GEBZE TECHNICAL UNIVERSITY
DEPARTMENT OF COMPUTER ENGINEERING

CSE564 DIGITAL IMAGE PROCESSING PROJECT

ROAD EXTRACTION FROM SATELLITE IMAGES

Student Name & Surname: Enes Aysu
Student ID: 1901042671

Abstract

The project focuses on the development of an automated system for road extraction from satellite images using image processing algorithms. The objective is to create an efficient and accurate solution that aids in urban planning, navigation systems, and infrastructure management. The project utilizes a combination of K-means clustering, thresholding, morphological operations, and connected components analysis to identify and highlight road segments in satellite imagery.

Keywords

- Road extraction
- Satellite images
- Image processing
- K-means clustering
- Morphological operations
- Connected components analysis

Index

1. Introduction

1.1 Motivation

Manual mapping of roads from satellite images is a time-consuming and labor-intensive process. The motivation for this project stems from the need for automated methods to handle large-scale satellite imagery efficiently. An automated road extraction system contributes to improved urban planning and infrastructure management.

1.2 Objectives

The primary objectives of the project include:

- Development of a robust pipeline for automated road extraction.
- Implementation of image processing techniques, including K-means clustering and morphological operations.
- Utilization of connected components analysis for segmentation and refinement of road boundaries.

2. Methodology

2.1 Image Preprocessing

The process begins by reading the satellite image using the OpenCV library. To enhance the image quality and reduce noise, a median blur operation is applied. This operation is crucial for preparing the image for subsequent processing steps.

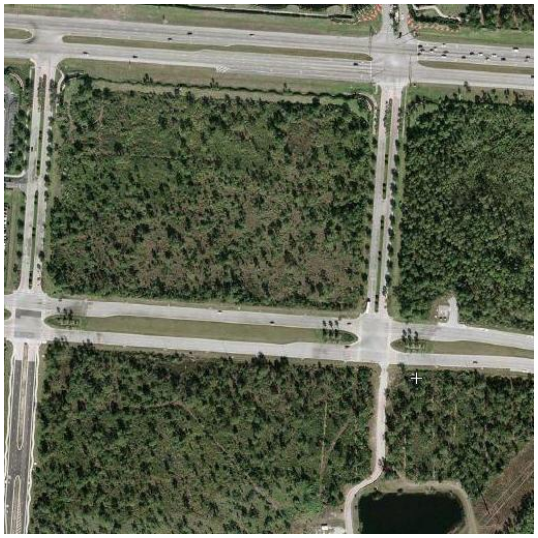

image 2.1: *original image*

image 2.2: *blurred image*

The reason for using median blur is its effectiveness in handling edge cases. This way, the edges of the path are preserved during the blurring process. Details such as trees and bushes in the field are freed from details through the blurring process. (The effect of the filter is observed when image 2.2 is compared to image 2.1.)

2.2 Color Segmentation (K-Means Clustering)

K-means clustering is employed to segment the image into distinct color regions. The image is reshaped into a 2D array of pixels, and K-means clustering is applied to group pixels with similar color characteristics. The resulting segmented image is then reshaped back to its original dimensions.

image 2.2: *blurred image*

image 2.3: *clustered image*

After specific trials, a 7-pixel clustered k-means filter was employed to simplify the road we want to extract in terms of pixel diversity. This helped prevent instantaneous pixel differences present on the road. (When image 2.2 and 2.3 are compared, simplification at the pixel level can be observed.)

