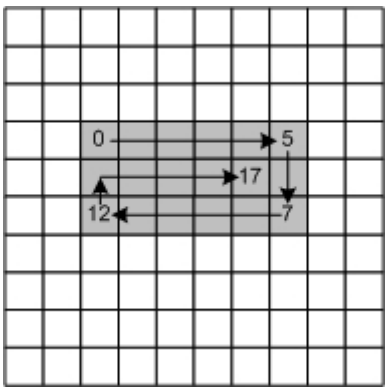


Figure 5.9: Line 3 pixels wide and its storage. It is detected as an spiral line.

Direction vectors

A deeper analysis of the expression 4.1 shows the importance of the order of the coordinates to obtain accurate and consistent results. Even if the line is a straight line, the wrong ordered coordinates can cause false results.

As seen in the tables 5.2 and 5.1, the algorithm needs the coordinates well-ordered. The Δx calculation depends on the position of the coordinates in the vector. The tables show how the process works in the ends of the line. The d parameter changes along the ends. The first and the last position have no value neither the d parameter nor the direction vectors.

The table 5.3 shows, in grey cells, the mistakes (or false direction vectors) caused by the wrong order of the coordinates. These are stored in two halves; the first part is at the end, and the second one is at the beginning. So when the Δx arrives at position 11, the distance the algorithm is taking is false. And it causes false direction vectors. In this case they are not too odd, and that is the worst because they may be undetected.

Other errors may have more strange values in the line context. And the program could interpret that the line has sudden direction changes like a line with sharp curves. Therefore, two straight parallel lines could be interpreted as non-parallel. The results in this table and the above conclusions highlight the importance of having well-stored the coordinates.

pos	d	Δx
0		
1	1	$x[2] - x[0]$
2	2	$x[4] - x[0]$
3	3	$x[6] - x[0]$
4	4	$x[8] - x[0]$
5	4	$x[9] - x[1]$
6	4	$x[10] - x[2]$
7	4	$x[11] - x[3]$
8	4	$x[12] - x[4]$
9	4	$x[13] - x[5]$
10	4	$x[14] - x[6]$
11	4	$x[15] - x[7]$
12	4	$x[16] - x[8]$
13	4	$x[17] - x[9]$
14	4	$x[18] - x[10]$
15	4	$x[19] - x[11]$
16	3	$x[19] - x[13]$
17	2	$x[19] - x[15]$
18	1	$x[19] - x[17]$
19		

Table 5.1: Calculation of direction vectors.

Wide lines

The fact that lines are 1 pixel wide is the base of the algorithm of collecting lines. The tracking system does not allow collect a wider line and recognize it as a line wider than 1 pixel. Thinking about a wide line as a two-sided polygon, the tracking system would collect first one side then the other side as if it were a continuation of a single curved u-shaped line. If the line were 3 pixels wide, the collected line would be a spiral, taking first the exterior pixels as a u-shaped line, then the pixels inside. As the figure 5.9 shows, a line of 6 pixels long and 3 pixels wide is stored as a spiral line 18 pixels long and 1 wide. It would be a serious problem because the lines are defined by the coordinates stored.

5.2 Step 2: reconstruction of lines

The processes that compose this second step are the construction of the search region, the candidate assessment and the connection of segments. The first one takes a segment and looks for other segments which could follow it. This region may find several seg-

pos	x	y	d	$\Delta x/(2 \cdot d)$	$\Delta y/(2 \cdot d)$
0	4	18			
1	5	17	1	1	-0.5
2	6	17	2	1	-0.5
3	7	16	3	1	-0.5
4	8	16	4	1	-0.5
5	9	15	4	1	-0.5
6	10	15	4	1	-0.5
7	11	14	4	1	-0.5
8	12	14	4	1	-0.5
9	13	13	4	1	-0.5
10	14	13	4	1	-0.5
11	15	12	4	1	-0.5
12	16	12	4	1	-0.5
13	17	11	4	1	-0.5
14	18	11	4	1	-0.5
15	19	10	4	1	-0.5
16	20	10	3	1	-0.5
17	21	9	2	1	-0.5
18	22	9	1	1	-0.5
19	23	8			

Table 5.2: Calculation of direction vectors.

ments so the next process chooses the segment which fits better to the main segment properties. Finally, the third process connects these two segments with a third one. And it has sort the data in the resulting line.

5.2.1 Search region

Before creating the search region, the program processes the data of the lines to make work in this second phase easier. The data processing involves the calculation of several characteristics of the line. At this point, the program sets the Booleans attributes about direction where the line goes vertically and horizontally. If it goes from right to left, it rearranges the coordinates from left to right. As mentioned in last section, the order of the coordinates is crucial for calculating the direction vectors and these vectors are the basis for the next steps of the program.

There is a standard order after the rearrangement. Then, some others Boolean attributes can be set: the endpoints. So the left endpoint is the start point, and the line ends at the right endpoint. The search region is made in the endpoint and it starts with a vertex

pos	x	y	d	$\Delta x/(2 \cdot d)$	$\Delta y/(2 \cdot d)$
0	13	13			
1	14	13	1	1	-0.5
2	15	12	2	1	-0.5
3	16	12	3	1	-0.5
4	17	11	4	1	-0.5
5	18	11	4	1	-0.5
6	19	10	4	1	-0.5
7	20	10	4	-1.5	0.75
8	21	9	4	-1.5	0.75
9	22	9	4	-1.5	0.75
10	23	8	4	-1.5	0.75
11	4	18	4	-1.5	0.75
12	5	17	4	-1.5	0.75
13	6	17	4	-1.5	0.75
14	7	16	4	-1.5	0.75
15	8	16	4	1	-0.5
16	9	15	3	1	-0.5
17	10	15	2	1	-0.5
18	11	14	1	1	-0.5
19	12	14			

Table 5.3: False direction vectors.

on it. A direction line goes from this vertex to the center of the opposite side. This line is an estimated line and it points to the direction where the algorithm looks for following segments.

The estimated line is a straight line generated from an average of the latest direction vectors. Its length depends on the average of the length of the stored lines. That means how far away the region searches for the segments. If the most lines are small lines; it has no sense to look for far away from these lines. They may be lines with lots of gaps: they are small segments but not far away each other. So, the search region has not to be too large. The larger the region is, the more operations the program has to do. Because it has to analyze more pixels in the image and it has to compare more candidates. The most effective region cannot be too large and it has to look for in the right direction.

The parameters needed to make the estimated line are the length, the slope and the "dir" parameter. In the vertical line case, the slope is not a problem because is there an algorithm in the line builder that needs only the length.

The triangular shaped search region is created as the chapter 3 describes. The program

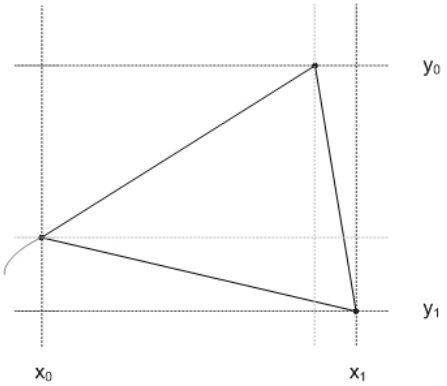


Figure 5.10: x and y limits.

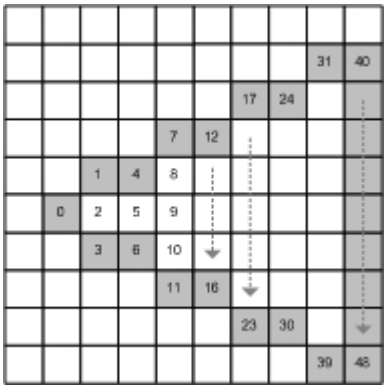


Figure 5.11: Storage of search region.

stores the pixels inside the triangular region in a vector. And it does not know its size because it may change. The coordinates are collected using a method which has the lines of the region as edges that cannot be passed. The endpoints of the two latest lines built mark the x and y values of the limits of the triangle. Then every point between the lower and the higher x value is checked.

The figure 5.10 shows x_0, x_1, y_0 and y_1 as limits for the storage of the coordinates of the search region. The points which mark the limits of x and y depend on the position of the triangle. So every time these points will have to be searched, since the position of the triangle may change with every endpoint of each segment. Regarding the boundary lines, their points are also stored in search region, as seen in the figure 5.11. It is noteworthy that the boundary lines occupy many pixels which are close to the extreme. Any of them could be, easily, one end of the next segment. Otherwise, it would be checked very few points near the end of the line and closest segments might be not found. This figure also shows the order in which points of the region are stored.

After establishing the limits of x the program checks if the point is a limit point in order to do not pass the limit. Then it stores the coordinate. All the coordinates inside the triangle are stored as well as the coordinates of the lines of the triangle shape. And all of them are the coordinates which the program has to check in order to find ends of other segments. They are the search region. The program compares these coordinates to the ones of the endpoints of lines stored in the first step. When the program finds a line in the region, it stores this line as a candidate in a vector of lines. The program only checks the endpoints of the lines because if it checks any point of the lines, the comparison would be harder with no better results. A line can pass through the region but if it does not end within it, that line should not count as a candidate.

A wrong line in the search region or a gap in one of its lines could break the algorithm. According with the algorithm for storing the region coordinates, if it never finds a border

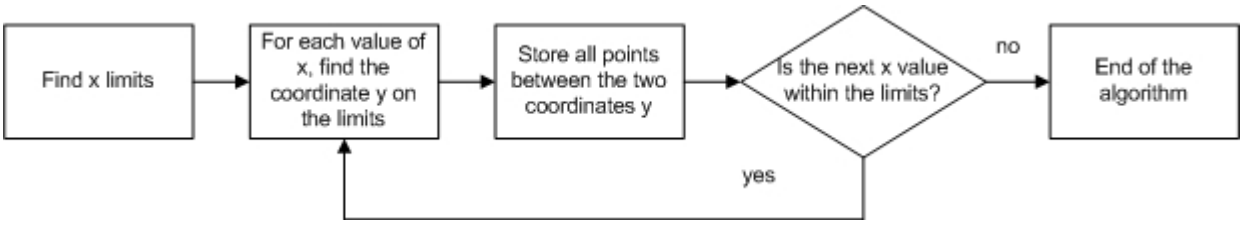


Figure 5.12: Diagram of the storage process.

line, it follows checking every pixel in the image. The program would have to handle a large variable after waiting for its storage. And it would have to check every coordinate of this huge vector looking for coordinates of line endpoints. If this does not cause problems with memory, it would last too much time. It would be working with too large data but it would look like that the program has stopped working. So the good construction of the search region is essential for proper functioning of the second step of the program.

5.2.2 Candidate assessment

This process in the second step runs with every endpoint of every line collected in the previous step. When the program has executed the first step, there is a vector with all lines stored. The search region is made for every endpoint of every line but in memory only is there one region at the same time. There is not accumulation of several search regions because that could cause memory problems. So the program makes a region, works with it and erases the region before making the next. Every time has to be a candidate assessment with all segments found. The better the assessment is, the better the reconstruction is.

The main information available to the program is the position of endpoints and the estimated line. The distance between endpoints could be a good parameter to decide if a candidate is the best. But distances in the region are not large enough to be significant. If the following segment is near the endpoint, their slopes should not be too different. Thus, the distance is important but is not an explicit parameter in evaluation.

So, the main comparison parameter to choose the following segment is an angle. This angle is composed by the line which links the main line and the candidate and the vector in the last direction of the segment candidate, as shows the figure 5.13. The cosine of the angle is calculated using the direction vectors. In the expression 5.1 α is the real angle between the two direction vectors of the lines $\overrightarrow{u}$ and $\overrightarrow{v}$. The value of the parameter *angle* depends on the value of $cos(\alpha)$, as seen in the expression 5.2.The goal is to limit the parameter value between 0 and 1. By this way is easier to decide which segment is

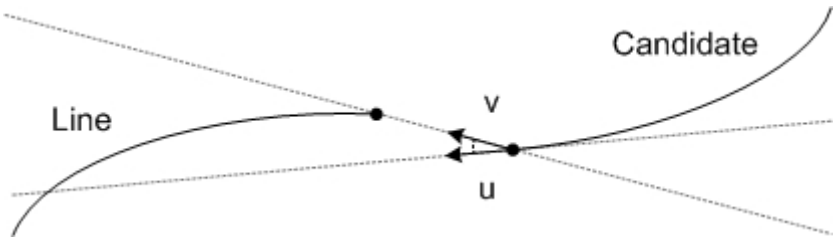


Figure 5.13: The angle of candidate assessment process is composed of the vectors u and v. v is the direction of the line that links the main line and the candidate. u is the direction of the endpoint of the candidate.

more lined up with the line. A value close to 0 means that the directions are equal to each other. Otherwise, if the value is close to 1, the both directions are orthogonal.

$$cos(\alpha) = \frac{\overrightarrow{u} \cdot \overrightarrow{v}}{\|\overrightarrow{u}\| \cdot \|\overrightarrow{v}\|} \tag{5.1}$$

$$angle = 1 - cos(\alpha) \tag{5.2}$$

This parameter is useful in order to rank all candidates. The candidate with this parameter closest to 0 and below a threshold becomes the following segment of the line. When the program has the best candidate it has to connect both lines. Before that, it has to be known the order of the coordinates in both lines. Its importance related to the study of the derivative has been mentioned above. It checks that a left endpoint is connected with a right endpoint of the other line. That is right if the $left$ variable has the same value in both lines. If they have different values, there has to be a rearrangement of the coordinates of the candidate. In the case that the algorithm tries to connect one left endpoint with another left, the reasoning is analogous. But, unlike the previous case, the value of $left$ would have to be different in both lines.

