Gebze Technical University
CSE 464 - Digital Image Processing

Field Boundary Detection
Final Project Report

Sena Özbelen - 1901042601

Fall 2023

Contents

List of Figures

1.Abstract

With the improvement of satellite technology, there is a huge source coming from satellites. Researchers can use satellite images to observe the planet. It also gives an opportunity to see land use and land cover by combining these images. For the land use case, agricultural fields are very important and sometimes, there is a need for delineation of these fields as shown in Figure 1.1.

Figure 1.1 : A real world example for field boundary detection [1]

Therefore, using these images and other sources, some algorithms and programs are developed to extract the boundaries. With the rise of AI, it becomes the main pathway to develop such an application. However, this project aims to extract those boundaries using only image processing techniques.

2.Project Details

This project has 3 parts: preprocessing, edge detection, and post-processing. With the help of some papers about this case, this project is shaped. Most applications are based on AI. Only a few of them use only image processing techniques. This project uses only RGB images from satellites so the most beneficial paper for this project is called "Extracting Agricultural Fields from Remote Sensing Imagery Using Graph-Based Growing Contours" by Matthias P. Wagner and Natascha Oppelt [2].

2.1.Preprocessing

There are some problems to get clear edges. These are trees, houses, and the color imbalance of fields. These create noise when edges are extracted. There are some techniques to remove noises but they are not sufficient to remove these problems completely. At first, I tried Gaussian blurring but it didn't give a proper output. After reading the paper and doing research, I ended up using Bilateral filtering. Additionally, I use fastNlMeansDenoising function from OpenCV. Although, they didn't completely remove these problems, they made the image smoother and preserved edges better. Also, the image is resized and converted to gray-scale. Blurring and converting to gray-scale made the edges unclear. Therefore, I applied histogram equalization via CLAHE. However, this also highlighted color imbalances too as shown in Figure 2.1.1.

Figure 2.1.1: Before and After preprocessing

2.2.Edge Detection

For this part, Using Sobel or Canny Edge Detection was in my mind. Since Canny also uses Sobel in its OpenCV implementation, Sobel is used for this project. I used 5 for kernel size. The results are shown in Figure 2.2.1.