2.3 Grayscale Conversion and Histogram Equalization

The segmented image is converted to grayscale to simplify further analysis. Histogram equalization is applied to enhance the contrast of the grayscale image, making details more discernible and preparing it for subsequent thresholding.

image 2.3: *clustered image*

image 2.4: *grayscaled image*

image 2.5: *equalized image*

2.4 Thresholding

A thresholding operation is performed to create a binary image. Pixels with intensity levels above a specified threshold value are set to white, representing potential road regions, while pixels below the threshold are set to black.

image 2.5: *equalized image*

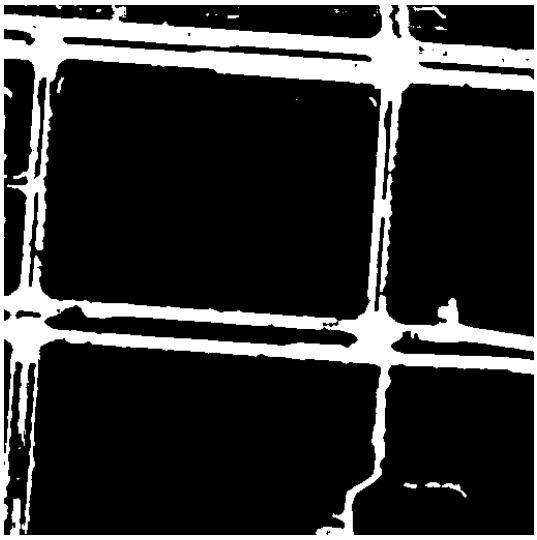

image 2.6: *thresholded image*

As a result of the histogram equalization process on the pixels belonging to the road, reaching high pixel values facilitated the thresholding process. Areas with pixel values above 210 were displayed as white, while areas below were displayed as black. (When image 2.5 and 2.6 are compared, it is observed that the thresholding process extracts the roads from the main image.)

2.5 Closing Operation

Morphological closing is applied using a kernel to fill gaps and smooth the boundaries of the thresholded image. This operation aims to create cohesive road segments for improved accuracy.

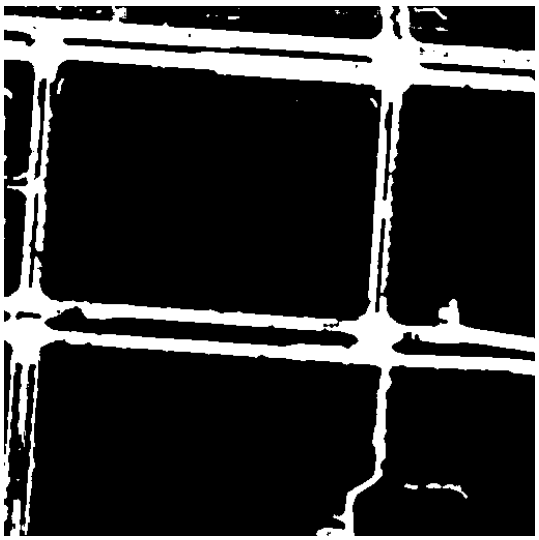

image 2.6: *thresholded image*

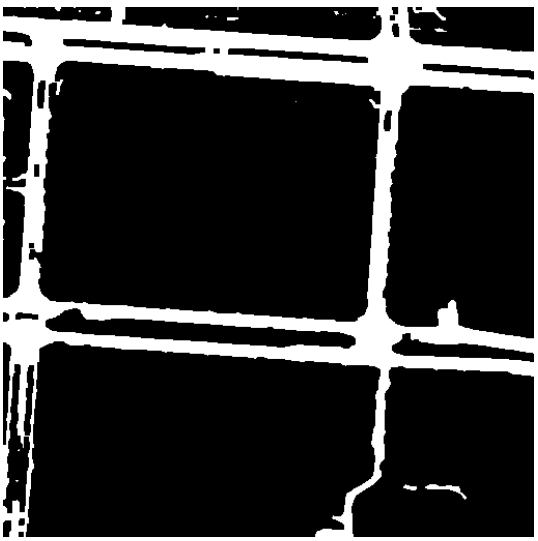

image 2.7: *closed image*

To maintain road integrity and connect disjointed pieces, dilate and erode operations were applied first and deemed appropriate. As a result, a closing filter was applied. (In image 2.6, it can be observed that gaps and disconnections on the roads have decreased in visual 2.7.)