5.2.3 Connecting lines

In order to connect lines the program uses a function which builds lines from the start point to the end point. It needs only the coordinates of both points. The algorithm makes a line that connects the main line with the chosen candidate. To do this, it must take into account the order of the coordinates stored of both lines. The algorithm checks whether the line is vertical or not because it calculates the slope that the straight line has to have in order to connect the two endpoints. If the line was vertical, the value

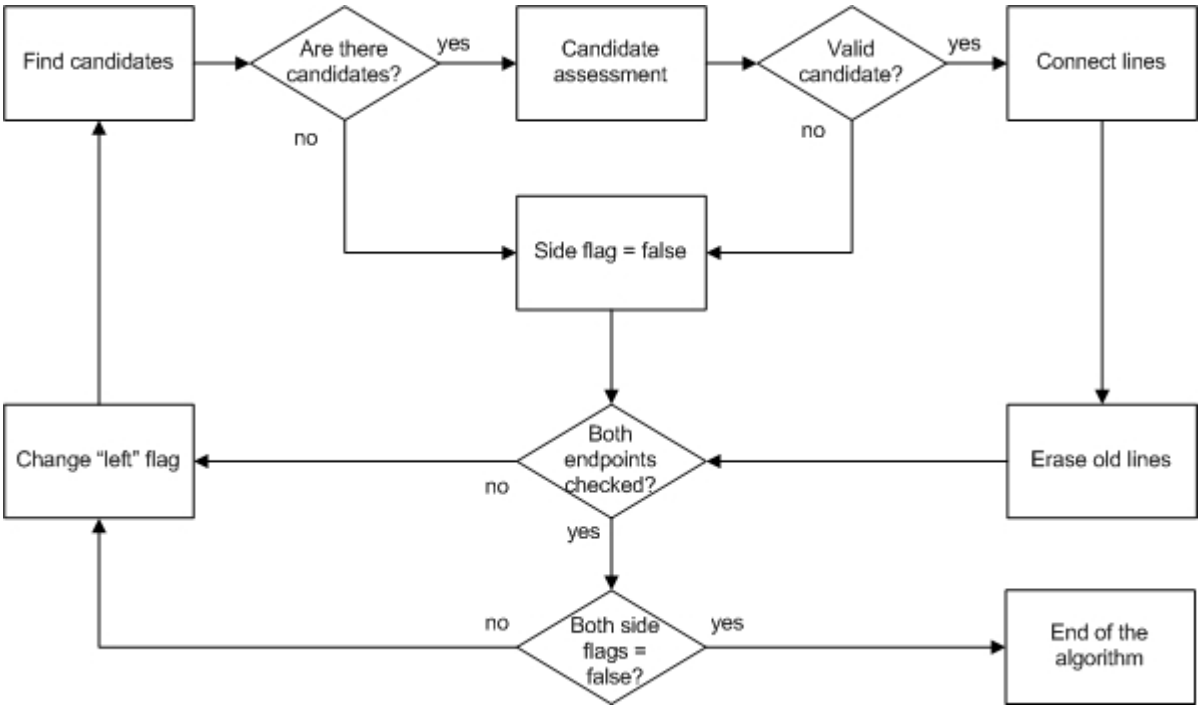


Figure 5.14: Diagram of the lines reconstruction.

of this calculation would be useless because the infinite number cannot be processed. Once it is made, it is analyzed to get its state variables and check the variable $left$. It may be that the connecting line goes from left to right or vice versa, so it is necessary to analyze it.

The connecting line and the new segment is stored in the main line, since it keeps the original order. The coordinates are stored in the same vector that main line. Here the program takes advantage of the vector class to add elements to the beginning without creating algorithms to shift the position of the elements in order to free up the top positions. It is important not to forget to update the data about the new line, as are the value of the new endpoint, the vector of values of the derivative and the value of the slope at the last valid point.

After updating of data related to the line, it proceeds to delete the segment and the candidate. This way it can perform a new search for the new endpoint. Searches for segments are made for both ends of the line which it wants to rebuild and they do not finish until no valid segment is found at neither of endpoint.

When this happens, the program goes on with another line stored vector lines. The goal

is to have a reasonable number of long lines rather than a lot of small segments. Lines of acceptable length allows do good studies in direction of the lines and whether these lines are parallel among them or not. When the program gets results from studies on segments, these are less reliable information because it is based on less data.

5.2.4 Limitations

This phase of reconstruction has some clear limitations. The search region is created from the perimeter of a polygon. If the perimeter cannot be established, the region cannot be created, so it cannot connect the lines. This happens near the edges of the image. The intention was to avoid problems with those areas outside the image, since the value of the pixels inside these areas is unknown. But this limitation, in big images, does not suppose a big deal because these areas near the edges are relatively small and, hence, little significant in comparison with the data processed in the whole image.

The union of lines with a sharp bend can also be complicated if the missing fragment is precisely the curve. Fortunately, the type of curves, for which the program has been designed, should not be too sharp.

There are curves whose analysis may be complicated, such as a C-shaped curve. In this case, it would be difficult to determine which end is the left endpoint and which is the right one. Probably the top left would be established as being the first to be detected in the case of the C was detected completely from the beginning. But a reconstruction of a C-shaped curve would give a different result. The data about this curve, once it is analyzed, could suppose problems for the parallelism study.

The figure 5.15 shows a C-shaped curve with a segment missing in the middle. The upper segment would have its left end and right well-defined, as the lower segment. In this case both lines would be joined by their left sides. But, in the complete new line, it is difficult to assign clearly a left endpoint and a right one.

This is because, at the upper endpoint, the coordinates go from left to right, as also happen in the lower segment. For the operation described above, the program would change the order of the coordinates of the lower segment and it would be stored as the start of the new line. That is, the coordinates of the lower segment would be in top positions in the vectors of coordinates. Thus, in the case of the C-shaped curve, the left endpoint would be the lower one and the right endpoint would be the upper one, considering that both ends are located right in the same coordinate X.

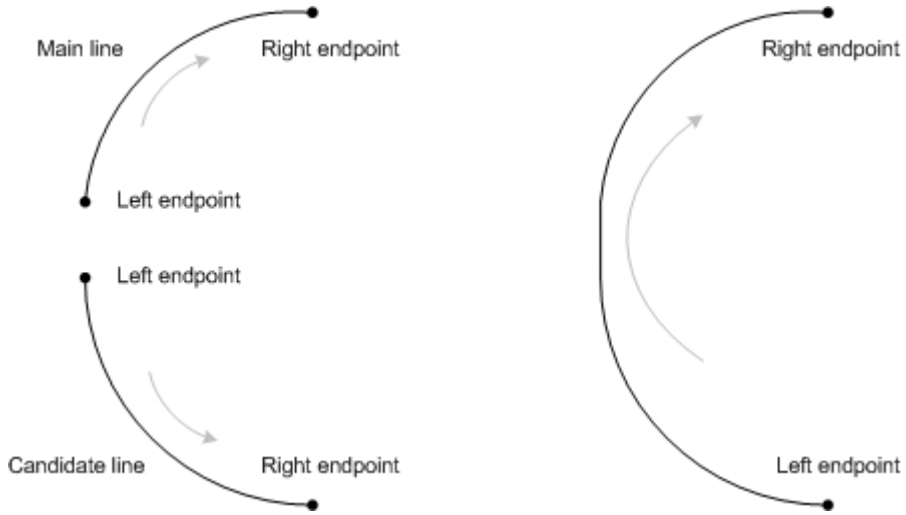


Figure 5.15: C-shaped curve example. The assignment of left and right endpoint is determined by how the program works. It depends on which side has joined the upper segment to the bottom.

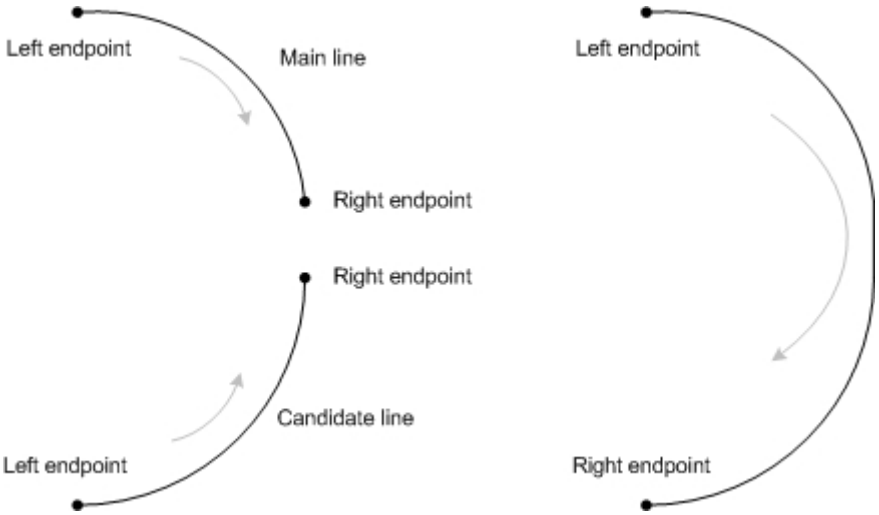


Figure 5.16: Reverse C-shaped curve example. It results the opposite that in the example of the C-shaped line.

In contrast, in the figure 5.16, opposite happens. The algorithm connects both right endpoints and changes the order of the coordinates of the lower segment. Up to this point, the same happens because the assumption that the main line is the upper one is the same in both cases. The upper segments are in top positions in the vector of lines due to the image is explored from top to bottom. The left endpoint of the lower segment becomes the right endpoint of the new line.

The reconstruction of lines using a straight line is quite primitive and only aims to provide continuity between the different segments, which initially formed a single line. This reconstruction itself is not the main objective of this work, since what matters is the decision about the existence of parallel lines in the image. That decision is taken after the reconstruction and, as is mentioned above, a good reconstruction allows a good study of the lines and gets more reliable results about its parallelism. One method that could be followed in future work could be the interpolation between the two lines must be tied together with Newton's method, for example. The union of straight lines may cause critical points where the derivative cannot be infinitesimal and a good interpolation may avoid these points. But the system used to obtain the slope data makes these points are no longer critical, since the derivative is not at a particular point, but that is the orientation of the line in a region. If the line were continuous and differentiable, then the derivative value belongs to a point. But the handicap of working with a finite number of points in this case becomes an advantage for the proper functioning of the algorithm because the program does not find these points as critical ones.

5.3 Step 3: study of parallelism

The final step is composed of two processes: the searching candidates and the comparison for the parallelism study. In this third step, the program has a vector with reconstructed lines. It has the lines in the best possible condition. The first process has to search the lines which may be parallel to a certain line. Then, the second process checks if these lines, the candidates, are parallel to the main line. These processes work with every line of the resulting vector lines after the second step. And their aim is to find groups of several parallel lines.

5.3.1 searching candidates

In order to find groups of parallel lines, the program allows each line to find its set of parallel lines to it. So each line has to make the best search on the region of the image where is located. This could lead to repeat searches or studies in a group of several lines, because the program would execute the search method with each line of the group.

So the variable line also has a variable which identifies the found lines parallel to it. Before doing the study on candidate lines found, the program checks this variable in these lines in order to avoid repeat the study of parallelism done before.

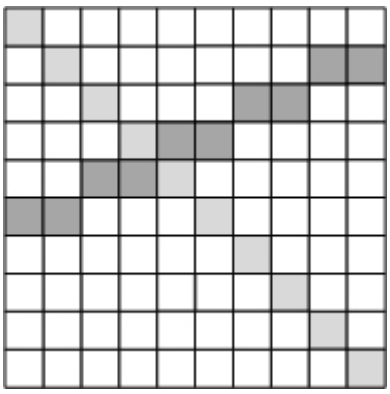


Figure 5.17: These two lines do not share a pixel. That is why the program has to check the neighboring pixels.

The search is based on a line orthogonal to the line that is trying the program algorithm. It can perform multiple orthogonal lines with each search and it would proceed in the same way. In the case of using a single orthogonal line, it would cross through the center of the main line. The program finds all lines that cross this orthogonal line. Specifically, the program looks for lines which share any pixel with this line. The program also searches the neighboring pixels of this line. This provides lines which are candidates to be parallel to the main line. These candidates are stored in a vector. The orthogonal line extends equally to either side of the main line.