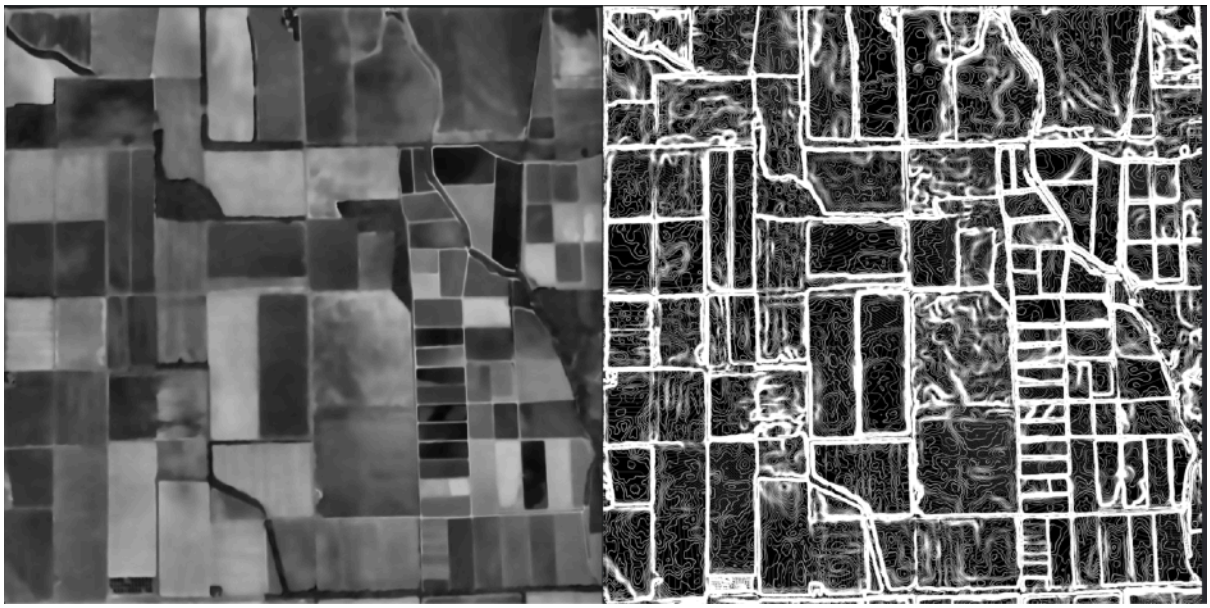

Figure 2.2.1: Extracting edges with Sobel

2.3.Post-processing

The output of the Sobel edge detection needs a post-processing step. In Canny edge detection, non-max suppression and hysteresis edge tracking are used. In the paper, Meijering filtering is used for gradient magnitudes. I tried both of them. Meijering filtering gives smoother edges and this is suitable for non-linear edges as shown in Figure 2.3.1.

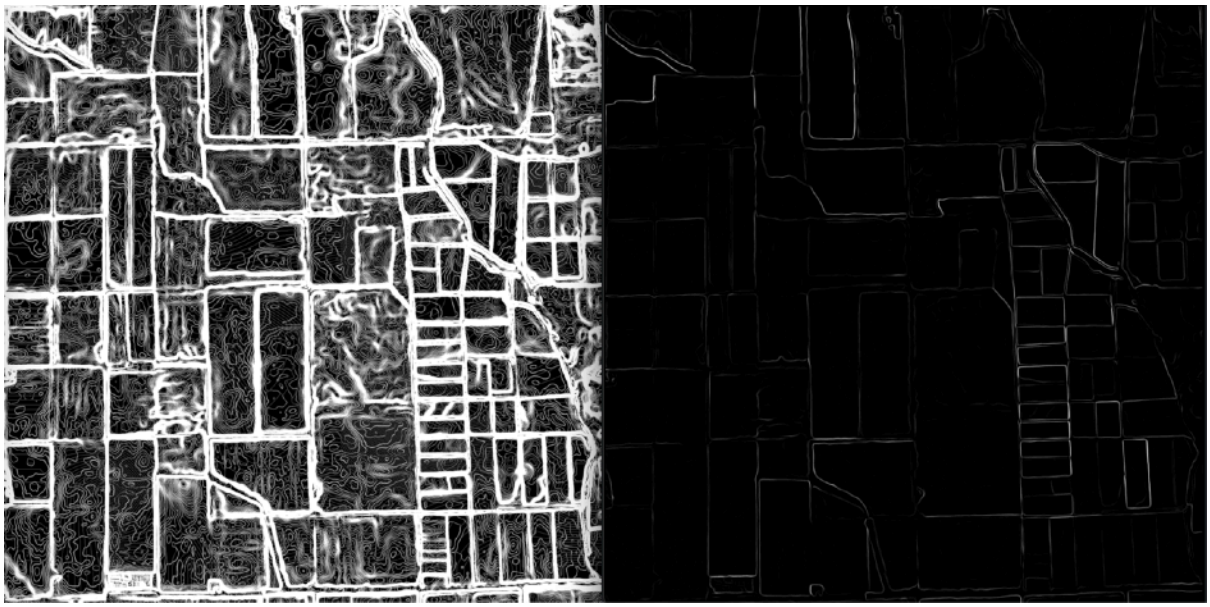

Figure 2.3.1: Before and After Meijering filtering

Therefore, I used Meijering filter and implemented hysteresis edge tracking to link and finalize the edges as shown in Figure 2.3.2. Basically, I combined these two approaches.

Figure 2.3.2: Before and After Hysteresis Edge Tracking

3.Results

The results show that noises are still there because preprocessing cannot completely remove all of them. Also, there are some missing edges because of unclearness of the edges for the field. If colors are too similar, then the edge between fields might be partially or completely removed while preprocessing and edge tracking is unable to link all the broken lines.

Figure 3.1 is a low quality image so noise is not the biggest issue. Because of the similarity of colors, there are some broken lines.

Figure 3.1

Figure 3.2 is a high quality image so trees and houses are the biggest problem since they create a lot of noise. Even though preprocessing removes partially, it still exists. This has also the similarity of colors causing broken lines.