2.6 Connecting Components Analysis (Segmentation)

Connected components analysis is introduced to segment the binary image into distinct regions. Conditions are applied to filter out non-road areas based on the size of the connected components. The resulting mask represents road segments.

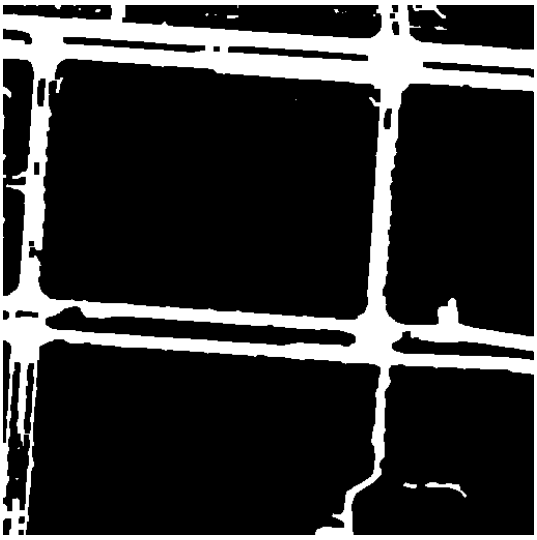

image 2.7: *closed image*

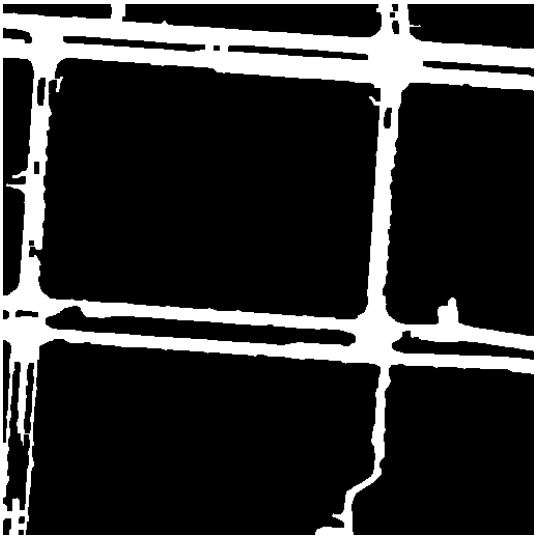

image 2.8: *segmented image*

To remove parts not connected to the road (noise and objects), segmentation was applied, marking all of them and boxes were then extracted for each segment. Segments that were detected as noise or non-road objects, not starting from the borders of the image, or having a bounding box area so close to the white area, were removed. Road bounding boxes, even though being very wide, were preserved if there wasn't much difference between their white area and their own box area. (In image 2.8, the result with noise and objects removed is displayed.)

image 2.9: *result*

3. Results

Visualizations of the intermediate and final results demonstrate the effectiveness of each processing step. The overlay of road segments on the original image highlights the success of the automated extraction process. When the morphological filters mentioned in the report are applied sequentially, they can accurately extract roads from aerial photographs of landscapes containing roads, taken from satellite images.

4. Challenges

Errors such as missing extractions or the removal of non-road objects may occur in areas with high urbanization or on newly constructed roads due to the dark color of asphalt in these regions.

image 4.1

image 4.2

As seen in image 4.1, the roads appear dark due to being newly constructed. The filters used in the project failed to detect the road, resulting in only the open parts of the road being extracted.

In image 4.2, the high level of urbanization and some houses having light-colored roofs caused non-road areas to appear similar to roads, resulting in the extraction of more regions than desired.

5. Future Enhancements

- **Adaptive Parameter Optimization:**

Implementing adaptive parameter optimization techniques could enhance the algorithm's ability to handle diverse image characteristics without manual intervention.

- **Machine Learning Integration:**

Integration of machine learning techniques, such as deep learning, could enhance the algorithm's adaptability and generalization to different geographical regions.

- **Expansion of Ground Truth Data:**

Expanding the availability of accurate ground truth data for training and evaluation purposes could improve the algorithm's performance and reliability.