This line is composed by two lines which start at the middle of the main line. Each line goes in the opposite direction, but they always are orthogonal to the main line. This searching region is these two lines and the neighboring pixels. That is because it might miss a line if it only searches in a 1 pixel wide line. The figure 5.17 shows the case that two lines cross without sharing a pixel. That would mean that the line is not detected because the program does not detect its intersection. The aim of searching in neighboring pixels of the orthogonal line is to avoid this mistake.

Figure 5.17 shows how the two lines cross without sharing any pixel. There are cases that it does not happen, as it does not when the lines are horizontal or vertical. But in the most cases beforehand is unknown. This is easily avoided by making one of two lines a thicker line. The solution is obvious when the options are: make thicker all candidates or make thicker the orthogonal line. Thus, searching in the neighboring pixels is a way to make thicker the orthogonal line. This is the first problem caused by the line 1 pixel wide.

The advantage of using multiple lines is a possible increased detection of small lines parallel to the main and greater knowledge about the relative position between two long lines. As well as is there a better verification that the candidate is parallel, if it meets

the requirements. The main disadvantage is the increased number of calculations that must be performed. To make this line, the program uses again one of the functions which is used to make the search region in the previous stage.

5.3.2 Comparison for parallelism study

The comparison is the means for deciding whether two lines are parallel or not. As mentioned above, the comparison of the variation in the slopes of the lines is not sufficient to consider these parallel lines. It makes two comparisons: the first is between the vectors of slopes and the second is the slope of the candidates at the intersection with the orthogonal line that has been used for searching.

The program checks that the candidates are also orthogonal to this line, with the scalar product. If the result is close to 0, the lines meet the first requirement and they are candidates to compare their direction vectors with those of the main line. That is a first step in the comparison.

After the first comparison it performs another one more thoroughly than has a difficulty: the lines in the comparison are not usually of the same length and, therefore, the remaining vectors compared are not. It compares the shorter line all along the longer one and it takes the best result. This comparison, since the elements are two vectors, is performed with the Euclidean distance between them (expression 5.3).

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} \tag{5.3}$$

This is done with the vectors of X and Y direction of each line. The coordinates are compared separately. It cannot compare the slope because it could be a case of infinite to compare with another number, and the computer does not process well these calculations. The number of items compared is the number of positions that has the shortest vector. This distance is stored in a vector for each component and it searches the lowest value of each vector. It compares the best performance of both components and chooses the highest value. That is, it chooses the worst case, because if it is valid, the value of the other component also will be. Finally, the program checks that the value is lower than a threshold. The threshold reflects the relaxation of the requirements mentioned in the beginning of the third step. The ideal value would be 0, but in curved lines is very unlikely to happen.

For this reason, if the value is close to zero and it is lower than the threshold, this is enough to meet the first requirement. It discards those that its best value, lowest, it is

not lower than the threshold.

6 Results

6.1 Numerical and graphical results

The previous description of the program has explained three different steps. This chapter discusses the results. But there are results. It shows the intermediate results of each phase as well. The results are mostly graphics. And it will show several images of the intermediate results of the program.

The results of each phase are shown in two images. The first image is composed of black background and white lines. This is the image which the program works with. It's the real result obtained. The second image shows the white lines in red color with the original image as a background. This image shows the larger context of the results, whether the line belongs to a field, a slope, a road or another element.

Some images have allowed the program to complete the numerical evaluation of each phase of the program. These are the images which contain well-defined lines. These lines have been counted previously. In this assessment, there are different aspects to consider.

The first of them is to check the percentage of detected lines. A line may be partially or completely detected. As can be seen in the images, most of the lines are detected partially and they consist of several segments. Lines may also be considered with gaps. These lines are which are intended to reconstruction in the second step.

In this first step the data also shows the number of segments that the program has collected. The first number, always very high, is related to the segments before passing the length filter. The second number is related to the segments which do pass the filter. And the third numerical result discussed about the first step is the percentage of the reduction of lines after the length filter. As the results show the reduction is really high. There are some lines whose detection is a small segment. This segment may be erased and the line stays as non-detected line. This is negative for the final result. When the number of detected lines is low, the program has less data to find parallel lines.

In the second step is also studied the reduction of the number of lines. This value is related to the number of lines have been reconstructed. When the program links two lines the number of lines is reduced by one. In this second step there is no significant reduction of the number of lines. The results of the satellite images are not many connections. Most connections are between segments with no gaps but the program had not detected as a single line. On the other hand, in synthetic images are more connections between segments.

The results of the third step are basically graphical results. The lines that appear are those that belong to a group of parallel lines that exceed a certain number set as a threshold. Thus, one can identify a crop field. When the program detects large groups of parallel lines are detected means that it is analyzing the plowing of the field. In order to measure this third step is shown in table 6.3 the number of lines that belong to a group of parallel lines. If the percentage is close to 50% or higher, the image may contain an agricultural land. The importance of achieving long lines makes sense in this step. A long line may be parallel to several shorter segments. But the program does not need that these segments are parallel to each other.

This happens in curved lines. The program detects two parallel curved lines. One of them is detected as a long line. But the other is detected in several segments. These segments, in a curved line, are not parallel between them. But all of them are parallel to the long one. This is a problem in this third step. The number of parallel lines may be higher than it actually is. But this way also allows the program to detect several curved lines parallel to a long one.

One problem that exists is the detection of two parallel curved lines. But depending on which pieces of each curve are detected, the program may not consider these two lines as a parallel curves. Thus, the problem before it becomes an advantage because the long line would be parallel to each of the segments of the other parallel lines.

The figure 6.1 is the best example of an agricultural field. It cooperates very well with the program. The plough forms clean lines and it is a high contrast image. This image is taken as ground truth to evaluate how the program works.

The first step of the program, as the figure 6.2 shows, is to collect the lines. The image has 188 valid lines of plough, which helps to decide if the image belongs to a crop field or not. The program has to detect, reconstruct and study the parallelism of these lines.

The figure 6.3 shows, in red, the lines collected by the program in the first step. 100% of the lines have been detected. That is, each line has been detected at least by one

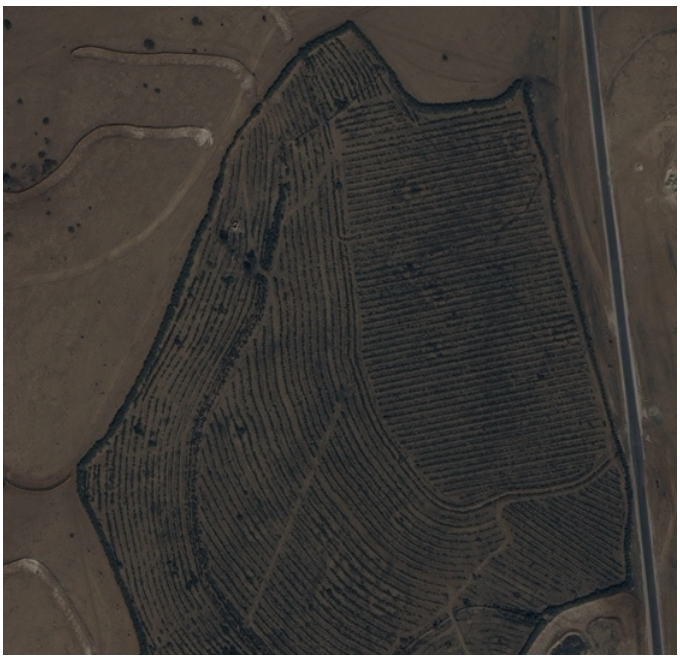


Figure 6.1: Crop field in Africa. This is the sharpest image of a crop field.

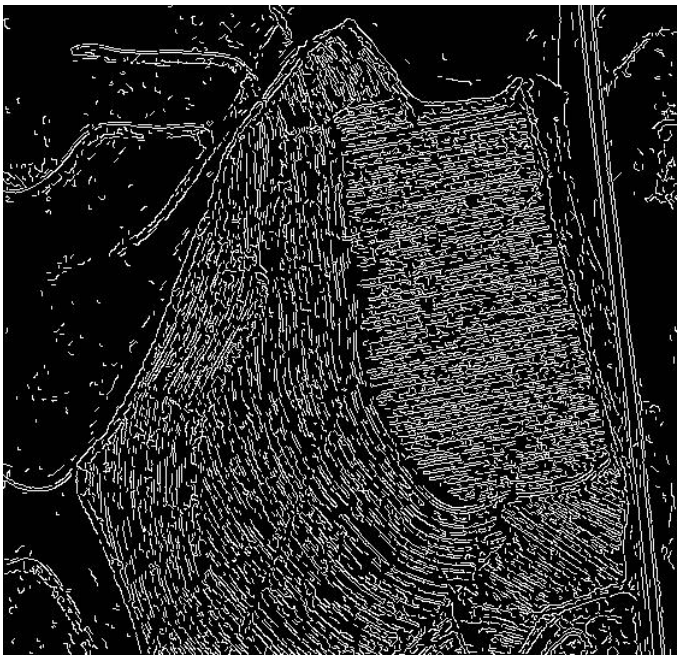


Figure 6.2: Image after the first step of the program. It has collected a lot of segments.

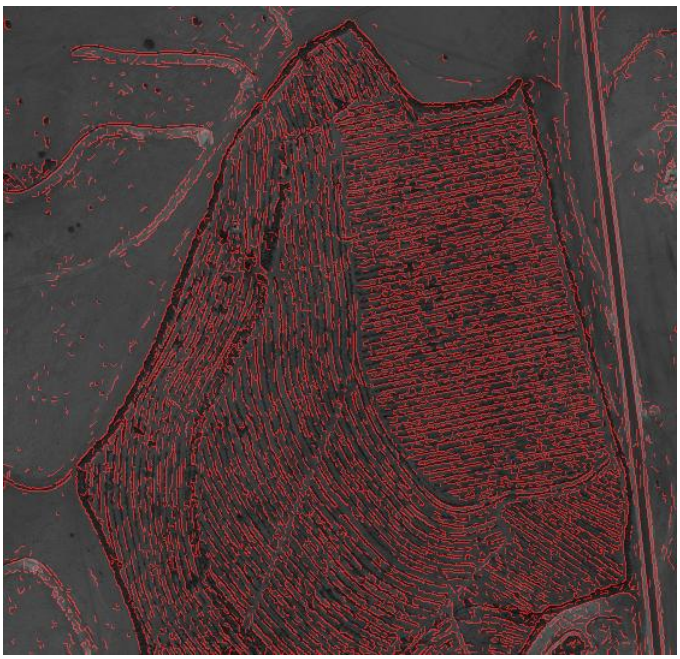


Figure 6.3: The red lines are the lines collected by the program. This image helps to know the context of these lines.

segment. All lines are composed of segments. None is detected in its entirety in the first step. In black and white image can be seen a large number of small segments.

The red lines are already classified and their coordinates are stored. It is also noted a Canny filter effect. Sometimes it represents the edge twice. The reason is that in a thin line is detected two changes of intensity. They are more parallel lines that make up for the not detected lines.

The largest segments on the black and white image (figure 6.4) belong to the perimeter of crop fields and roadsides. The segments formed by slopes are also detected clearly. The number of variables is very high, since for each segment there is an object variable.

That is why the next step is choosing the best lines that allow the study of parallelism. These segments are the longest, which provide more data. The reduction of the number of lines is very high, about 90% in each tested image. In the case the figure 6.4 is 90.17%. That is, the program collects 5933 segments and it only keeps 583 for the following steps.

This reduction of variables makes easier the following processes. The program removes the segments not long enough which could act as noise. This image is very sharp and

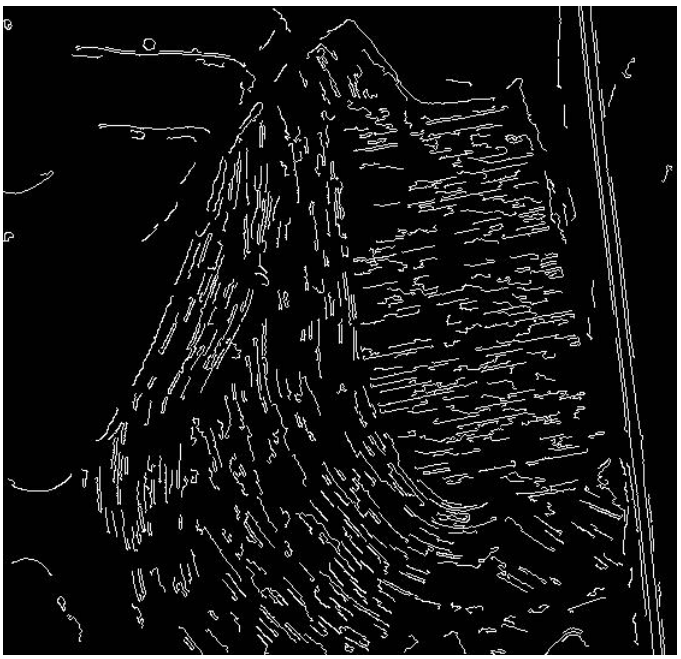


Figure 6.4: The program removes the shorter segments because they are less reliable data.

most of the segments belong to the plough lines. But not always is that way. They can be part of trees (figure 6.19) or even of a building. The program should discard the segments too short because they do not allow the second step of the program to work on an efficient way. This is because these segments contain data not reliable enough about their direction. The program should discard the shorter segments because they do not allow the second step of the program to work on an efficient way. That is the reason why segments do not contain enough reliable data about their orientation.

