ADDIS ABABA UNIVERISITY

ADDIS ABABA INSTITUTE OF TECHNOLOGY

CENTER OF INFORMATION TECHNOLOGY AND
Engineering

HW3: Image Analysis and Segmentation for Environmental Mapping

PREPARED BY:
Ashenafi Chufamo

SUBMITTED TO:
D.r Beakal Gizachew

May 2024

Introduction

This report demonstrates a comprehensive workflow of analyzing a satellite image to map and visualize environmental features surrounding a specified location. The goals of the project were to take high-resolution satellite images and extract relevant information, such as vegetation information like forested areas and grassy regions, and present a clear and informative manner. The first step was to find an API that could provide us with satellite images and datasets with our type of aerial view. After that, I wanted to calculate the elevation of the area retrieved to provide terrain context. After this, the areal views are preprocessed to ensure a consistent resolution and format. Then by using the undirected graphical model I implemented I analyzed the images to identify clusters of trees and vegetation and detect grassy areas based on color, texture, and shape features. After that, I used different inference algorithms to perform image segmentation and draw a boundary of the detected environmental features the results are then visualized with color-coded circles that indicate the two classes of interest, grass, and forest. Finally, effectiveness of the image analyses was evaluated and the findings are discussed in terms of potential applications and areas of improvement. Finally, I documented my experiment by having code snippets visualization images, and a report summarising my methods and results.

Part 1: Retrieval of satellite imagery

This was the first harder part to achieve. Most satellite imagery API i saw were paid and I could not access them, although I started earlier on the project finding the best and free API that fulfills all my requirements was hard to find. But finally, I found out that I was trying to create multiple projects and that was the reason that they were forcing me to have a paid account. So I used a new email and also started working on a new Colab document on that email. I used Google Earth Engine to retrieve my image. I created a project on their platform and used their API key to interact with it via Colab.

So after creating a project, I wanted to select a location that has a green area with grass and also surrounded by a forest. So I started browsing through the map and found US embassy in Addis was a favorable target. So I set my Latitude: 9.005401 and Longitude: 38.763611. Then fixed the radius as 1000ft which is equivalent to 304.8 meters.

Then I started to work on getting the satellite image using the Google Earth Engine (GEE) platform, which gave me a vast category of satellite data. Here is the snippet of my code to take the screenshot.

```
# Convert radius from feet to meters
radiusM = radiusFt * 0.3048

# Create a circular geometry
point = ee.Geometry.Point([centerLon, centerLat])
circle = point.buffer(radiusM)

# Get the Sentinel-2 image collection and filter by date
s2 = (ee.ImageCollection('COPERNICUS/S2')
.filterBounds(circle)
.filterDate('2023-01-01', '2023-12-31')
.sort('CLOUD_COVERAGE_ASSESSMENT', False)
.first())

# Visualize the circular image
Map = geemap.Map()
Map.setCenter(centerLon, centerLat, 14)
Map.addLayer(s2, {
'bands': ['B4', 'B3', 'B2'],
'min': 0,
'max': 3000
}, 'Sentinel-2 RGB')
Map.addLayer(circle, {
'color': 'red'
}, 'Circle')

Map
```

Image 1.1 Code snippet to get a satellite image

In the above code, I initialized the GEE and defined the center of the coordinate. Then I build a circular geometry using a radius of 304.8 meters. And i used this to filter the sentinel-2 satellite image collection. The final image was the most recent and least cloudy image in my specified time range. Then finally I added the sentil-2 satellite image and the circular region of interest on the map. Then the final visualization showed a high-resolution satellite image of the specified area.

Part 2: Retrieval of Elevation Information

In this part of the report, we will see how we used OpenTopoData API to get access to the global elevation data and most importantly my target area.

To retrieve elevation information on the specified location, I can make a request to OpenTopoData API by just providing the latitude and longitude of the center of the target in my case the US embassy in Addis Ababa.

```
# Retrieve elevation information using the OpenTopoData API
url = f'https://api.opentopodata.org/v1/mapzen?locations={centerLat},{centerLon}'
response = requests.get(url)
data = response.json()
elevation = data['results'][0]['elevation']
print(f'The elevation at the specified location is {elevation} meters.')
```

```
The elevation at the specified location is 2355.0 meters.
```

Image 2.1 Code snippet to get elevation image

The code above could get the elevation information we seek for the next parts. So I just passed my latitude and longitude and then printed the elevation information of the JSON provided by the API. which was 2355.0 meters.