6. Conclusion

The automated road extraction project from satellite images represents a significant stride toward leveraging computer vision techniques for infrastructure analysis and urban planning. The methodology employed in this project combines various image processing operations to identify and highlight road segments within satellite imagery. Despite encountering challenges inherent to diverse image characteristics and the intricacies of road features, the project has demonstrated promising results.

7. Appendix

```
import cv2
import numpy as np

def showImage(filter, name):
    cv2.imshow(f"{name}", filter)

test_image = "road.jpg"

##### Read the image #####
image = cv2.imread(f"test_images/{test_image}")
#########################

##### Blur the ima ge #####
median = cv2.medianBlur(image, 5)
#########################

##### Applying k-means clustering #####
pixels = median.reshape((-1, 3)) # Reshape the image into a 2D
array of pixels
pixels = np.float32(pixels) # Convert the data type to
floating point
k = 7 # Define the number of clusters (K)

criteria = (cv2.TERM_CRITERIA_EPS +
cv2.TERM_CRITERIA_MAX_ITER, 10, 0.5) # Apply K-means
clustering
_, labels, centers = cv2.kmeans(pixels, k, None, criteria, 10,
cv2.KMEANS_RANDOM_CENTERS)

centers = np.uint8(centers) # Convert the centers to integers
segmented_image = centers[labels.flatten()] # Map the labels
to their respective centers
# Reshape the segmented image back to the original shape
segmented_image = segmented_image.reshape(image.shape)
```

```
##### Applying gray-scale #####
gray = cv2.cvtColor(segmented_image, cv2.COLOR_BGR2GRAY )
##############################

##### Applying gray-scale #####
equalized = cv2.equalizeHist(gray)
##############################

##### Applying threshold #####
ret, thresh = cv2.threshold(equalized, 210, 255,
cv2.THRESH_BINARY)
##############################

##### Applying closing filter #####
kernel_size = 5
kernel = np.ones((kernel_size, kernel_size), np.uint8)

# Perform closing operation
closed = cv2.morphologyEx(thresh, cv2.MORPH_CLOSE, kernel)
###################################


##### Apply connected components analysis #####
num_labels, labels, stats, centroids =
cv2.connectedComponentsWithStats(closed)

mask = np.zeros_like(closed) # creating mask
min_area = 100 # for ignore white areas that below 100 pixels

# Iterate through each connected component
for i in range(1, num_labels):
    # Extract the region of interest (ROI)
    x, y, w, h, area = stats[i]
    bounding_area = w * h # finding bounding box for every
segmentation
    if w > mask.shape[1] * 0.95 or h > mask.shape[0] * 0.95 or
(area > min_area and area < bounding_area * 0.25):
        # first and second condition: for avoid losing map-
sized and straight roads
        # third condition: for get rid of structures outside
the road and white noise smaller than 100 pixels
        # (if the white area < bounding box area * 0.25,
program counts the white area as a road))
        mask[labels == i] = 255
###############################################
```

```
# Convert the single-channel image to a 3-channel image
red_parts = cv2.cvtColor(mask, cv2.COLOR_GRAY2BGR)

# Set the color to red (BGR format) where the thresholded
image is white
red_parts[mask > 0] = [0, 0, 255]

# Create a mask where with red parts
red_mask = red_parts[:, :, 2] == 255  # Check if the red
channel is fully red

# Replace the corresponding pixels in the original RGB image
with the red parts
overlayed = np.copy(image)
overlayed[red_mask] = red_parts[red_mask]

# Display every step results
showImage(image, "original")
showImage(median, "blurred")
showImage(segmented_image, "k-means")
showImage(gray, "gray")
showImage(equalized, "equalized")
showImage(thresh, "thresholded")
showImage(closed, "closed")
showImage(mask, "segmented")
showImage(overlayed, "overlayed")

cv2.imwrite(f"output_images/{test_image}", mask)
cv2.imwrite(f"output_images/overlay_{test_image}", overlayed)

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