The percentage of deleted lines is similar in all tested images. The difference is the length, in average, of the lines which the program keeps. If the image cooperates, the segments are longer. This is because the removal is done by averaging pixels per line. You can not set a single numerical threshold for all images. The reason is that in low contrast or dark images, the program collects shorter segments. With a pre-established numerical threshold the program would have many problems in these images. This would work with very few lines, making it hard to detect groups of parallel lines.

In the figure 6.5 begins the second step of the program: the reconstruction of the lines. It fills the gaps between segments to create longer lines. These lines provide more reliable data. This data is, mainly, about the curvature and orientation of the lines. At this step, the reduction of the number of lines is not very significant (about 12%). Al-

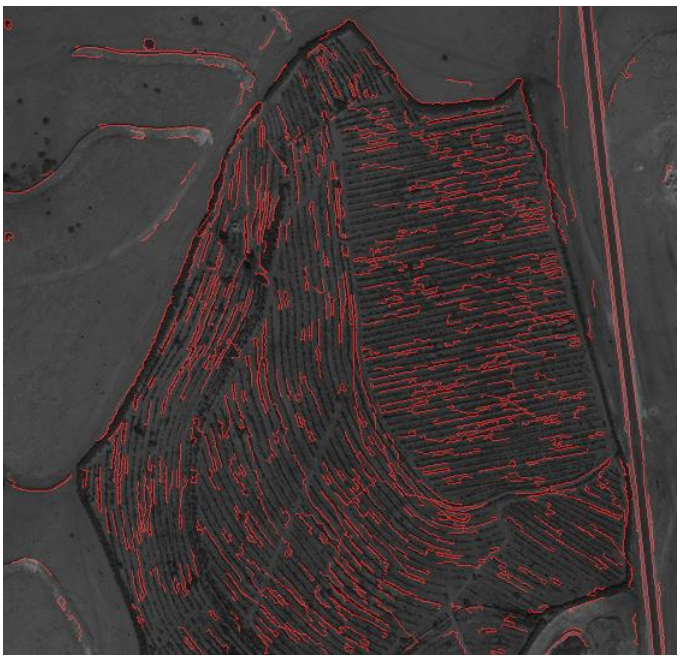


Figure 6.5: The lines in red are the best lines that the program can ontain after the second step.

though like mentioned above, this is not the main goal. The program works best in this phase with the images that contain more defined lines, or mostly straight lines. If the image contains green areas that are detected with small curved segments (as in the image 6.19), the parameter is smaller. That is because this green area has not a long lines structure. This small percentage shows how difficult the reconstruction of curved lines is.

These results give prominence to the first step. The reconstruction of lines is done in a low percentage. And this means that the third step, the parallel study, is carried out mostly on the lines as they were collected at the beginning. This phase is responsible for calculating the vector that the program uses on the comparison of the next step. This process is not run before to avoid wasting resources by extracting useless data.

In small images with straight lines this step works better (as shown in figure 6.24) than in satellite images.

These are the best lines the program can get during the two previous steps, the lines which the program compares in order to find groups of parallel lines.

This image is the result of the last step of the program. The lines shown in the figure

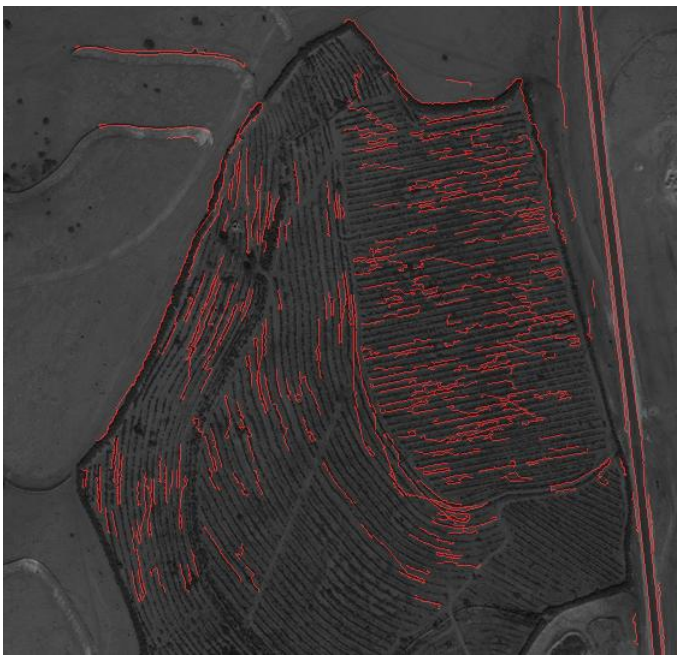


Figure 6.6: The red lines are the lines which belong to a group of parallel lines. That is the final result.

6.6 are those that belong to a group of parallel lines. The other lines are not shown in this image. The image shows a high number of lines. It means that in the original image there are many parallel lines. If the original image is in a green area, the most likely conclusion would be that it is a crop field.

This is the image that works best with the program for the clarity of its lines. The program has been able to collect many long lines of the first step. This good collection has allowed the last step to have many lines to compare and study parallelism.

In other images where the crop field is not as sharp as in this image, parallel lines are also detected. But there are many fewer lines. So the conclusion is not so clear.

6.2 Co-operation of the images

The tables 6.1, 6.2 and 6.3 show how the program works on each image. The results have been obtained without changing the program parameters. The different columns and their meaning for the first 2 steps of the program have already been discussed. The last column of the table 6.2 shows the average length (number of pixels) of the lines before

Figure	PL	SC	PL(%)	VS	NLR(%)	PL(%)
6.1	188	5933	100	583	90.17	85.64
6.14	33	6048	100	627	89.63	96.96
6.19	7	2434	100	264	89.15	100
6.30	16	8110	100	774	90.45	100
6.7		2102		197	90.62	
6.33		6144		705	88.52	

Table 6.1: This table shows how the program works in the first step. PL= lines in the picture, SC= segments collected, PL= percetage of segments which belongs to lines in the pictures, VS= valid segments after the first step, NLR=percentage of the number of the lines.

Figure	VS	TL	NLR(%)	AL(pixels)
6.1	583	508	12.86	41
6.14	627	577	7.97	28
6.19	264	250	5.30	27
6.30	774	734	5.16	26
6.7	197	174	11.67	42
6.33	705	682	3.26	29

Table 6.2: This table shows how the program works in the second step. VS= valid segments after the first step, NLR=percentage of the number of the lines, TL=number of lines after second step, AL=average length of the lines.

the third step. This average is done after removing the shorter segments, as in the figure 6.19. A higher average means that the program has collected long lines. And this means that the image is working positively with the program. A lower average means that the lines which the program has found are shorter. It may be because the plough of the field is not sharp enough or because the contrast of the image. A low contrast image makes harder that the program detects intensity changes. These are necessary to detect the lines which are processed by the program. So the program works worse on this kind of images.

This image is one the images that has obtained the highest value on the average lenth parameter, last parameter of the table 6.2 . The image is composed of light-colored fields where the straight-shaped plough is dark (figure 6.8). In this case, the image is not as sharp as 6.1, but the crop land contains high-contrast color lines. It is also an image with mostly straight lines and, as explained above, the program collects these lines more easily. Thus, the program has many long lines in the third step, as shown in the figure 6.10. The figures 6.5 and 6.10 have high value on this parameter. The average length of the best images is 50% above the average of the worst images. These images are the ones that have obtained the highest results in the second step. Thereby, the result

Figure 6.7: Picture which obtains the longest segments.

Figure	Final lines	Parallel ines	(%)
6.1	508	296	58.26
6.14	577	100	17.33
6.19	250	30	12
6.30	734	63	8.58
6.7	174	129	74.13
6.33	682	682	3.26

Table 6.3: This table shows how the program works in the third step.

that presents the figure 6.11 is a high number of lines belonging to groups of parallel lines.

The figure 6.1, commented at the beginning of this chapter, is also working positively with the program. The lines left in the third step are a 27% longer than average. The figure 6.7 works better than average. Despite the difficulties of the second step, the

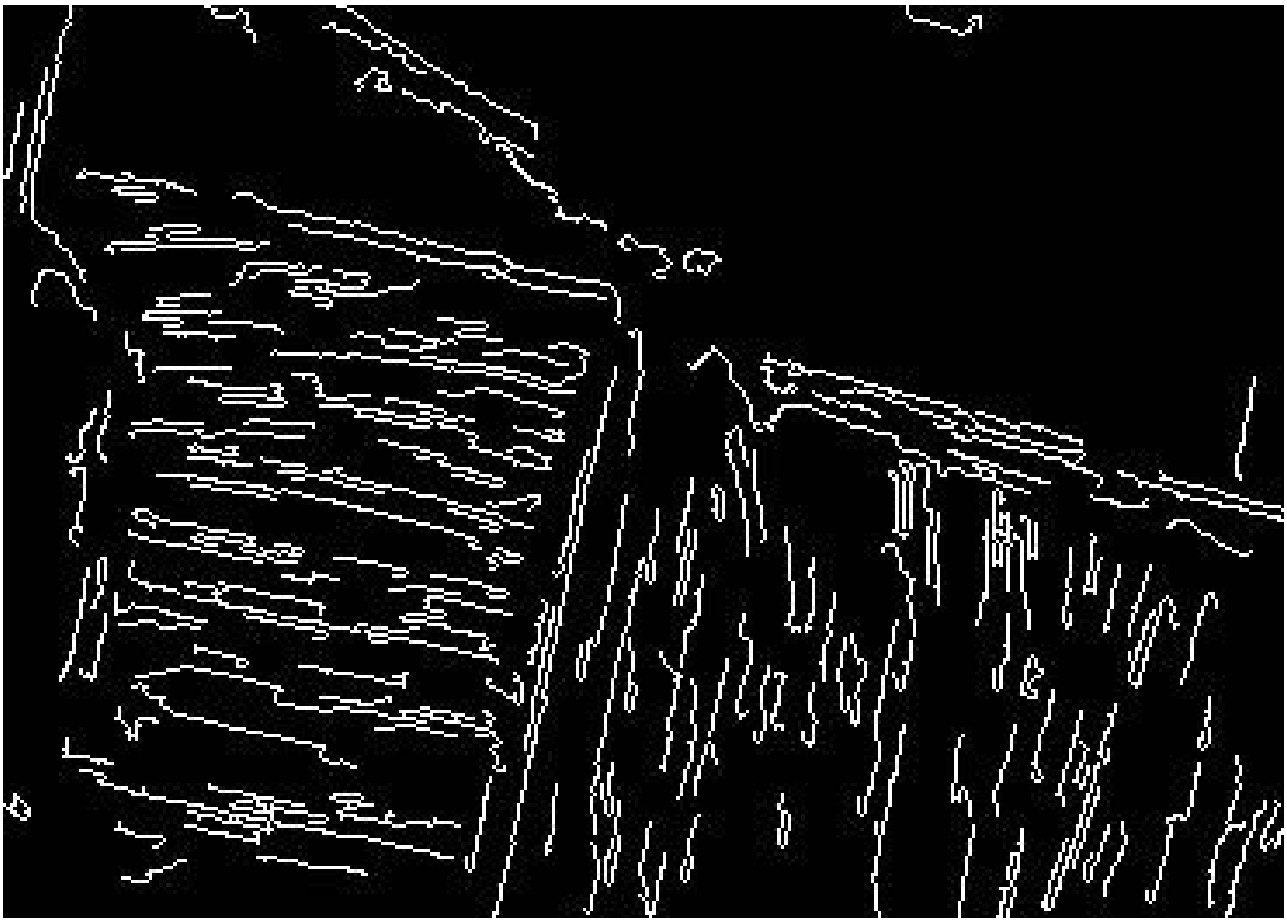


Figure 6.8: The best lines belong to the detection of the edges of crop lands.

final result shows a high number of parallel lines. It means that the crop field has been identified.

Thus, the length of the segments that make up the image measures the cooperation of the images with the program. As mentioned above, it is basic to the main goal of the program: the study of parallelism.

The figure 6.13 shows the behavior of the program when the image contrast (figure 6.12) is not as good as the contrast of the previous images. The field edges are detected very well since the contrast of the image is high enough. But the plough is not sharp in the image. There are some lines, but it is hard to distinguish them. That is why few segments are detected. There are not many long enough segments. The image shows many segments, but most are very short.