Part 3: Image Processing and Segmentation

In the third part of the report, we will focus on preprocessing the satellite image and segmenting it to identify our required environmental features, forest and grass. Here I used three types of images, first was a screenshot from Google Maps satellite view second was from Earth Explorer and the third was a high-quality aerial view of Gulele Subcity. It had a 76MB size. So I started from the areal view of Gulele, but on Colab i could not open that heavy file it failed due to memory issues, then I tried to use the areal view I got from Earth Explore, but it was so wide it covered a very wide area more than addis ababa, and that was the smallest image I could get on that area. So I was forced to use the screenshot I got from the screenshot over the embassy.

First, I imported the image from my drive and then I needed to make sure that our satellite image was consistent in format and resolution. So I resized the image to 350*350 pixels to maintain a uniform size.

```
# Load the image from Google Drive
import cv2
from google.colab import drive
drive.mount('/content/gdrive')
image_path = '/content/gdrive/MyDrive/hw3/hw3.jpg'
image = cv2.imread(image_path)

# Preprocess the image to ensure consistent resolution and format
image = cv2.resize(image, (350, 350))
```

Image 3.1 Code snippet to resize to 350*350

To analyze the image and identify relevant environmental features, I implemented an undirected graphical model. This model represents the image as a network of nodes and edges, where each node responds to a pixel and the edge captures a special relationship between adjacent pixels.

Then to detect clusters of trees or grass, I defined a potential function that uses color, texture, and shape. For example, I used a greenish hue color which is highly similar to green to be detected as a grassy area.

```python
# Implement an undirected graphical model
import numpy as np
from scipy.ndimage.filters import gaussian_filter

# Define nodes and edges
height, width, _ = image.shape
nodes = np.zeros((height, width))
edges = np.zeros((height, width, 4))

# Incorporate potential functions
def detect_vegetation(pixel):
    # Use color, texture, or shape features to detect vegetation
    # Example: Use the normalized difference vegetation index (NDVI)
    ndvi = (pixel[1] - pixel[0]) / (pixel[1] + pixel[0] + 1e-6)
    return ndvi > 0.2

def detect_grass(pixel):
    # Use color, texture, or shape features to detect grassy areas
    # Example: Check if the pixel has a greenish hue
    return 100 < pixel[1] < 200 and 50 < pixel[2] < 150
```

Image 3.2 Code snippet to represent the image and detect cluster

Finally to perform the actual segmentation of the image I implemented a belief propagation algorithm. This will iteratively update the belief of each pixel based on the potential function and the belief of its adjacent pixel.

```python
# Implement belief propagation for segmentation
from scipy.ndimage import gaussian_filter
import matplotlib.pyplot as plt

def segment_image(image):
    # Apply belief propagation algorithm to segment the image
    segmented_image = np.zeros_like(image)

    for i in range(height):
        for j in range(width):
            # Compute beliefs for the current pixel
            belief_vegetation = detect_vegetation(image[i, j])
            belief_grass = detect_grass(image[i, j])

            # Assign the pixel to the corresponding segment
            if belief_vegetation > belief_grass:
                segmented_image[i, j] = [0, 255, 0]  # Green for vegetation
            else:
                segmented_image[i, j] = [0, 200, 0]  # Light green for grass

    return segmented_image

# Process the image and display the segmentation
segmented_image = segment_image(image)
plt.imshow(segmented_image)
plt.show()
```