Figure 3.2

Figure 3.3 is a high quality image and it also shows a new problem. Inside of fields, there are some edge-like patterns and these are detected as edges.

Figure 3.3

Figure 3.4 is a challenging image which has a lot non-regular shaped fields and some edges are quite blurry and faded. It has a lot of broken lines since edges are not clear on the original image too.

Figure 3.4

4.References

[1]: https://www.spacenus.com/en/field-boundary

[2]: Wagner, M.P.; Oppelt, N. Extracting Agricultural Fields from Remote Sensing Imagery Using Graph-Based Growing Contours. *Remote Sens.* **2020**, *12*, 1205. https://doi.org/10.3390/rs12071205

5.Appendix - Code

```
import cv2
import numpy as np
from skimage import filters

# Load the image
image = cv2.imread('image1.jpg')

# Resize the image
image = cv2.resize(image, (1500,1500))

# Apply bilateral filter
denoised_image = cv2.bilateralFilter(image, 9, 75, 75)

# Convert the image to grayscale
denoised_image = cv2.cvtColor(denoised_image,
cv2.COLOR_BGR2GRAY)

# Apply non-local means denoising
denoised_image = cv2.fastNlMeansDenoising(denoised_image,
None, h=10, templateWindowSize=10, searchWindowSize=21)

# Apply histogram equalization
clahe = cv2.createCLAHE(clipLimit=2.0, tileGridSize=(8, 8))
clahe_image = clahe.apply(denoised_image)

# Apply Sobel operator
sobel_x = cv2.Sobel(clahe_image, cv2.CV_64F, 1, 0, ksize=5)
sobel_y = cv2.Sobel(clahe_image, cv2.CV_64F, 0, 1, ksize=5)
sobel_magnitude = np.sqrt(sobel_x**2 + sobel_y**2)

# Apply Meijering filter
sigmas = range(1, 3)
filtered = filters.meijering(sobel_magnitude, sigmas=sigmas,
black_ridges=False)
frame = np.ones_like(filtered, dtype=bool)
d = 2 * np.max(sigmas) + 1
d += 1
frame[d:-d, d:-d] = False
```

```
filtered[frame] = np.min(filtered)

# Thresholds for hysteresis edge tracking
threshold_low = 0.03
threshold_high = 0.05

weak_edges = (filtered > threshold_low) & (filtered <=
threshold_high)
strong_edges = (filtered > threshold_high)

# Perform hysteresis edge tracking
def track_edges(i, j):
    if i < 0 or i >= weak_edges.shape[0] or j < 0 or j >=
weak_edges.shape[1]:
        return
    if weak_edges[i, j]:
        weak_edges[i, j] = 0
        strong_edges[i, j] = 1
        for ii in range(i - 1, i + 2):
            for jj in range(j - 1, j + 2):
                track_edges(ii, jj)

for i in range(1, weak_edges.shape[0] - 1):
    for j in range(1, weak_edges.shape[1] - 1):
        if weak_edges[i, j]:
            track_edges(i, j)

# Combine strong and linked weak edges into an array
all_edges = (strong_edges | weak_edges).astype(np.uint8) * 255
# Combine edges with the original image
result_image = cv2.bitwise_or(image, cv2.merge([all_edges,
all_edges, all_edges]))

image = cv2.resize(image, (800, 800))
clahe_image = cv2.resize(clahe_image, (800, 800))
sobel_magnitude =
cv2.resize(cv2.convertScaleAbs(sobel_magnitude), (800, 800))
filtered = cv2.resize(filtered, (800, 800))
all_edges = cv2.resize(all_edges, (800, 800))
result_image = cv2.resize(result_image, (800, 800))

# Display the images
cv2.imshow('Original Image', image)
cv2.imshow('Denoised Image', clahe_image)
cv2.imshow('Sobel Magnitude',
cv2.convertScaleAbs(sobel_magnitude))
cv2.imshow('Meijering Enhanced', filtered)
cv2.imshow('Hysteresis Edge Tracking', all_edges)
cv2.imshow('Result', result_image)
```

```
cv2.waitKey(0)
cv2.destroyAllWindows()
```