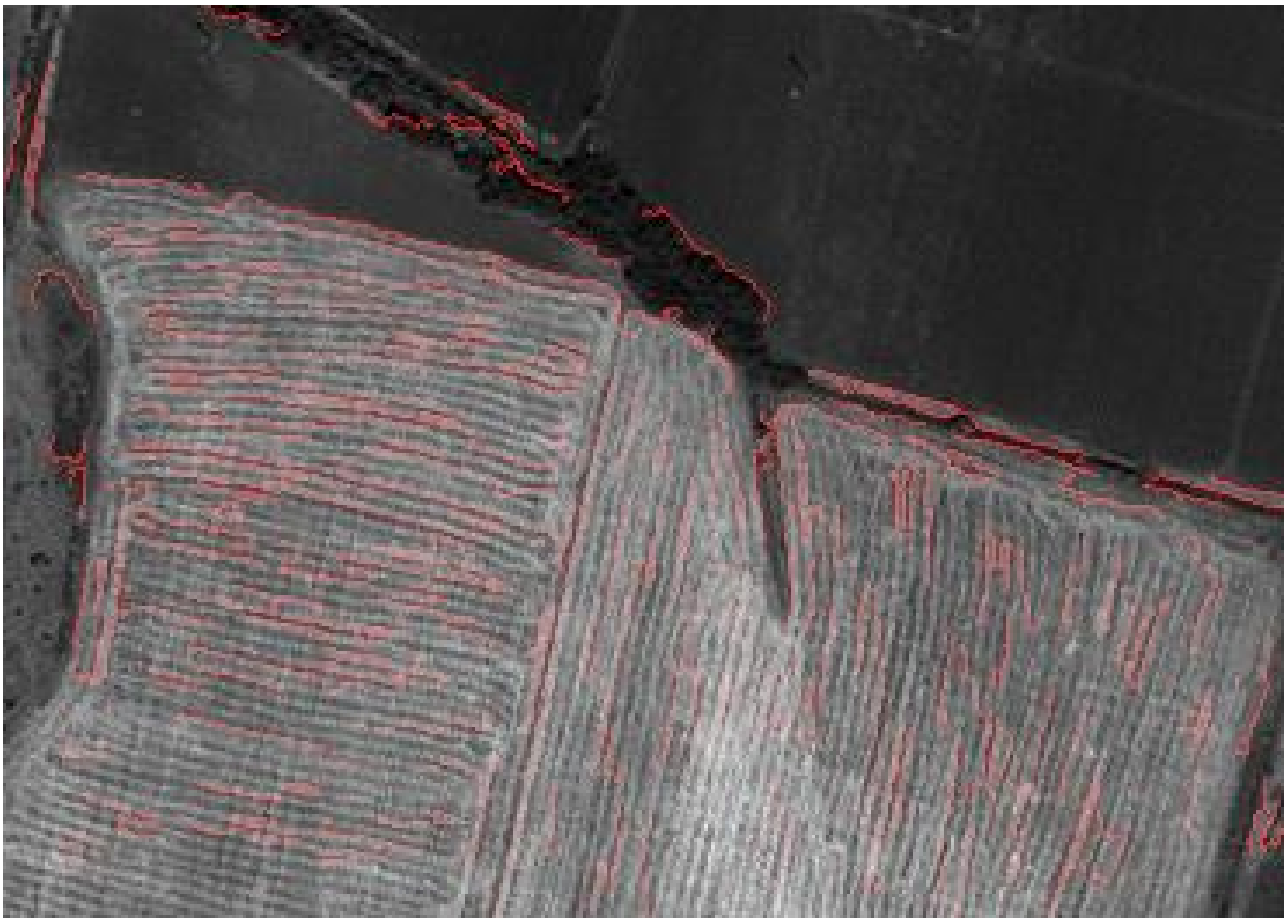


Figure 6.9: The program detects a lot of lines in the crop field and almost no line in the green area on the top of the image.

The image shown on the figure 6.14 tests the program with sharp curves. This image does not represent a crop field and its contrast is low. But the lines, due to slopes, are clear in many areas. These lines are very long. But the program does not detect them completely (figure 6.16). Even so, the program detects long enough segments (figure 6.17) to study their parallelism.

After removing all short segments, there are many long lines in two located regions. This allows the last step detect parallel lines, as shown in figure 6.18.

This image contains well defined parallel lines. The program detects them not as a single line. It detects them as consecutive segments. The figure 6.19 is a good example of the limitations in the tracking phase compared to the figure 6.1. The program detects a road with very long segments (10 times longer than average). The first step, like the

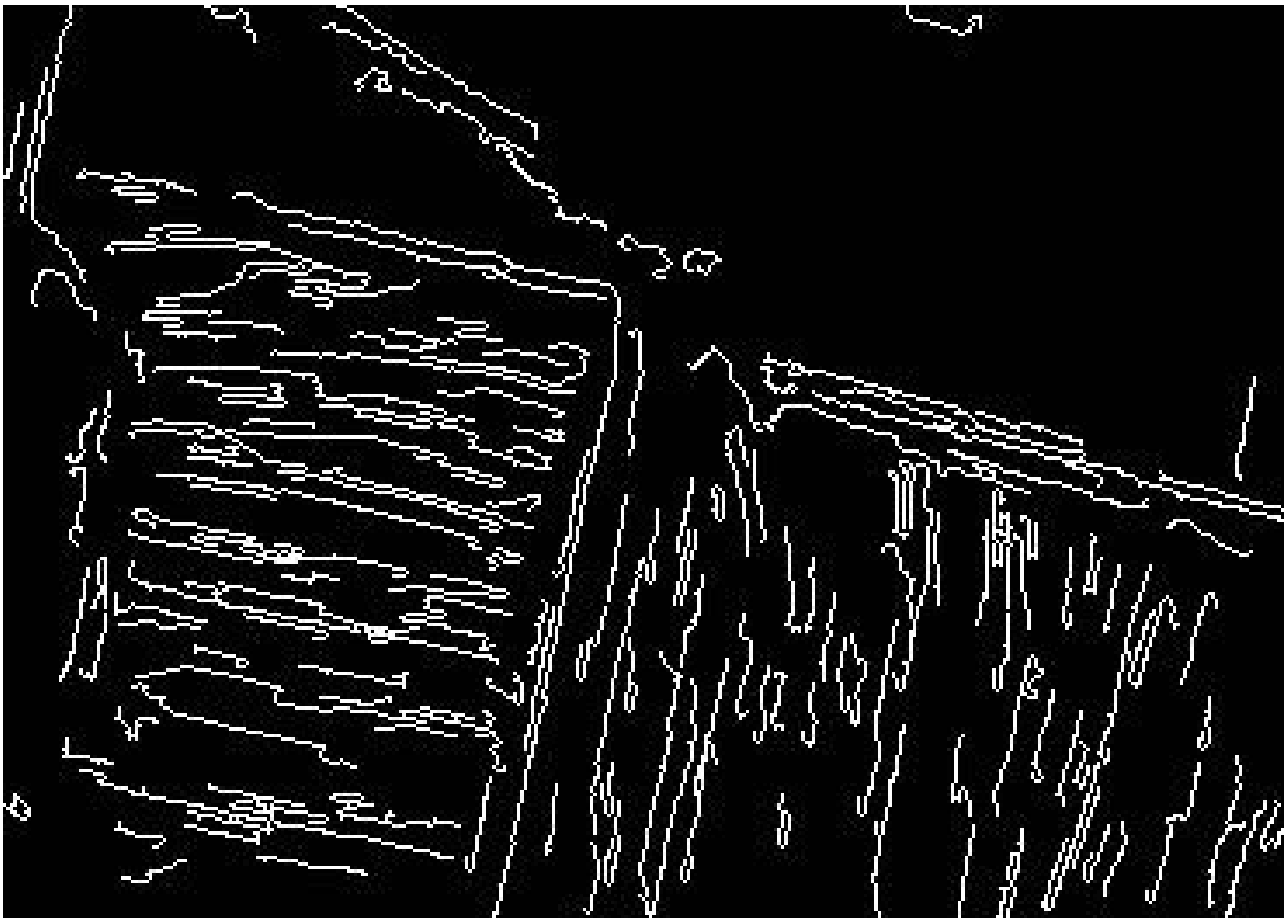


Figure 6.10: Image before the third step. There are many straight segments. That is why there are many long lines.

second one, works better with straight lines.

In this picture the lines do not form crop fields. These lines are much farther apart in space than in the first image, so they are not detected as a group of parallel lines. The distance that separates the lines should be approximately like the first image. Otherwise, they could be slopes or other causes different from an agricultural land.

From the data in this figure, it is apparent the importance of reducing the number of segments. In the center of the picture can be seen an area with green tones. This could be the image of some type of vegetation. But the provision is not arranged in rows, like a plowed field. The figure 6.20 shows that the program detects it as an area of short segments arranged chaotically. If these segments remain, the reconstruction of the lines would be harder. The significant reduction of these segments can be seen in the

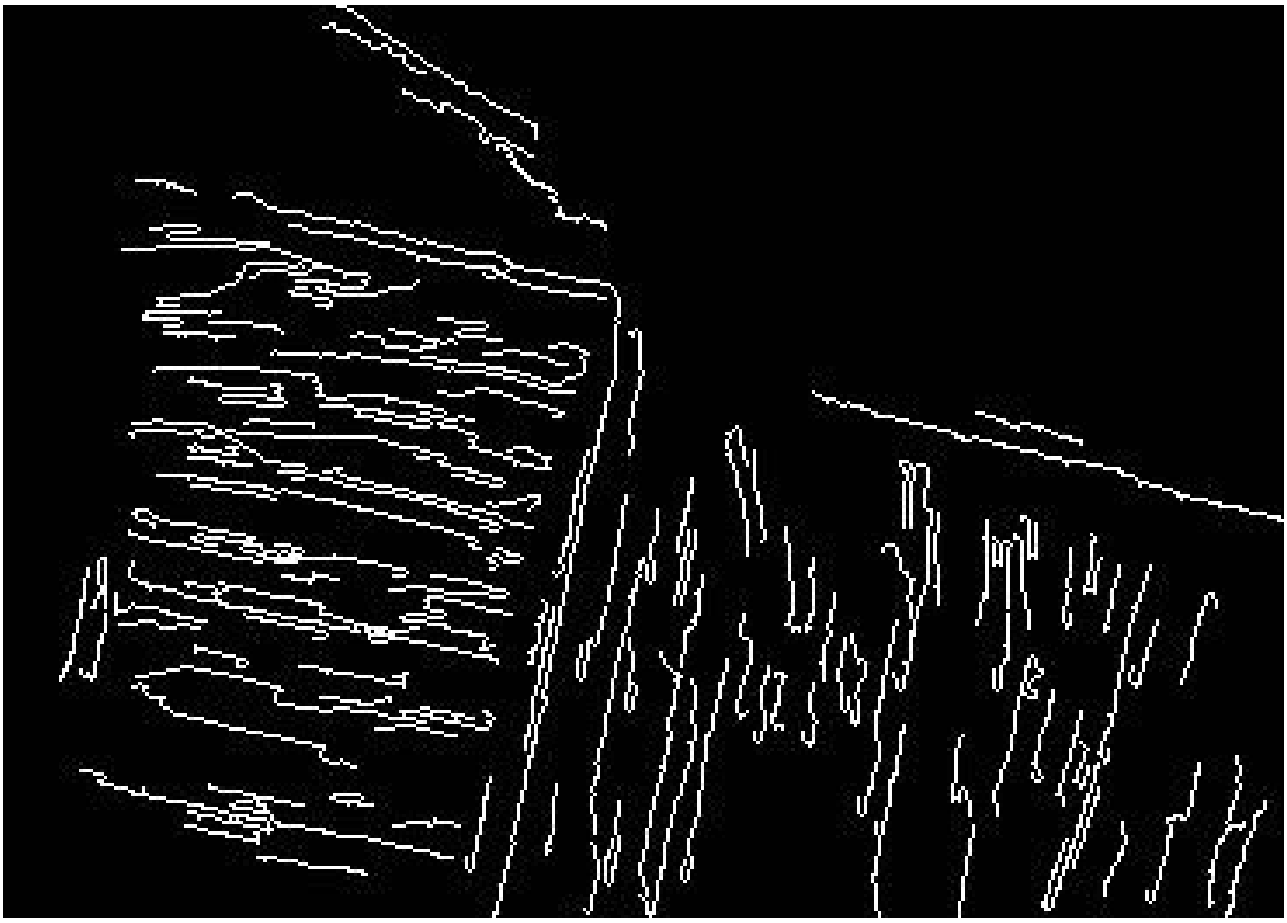


Figure 6.11: The program can detect many parallel lines. It works better with straight lines.

figure 6.21. The area after the first step is clearer. Thus, The program requires fewer operations and works better.

6.3 Synthetic images

The program has been developed by testing on this type of images. Figures 6.23 and 6.22 are composed of short straight segments and these images are appropriate to check the performance of the reconstruction of lines. In the synthetic images, which the program do not use the Canny filter, there are better results for the second step than in cases of real images. In Figures 6.24 and 6.25 shows better performance because the segments are straight. In real images are seen many segments with curved ends. A small curve at the end of the segment can change the direction of the search region. If the region is

Figure 6.12: This is a low contrast image. The lines of the plough are not sharp.