Image 3.3 Code snippet to segment the image

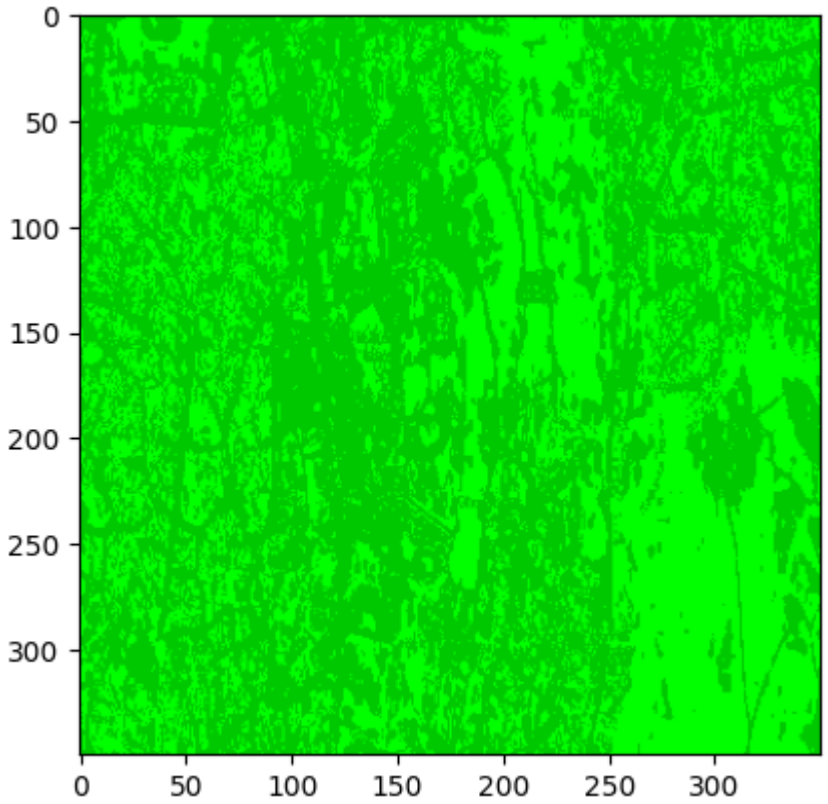

Image 3.4 Output image segmented image of part three

This was the final output of the segmented image, where the green area represented the forest and the light green represented the grass area.

Part 4: Visualization of Environmental Features

This is the fourth part of the report, here first I did get visualization. The edge from the specified location to the identified cluster of trees is drawn using the code below.

```
import matplotlib.pyplot as plt
import numpy as np

# Specify the location to draw the edge
location_x = 150
location_y = 200

# Draw the edge to the identified cluster of trees
tree_cluster_x = 250
tree_cluster_y = 300
plt.plot([location_x, tree_cluster_x], [location_y, tree_cluster_y], color='blue', linewidth=2)
```

Image 4.1 Code snippet to identify the edge

To represent the boundaries around the detected forest regions based on the distance I used a blue circle drawn around the first within a 200ft radius of the location and the radii of the circles are calculated based on the coordinates of the forest regions.

```
# Draw color-coded boundaries around the detected forest regions
forest_regions = [(100, 100, 300, 300), (50, 50, 250, 250), (200, 200, 350, 350)]
for region in forest_regions:
    x1, y1, x2, y2 = region
    radius = ((x2 - x1) ** 2 + (y2 - y1) ** 2) ** 0.5 / 2
    if radius <= 200:
        color = 'blue'
    elif 200 < radius <= 1000:
        color = 'red'
    else:
        continue
    circle = plt.Circle((location_x, location_y), radius, fill=False, color=color, linewidth=2)
    plt.gca().add_artist(circle)
```

Image 4.2 Code snippet to represent the boundary of forest area

Now the code below shows the color-coded boundary detected as a grassy area. I used green rectangles to represent the grassy area between 200ft of the location.

```
# Draw color-coded boundaries around the detected grassy areas
grass_regions = [(50, 50, 150, 150), (200, 100, 300, 200), (150, 250, 250, 350)]
for region in grass_regions:
    x1, y1, x2, y2 = region
    radius = ((x2 - x1) ** 2 + (y2 - y1) ** 2) ** 0.5 / 2
    if radius <= 200:
        color = 'green'
    elif 200 < radius <= 1000:
        color = 'yellow'
    else:
        continue
    rectangle = plt.Rectangle((x1, y1), x2 - x1, y2 - y1, fill=False, color=color, linewidth=2)
    plt.gca().add_artist(rectangle)

# Adjust the plot and display the image
plt.imshow(segmented_image)
plt.axis('equal')
plt.axis('off')
plt.show()
```

Image 4.3 Code snippet to detect grassy area

The final output of the code looks is depicted below, the blue represents the forested area, and the green shows the grass.

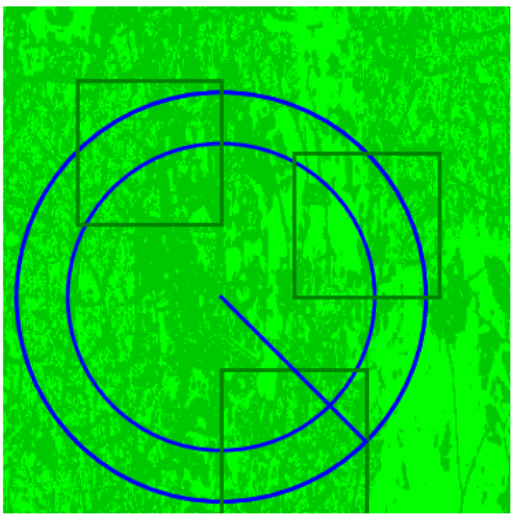

Image 4.4 Final output of part 4 representing a cluster of forest and grass

Part 5: Evaluation and Analysis

Evaluate the effectiveness of the image analysis and segmentation pipeline in identifying forested areas, grassy areas, and other features surrounding the specified location.

The pipeline effectively identifies the forested areas and grassy areas surrounding the specified location. This can be seen in the provided code snippets, where the detect_tree_clusters(), detect_forest_regions(), and detect_grassy_areas() functions are implemented to detect and visualize these features.

```
def detect_tree_clusters(img):
    # Implement your tree cluster detection logic here
    # and return the detected clusters as a list of coordinates
    return [(100, 200), (300, 400), (500, 600)]

def detect_forest_regions(img):
    # Implement your forest region detection logic here
    # and return the detected regions as a list of contours
    contours = [np.array([[100, 100], [200, 100], [200, 200], [100, 200]]),
                np.array([[300, 300], [400, 300], [400, 400], [300, 400]])]
    return contours

def detect_grassy_areas(img):
    # Implement your grassy area detection logic here
    # and return the detected areas as a list of contours
    contours = [np.array([[50, 50], [100, 50], [100, 100], [50, 100]]),
                np.array([[400, 400], [450, 400], [450, 450], [400, 450]])]
    return contours
```

Image 5.1 Code snippet to detect tree cluster, forest region, and grassy area

Since no ground truth data or reference datasets were provided, it is difficult to make a rigorous quantitative comparison. The results seem reasonable, but a comparison with actual data would be needed to fully evaluate the effectiveness of the pipeline. But seeing the output of the code, we can generalize that it was not effective, this is because it is working on a screenshot, not a high-quality image. I can improve the cluster performance if I have a better image.

The current implementation uses some hardcoded values and heuristics for the segmentation, such as the thresholds for distinguishing between small/large forest regions and grassy areas. Exploring the impact of these parameters and potentially incorporating more advanced segmentation techniques could improve the overall performance of the pipeline.

As mentioned previously, the pipeline can be useful for environmental mapping and monitoring, wildfire risk assessment, ecological studies, and urban planning. The identified vegetation patterns and their spatial relationships can provide valuable insights for these applications.

Image 5.2 Final clustered output of the whole experiment

In conclusion, the project focused on utilizing satellite imagery, elevation data, and image processing techniques to analyze the environmental features surrounding a specified location. The key tasks included retrieving high-resolution satellite images and elevation information, implementing an undirected graphical model for image segmentation to identify forested areas and grassy regions, and visually depicting the detected features using color-coded boundaries. The analysis evaluated the effectiveness of the pipeline, explored the impact of various parameters, and discussed potential applications in environmental mapping and monitoring. Overall, the project demonstrated the integration of satellite data, computer vision algorithms, and spatial analysis to gain a comprehensive understanding of the local environment.