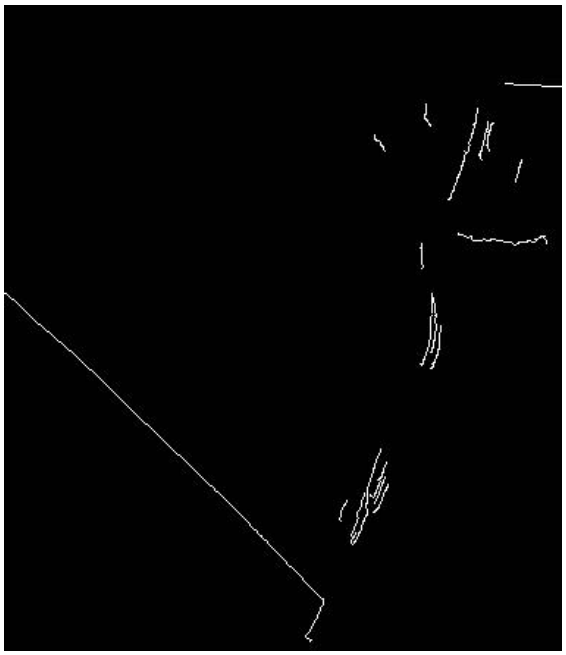


Figure 6.13: The program can detect few parallel lines. These are detected on the edge of the field.

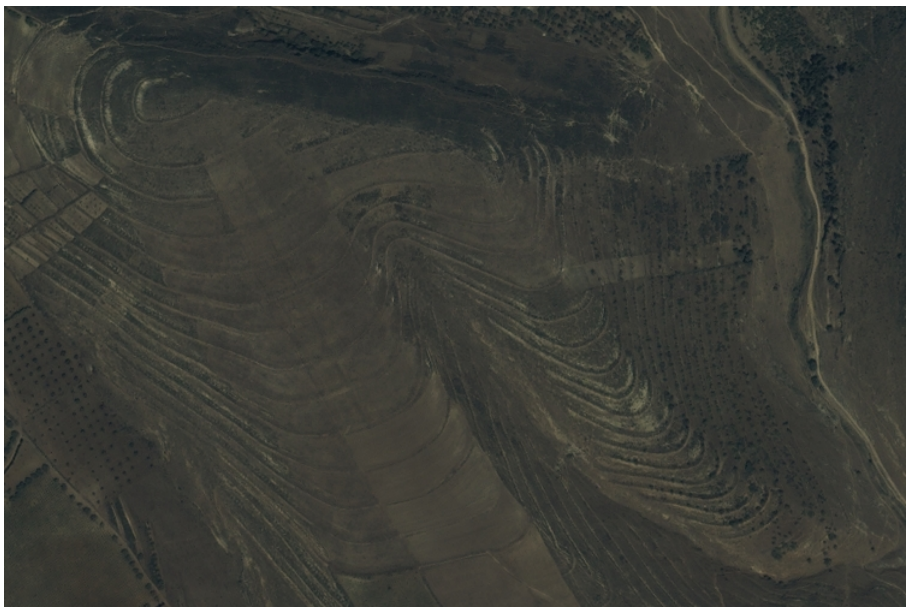


Figure 6.14: Low-contrast image with sharp curved lines.

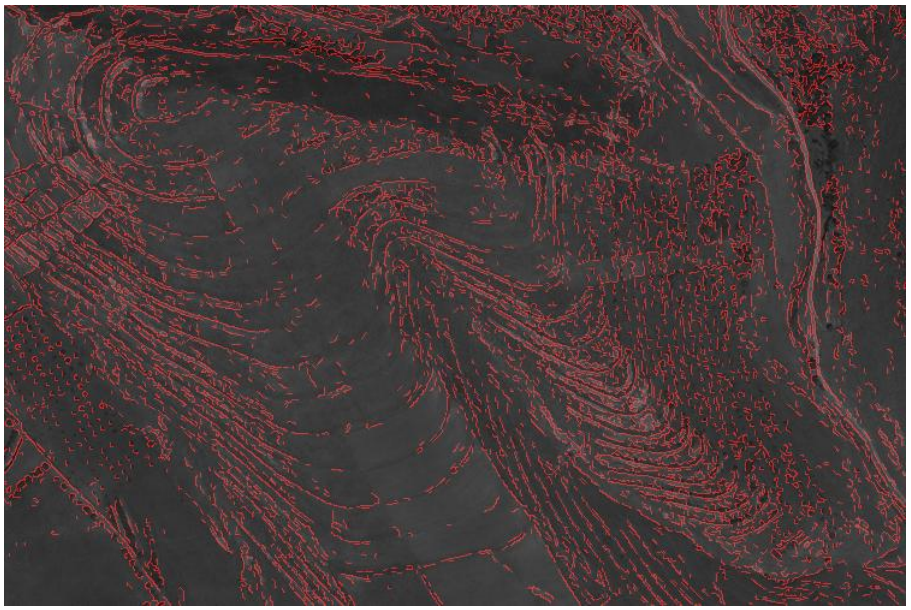


Figure 6.15: The image is full of short and long segments. But there is an area with few segments. The lines have been detected not well in this area.

facing the wrong place the program connects lines that should not be connected.

The method of candidate assessment in the reconstruction of the lines, explained in Chapter 3, is suitable for images with parallel lines. The figure 6.3 is a clear example

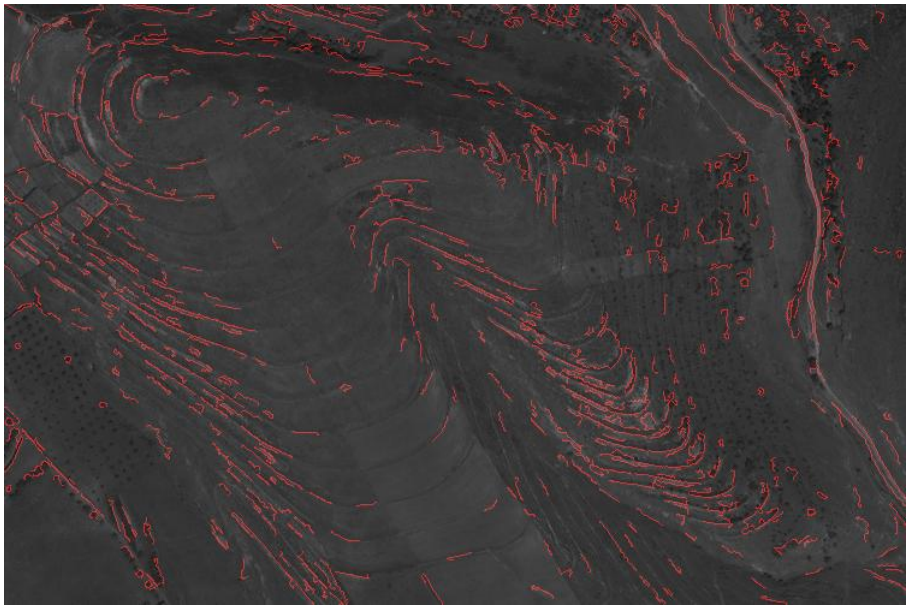


Figure 6.16: The program has removed most of the short segements.But there is an area with few segments. The lines have been detected not well in this area. Some chaotic curve-shaped lines remains in the image. These belongs to some trees in the picture. These lines are not short enough to be removed.

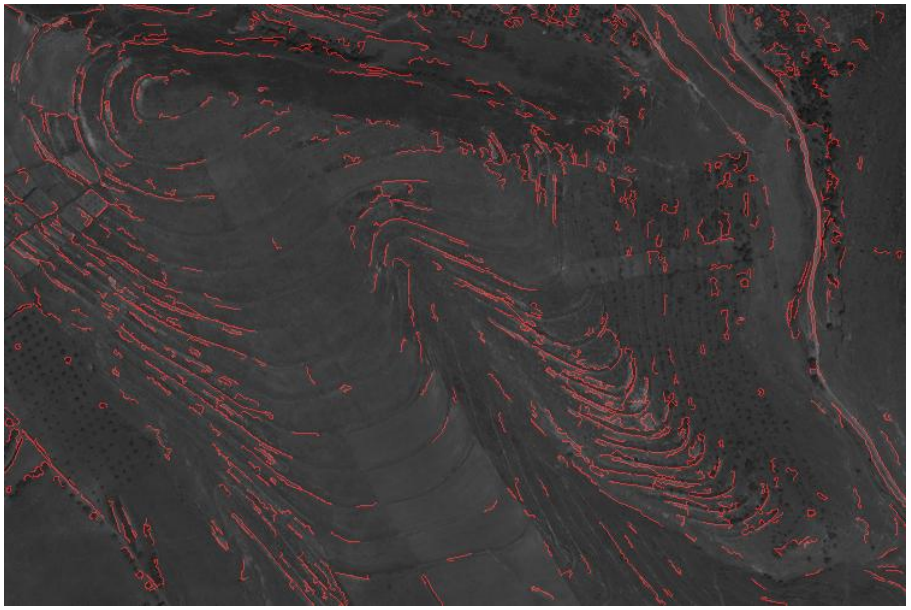


Figure 6.17: The program can not rebuild the area in which there are few segments. The long lines are divided into two localized areas.

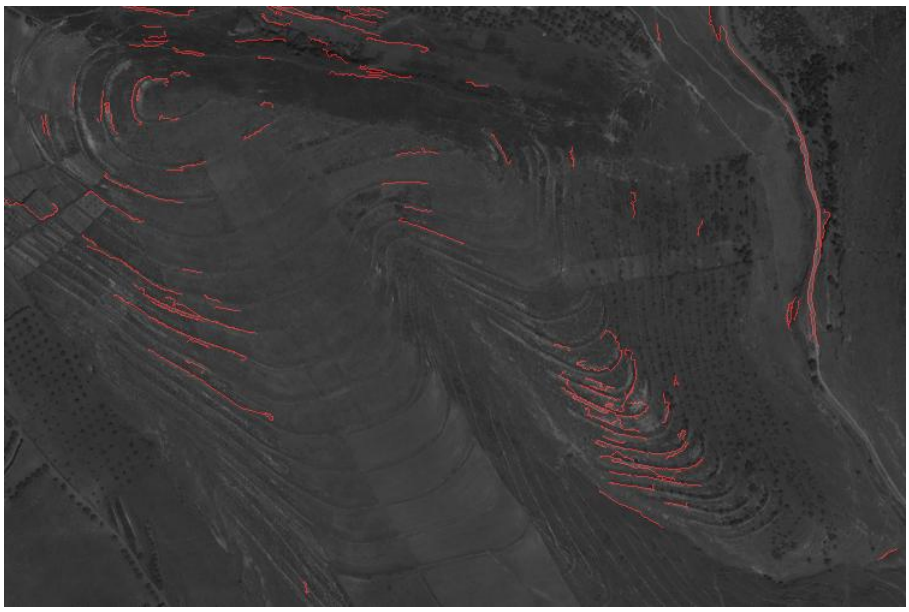


Figure 6.18: In these two areas there are groups of parallel lines.

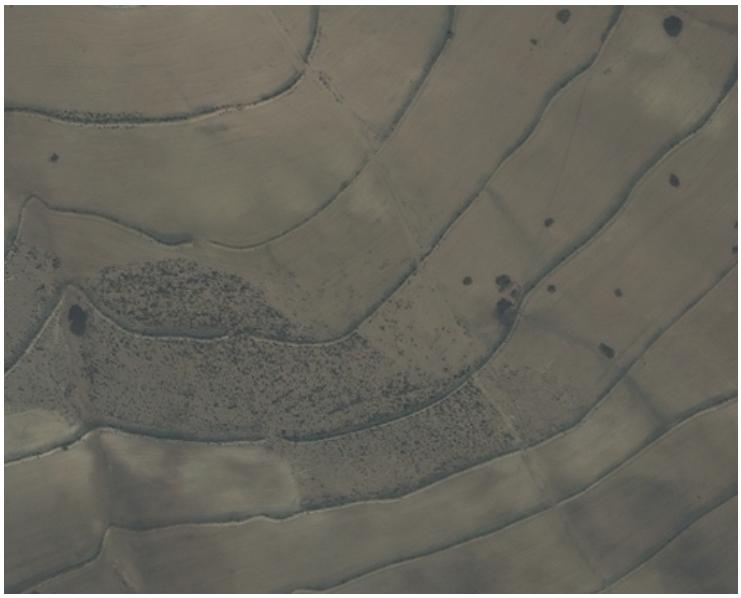


Figure 6.19: This image shows a green area in the center that causes problems to the program.

that the comparison must not be between the directions of the segments. In that case, the program would have chosen the first line it had found. But instead, the method chooses the segment lined up with the line on the left (figure 6.27.

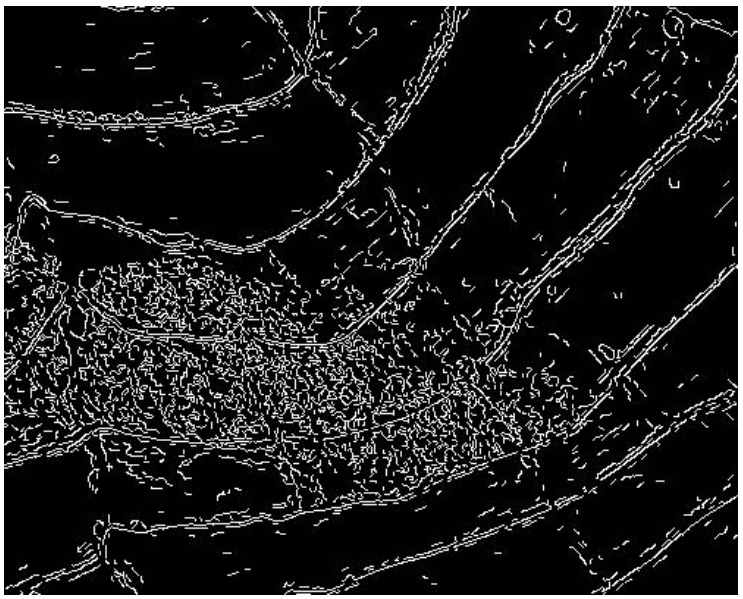


Figure 6.20: This image shows a green area in the center that causes problems to the program. It fills a region of the image with many short segments.

Although this comparison may help in the case of Figure 6.28. With the method, the program connects the line on the left with the first segment detected. This is because

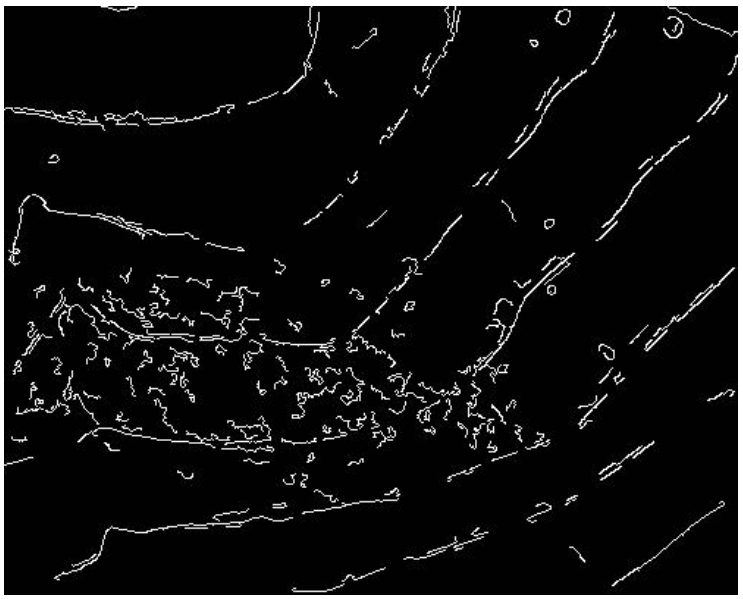


Figure 6.21: The program can remove the most of the short segments in the center of the image. This makes the work easier.

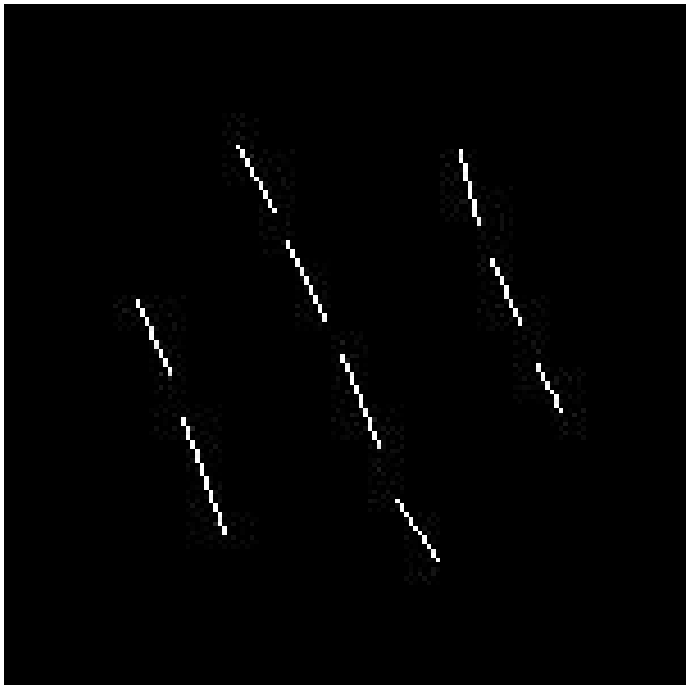


Figure 6.22: Short straight segments simulating long lines with gaps.

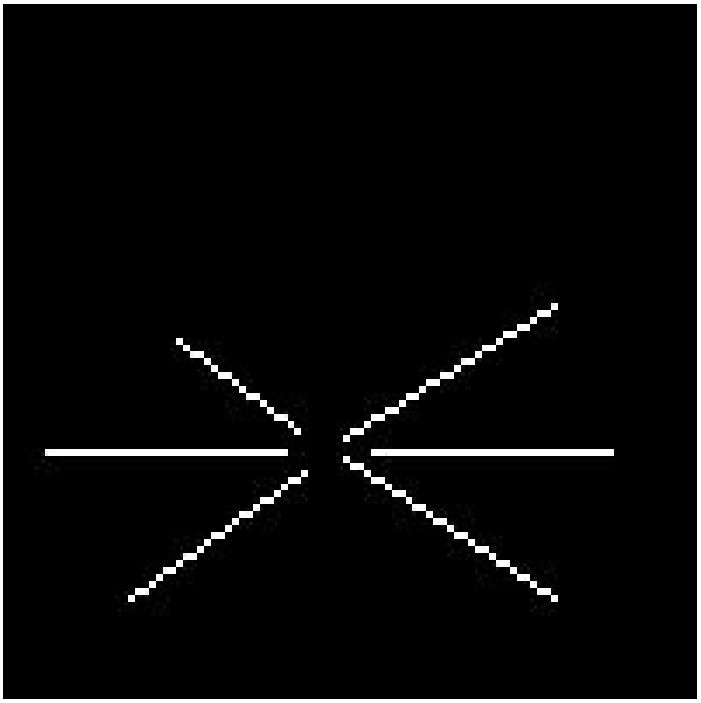


Figure 6.23: This image shows a gap in an intersection. The search region finds several candidates and the program has to choose the right one.

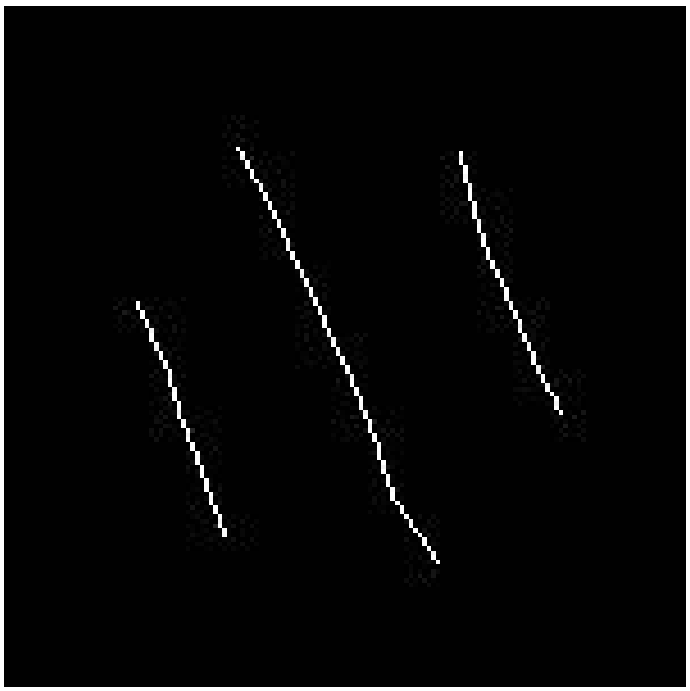


Figure 6.24: The program connects the short segments each other and the outcome is an image composed of 3 long lines.

all segments are lined up with the right endpoint of the line on the left. Thus, all comparisons have the same result. And if all are equal and acceptable results, the program chooses the first candidate 6.29. However, this cannot be considered an error since the program cannot know which way to go if all are equally valid.

6.3.1 Other images tested for results

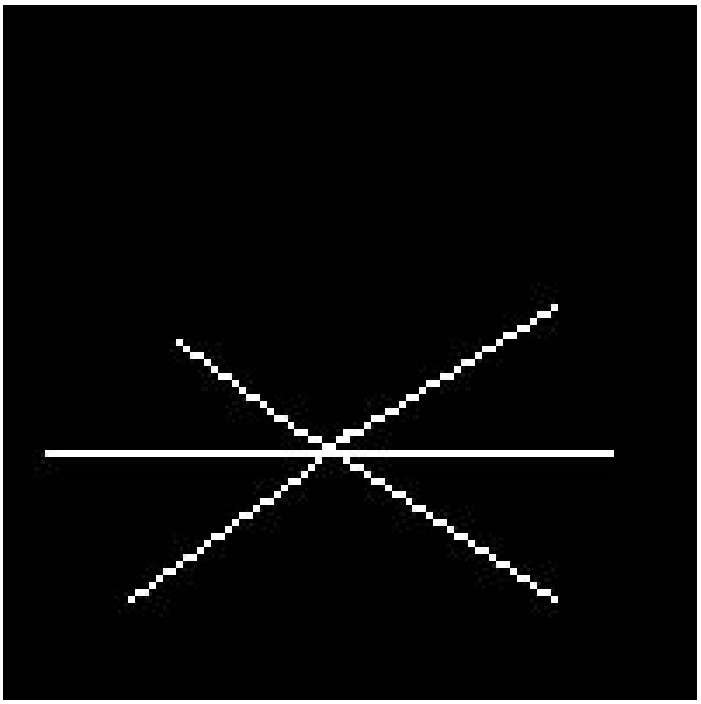


Figure 6.25: The assesment of candidates in the second step, explained in chapter 3, works properly.

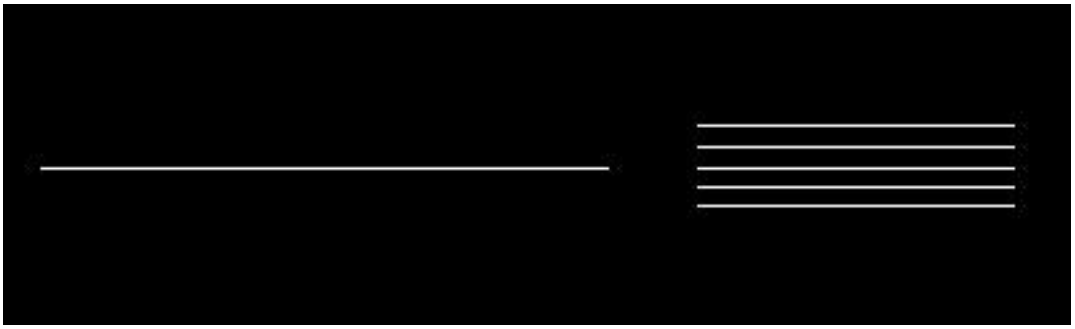


Figure 6.26: Only two segments are lined up. The program has to choose the correct segment among the segments on the right side.

Figure 6.27: The selection is right. The two segments are perfectly lined up and this makes the selection easier.

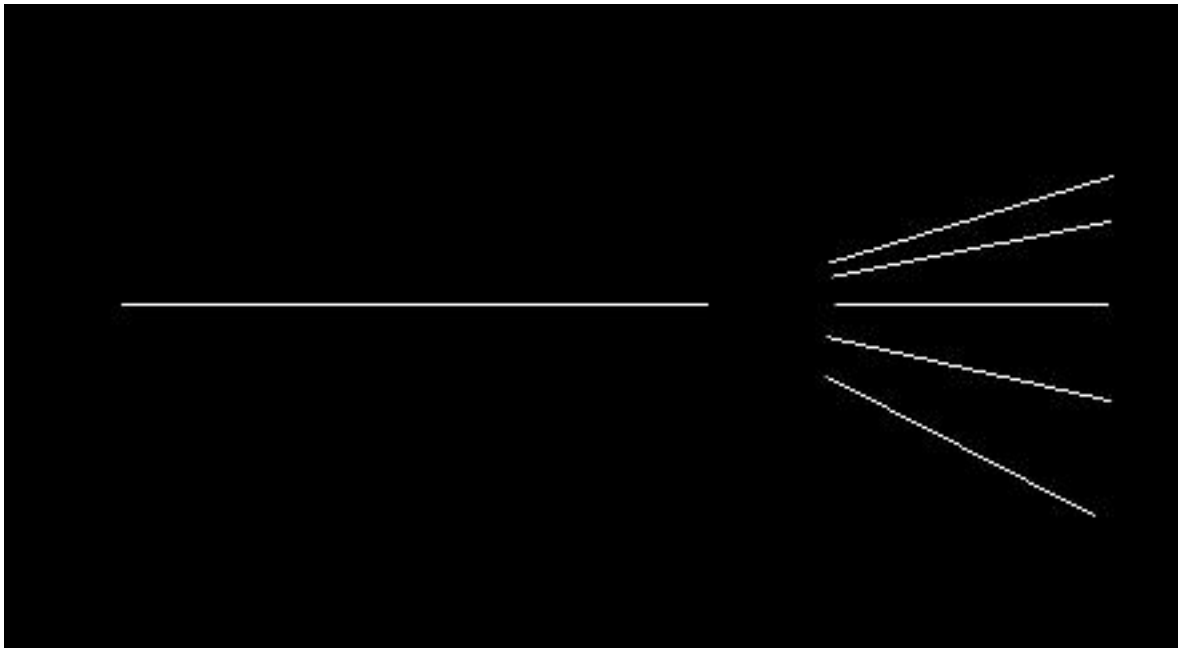


Figure 6.28: All candidates on the right point to the right endpoint of the line on the left.

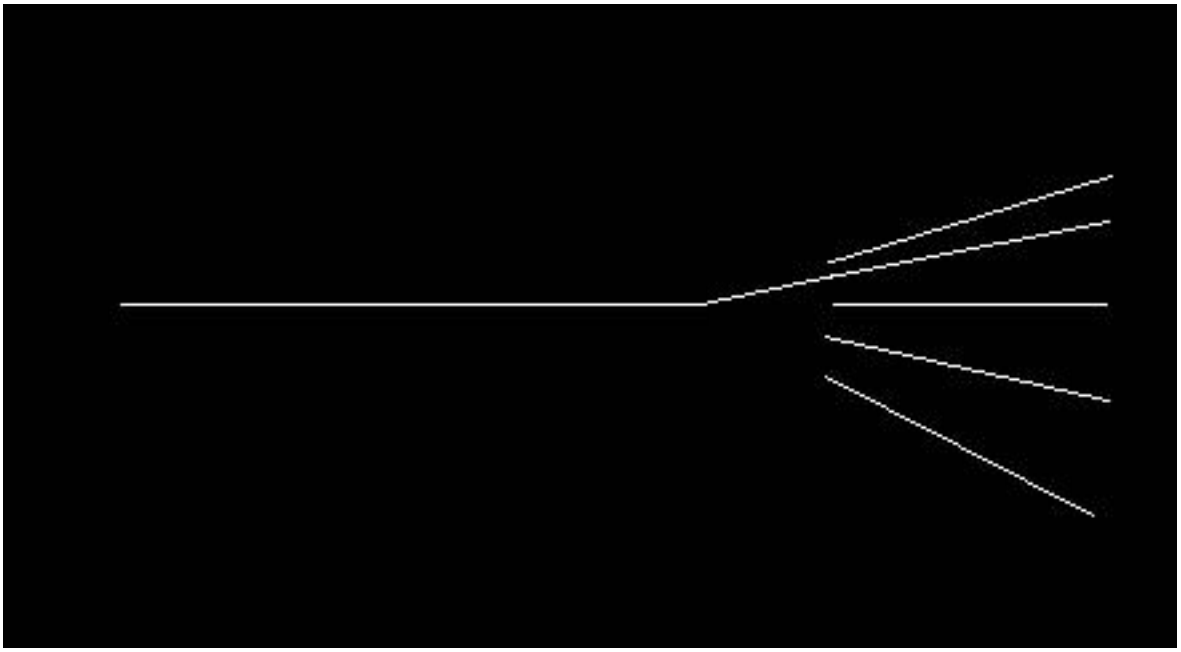


Figure 6.29: The assesment of candidates cannot work because all candidates are lined up. The program cannot choose the best one because all comparisons are equal each other. In this case, the program chooses the first candidate.

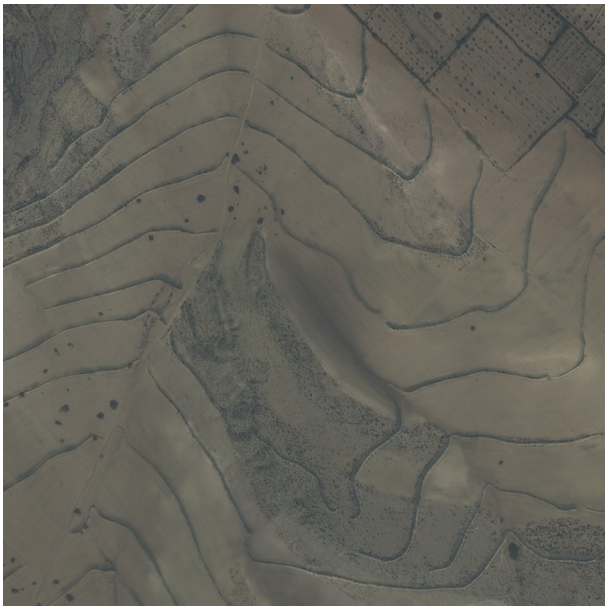


Figure 6.30: This image is similar to the figure 6.19, curved lines distant from each other and a green area in the middle. There are some plantations on the top right corner. And their edges are well detected too.

Figure 6.31: The green area is full of short segments. The long lines are sharp, so they are well detected.

Figure 6.32: The first step removes many short segments of the green area.

Figure 6.33: This image is only for test the program, totally different than the others.

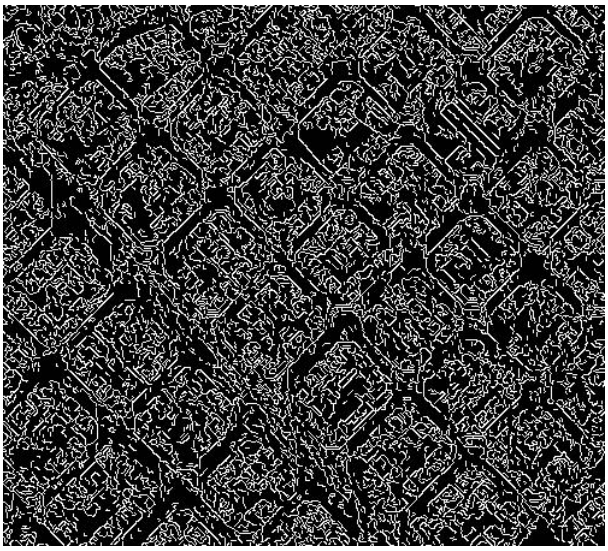


Figure 6.34: There are no long lines. The image is composed of short segments.

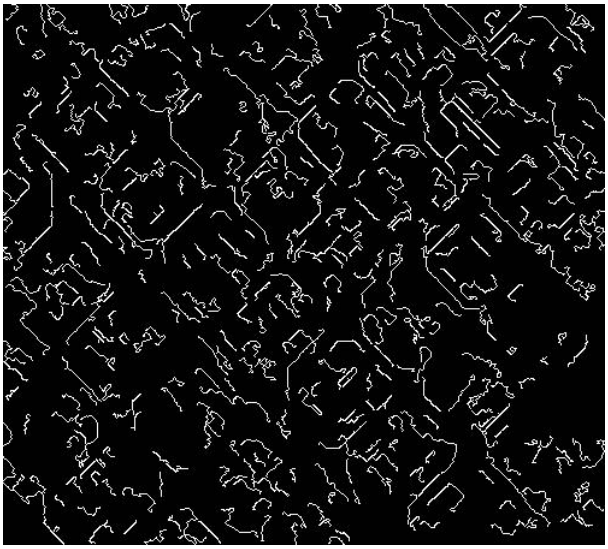


Figure 6.35: The reduction, despite the absence of very long lines, is significant.

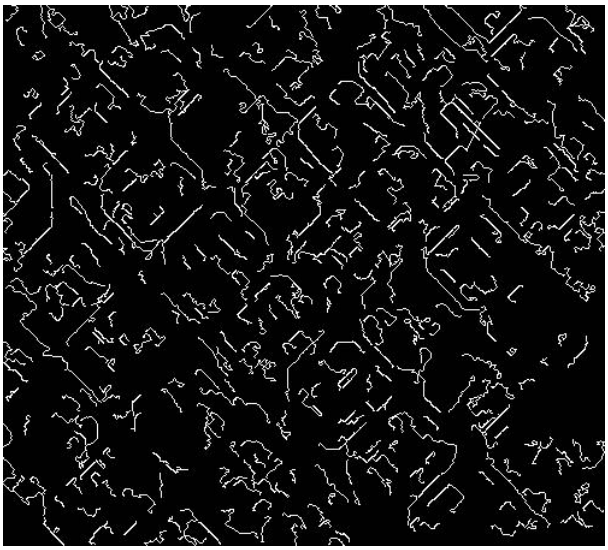


Figure 6.36: The longest lines are the edges of the blocks. But these are not as long as the lines of the crop fields.

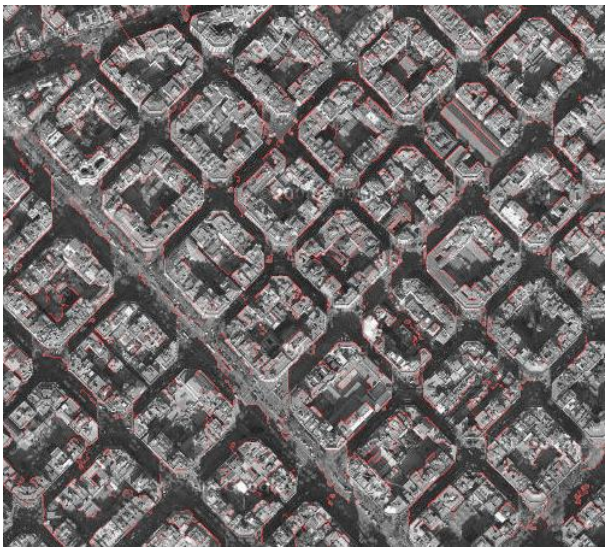


Figure 6.37: The red lines on the original image help to understand the context of the lines of the figure 6.36.

7 Summary

In order to identify fields in satellite photos, a detector structure of parallel lines has been developed. The real goal is the reconstruction of curved lines with gaps and the study of parallel curved lines.

The method consists of 3 steps: the first step is to detect the edges and store them as lines. This part of the algorithm chooses the best (longest) lines to carry out fewer operations and remove the shorter segments, that could act as noise. The second step is the connection of those segments that may belong to a single line. In this step the method considers two different factors: the slope of the endpoints of the line and the relative position between the segments that may be connected each other. The third and final step is to study the parallelism between the lines. The program analyses every line throughout its entire length, and the lines' position relative to each other.

Every step of the program begins from the results of the previous step, making error propagation an important concern, meaning the outcomes of the first steps are as important as the final result. This is why makes good sense to talk about intermediate results. As a result of the step of collection of segments, the program keeps 10% of the total collection of segments in the image. This makes the program work with longer lines which provide more reliable data. The results of the second stage, as seen in the chapter of results, depend on the quality and type of image processed by the program. If the crop land contains a sharp plough, the program can connect 14% of the lines of the image. Also similar results are obtained in straight-shaped ploughs. In low-contrast images, the number of connected lines may reach 7%. That is 50% less connections. For the study of parallel structures the program searches groups of several parallel lines. There is a parameter that helps to decide whether the image is a crop field. This parameter is the percentage of lines in the image belonging to groups of parallel lines. In the tested images, a percentage close to 50% or higher means that the image shows a crop land. This parameter depends on the size of the crop field relative to the size of the image.

For further work, increase the percentage of connected lines would help the study of parallel lines for several reasons: the program would work with more reliable data and would have a better chance to identify parallel lines. Another possible improvement would be studying different connections between segments, other than straight lines. A connection by interpolation or extrapolation of the segments could improve the study of parallelism.

Bibliography

[1] J. Buckner, M. Pahl, O. Stahlhut, C. E. Liedtke: geoAIDA, A Knowledge Based Automatic Image Data Analyser for Remote Sensing Data.
CIMA 2001, Bangor, Wales, UK, June 19th-22nd 2001,, (2001).
ftp://ftp.tnt.uni-hannover.de/pub/papers/2001/CIMA2001-OS.pdf

[2] C.-E. Liedtke, J. Bckner, M. Pahl, O. Stahlhut: Knowledge based system for the interpretation of complex scenes.
Ascona 2001 / Schweiz,, (2001).
ftp://ftp.tnt.uni-hannover.de/pub/papers/2001/AS2001-CELJBMPOS.pdf

[3] Christian Becker and Marcell Ziems and Torsten Büschenfeld and Christian Heipke and Sönke Müller and Jörn Ostermann and Martin Pahl: Multi-Hierarchical Quality Assessment of Geospatial Data.
ISPRS, (2008).

[4] Mike Heath, Sudeep Sarkar, Thomas Sanocki, and Kevin Bowyery: Comparison of Edge Detectors, A Methodology and Initial Study.
COMPUTER VISION AND IMAGE UNDERSTANDING Vol. 69, No. 1, January, pp. 3854 ARTICLE NO. IV960587, (1998).

[5] Jorge Valverde Rebaza: Detecciońn de bordes mediante el algoritmo de Canny.
Escuela Acadmico Profesional de Informtica Universidad Nacional de Trujillo, (2001).

[6] J. Canny: computational approach to edge detection.
IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 8, no. 6, November, (1998).

[7] Torsten Seemann: Digital Image Processing using Local Segmentation.
School of Computer Science and Software Engineering, Faculty of Information Technology, Monash University, Australia., (2002).
http://www.csse.monash.edu.au/ torsten/pubs/Seemann-thesis.pdf