Digital Image Processing

Project September 2023

264 Priyanshi Mohapatra 212784
267 Nitya Nair 212851

‘Mapping Urban Green Regions’

Department of Physics
St.Xavier’s College
Mumbai, India

Abstract

In this project, we have explored the usage of digital image processing techniques to calculate the green area in a given satellite image. Using Jupyter Lab and Python, we have completed the process. First, we identified the red border, and with the help of the OpenCV library, we identified the contour, applied red mask followed by the green mask, and calculated green pixels within the red border.

Content

Introduction

Digital image processing refers to the manipulation, enhancement, and analysis of digital images using computer algorithms and techniques. There are many advantages to it, such as image sharpening and smoothening, image restoration, remote sensing, etc. Our project explores how DIP is used in remote sensing, that is, ‘finding the green percentage’ of a certain area using its image.

‘Green areas’ or ‘Green spaces’ are becoming an important part of infrastructure as they have proven to reduce surface temperature and serve as insulation for structures below from excessive heat and pollution. Hence, proper planning of these areas by calculating the existing areas and predicting future areas is needed. Here, our aim is to show and map the green regions in the given satellite image.

Background

Our code carries out two main functions: one is identifying contours, and the other is applying masks.

Masking- Masking is a process used in graphics software such as Photoshop to hide and reveal portions of an image. It is a non-destructive process of image editing. One common method is to use a masking layer, which is essentially a second image that is used to cover up parts of the first image.

Contours - Contours can be explained simply as a curve joining all the continuous points along the boundary and having the same color or intensity. The contours are a useful tool for shape analysis, object detection, and recognition. Finding contours in OpenCV is comparable to finding a white object on a black background. Consequently, the background should be black and the object to be found should be white.

Theory

In terms of color image processing, the hardware-oriented models most commonly used in practice are the RGB (red, green, blue) model for color monitors and a broad class of color video cameras; the CMY (cyan, magenta, yellow) and CMYK (cyan, magenta, yellow, black) models for color printing; and the HSI (hue, saturation, intensity) model, which corresponds closely with the way humans describe and interpret color. In the RGB model, each color appears in its primary spectral components of red, green, and blue. Full-color image processing approaches fall into two major categories. In the first category, we process each component image individually and then form a composite processed color image from the individually processed components. In the second category, we work with color pixels directly. Highlighting a specific range of colors in an image is useful for separating objects from their surroundings. The basic idea is either to (1) display the colors of interest so that they stand out from the background or (2) use the region defined by the colors as a mask for further processing. One of the simplest ways to "slice" a color image is to map the colors outside some range of interest to a nonprominent neutral color.

The computation of the standard deviations is done only once using sample color data. Given an arbitrary color point, we segment it by determining whether or not it is on the surface or inside the box, as with the distance formulations. However, determining whether a color point is inside or outside a box is much simpler computationally when compared to a spherical or elliptical enclosure.

Algorithm

1. Import the required libraries.

2. Load the image using OpenCV.

3. Define the lower and upper bounds for green and red colors in BGR format.

4. Create masks for green and red areas based on color ranges.

5. Find the contours in the red mask.

6. Get the largest contour (red border).

7. Create a mask for the red border.

8. Apply the green mask within the red border.

9. Calculate the total number of green and red pixels within the red border.

10. Calculate the percentage of green cover within the red border.

11. Display the original image and green mask within the red border.

12. Print the percentage of green cover within the red border.

Commands explanation:

1. image_path: This is the path to the image file.

2. cv2.imread(): This function reads an image from the specified path.

3. cv2.cvtColor(): This function is used to convert the color space of an image. Here it converts loaded BGR image to Gray image and RGB image.

4. cv2.inRange(): the function creates a binary mask for pixels that fall within the specified range of values in the RGB color space. It is used here to create a mask to identify the red areas.

5. cv2.findContours(): this function detects contours in a binary image. It returns a list of contours found in the image.

6. cv2.contourArea(): This function calculates the area of contour.

7. cv2.drawContours(): This function is used to draw the contours on an image. It draws the largest detected contour (here, a red border) on the empty mask.

8. cv2.countNonZero() : returns the number of nonzero pixels in the array matrix. This function can come in handy when calculating the number of white pixels.

9. cv2. RETR_EXTERNAL: This retrieval mode retrieves only the external contours.

10. cv2.CHAIN_APPROX_SIMPLE: compresses horizontal, vertical, and diagonal segments and leaves only their end points. For example, an up-right rectangular contour is encoded with 4 points.

11. cv2.bitwise_and(): This function performs a bitwise AND operation on two arrays element-wise. Here, it applies the red ontour mask to the green mask, effectively restricting the green mask to the area within the red border.

12. np.zeros_like(image): This numpy method returns an array of the given shape and type as a given array, with zeros.

13. Matplotlib plot functions: The matplotlib plot function is a function that allows users to create plots of data. The function takes a number of arguments, including the x-coordinates, the y-coordinates, and the type of plot to be created.

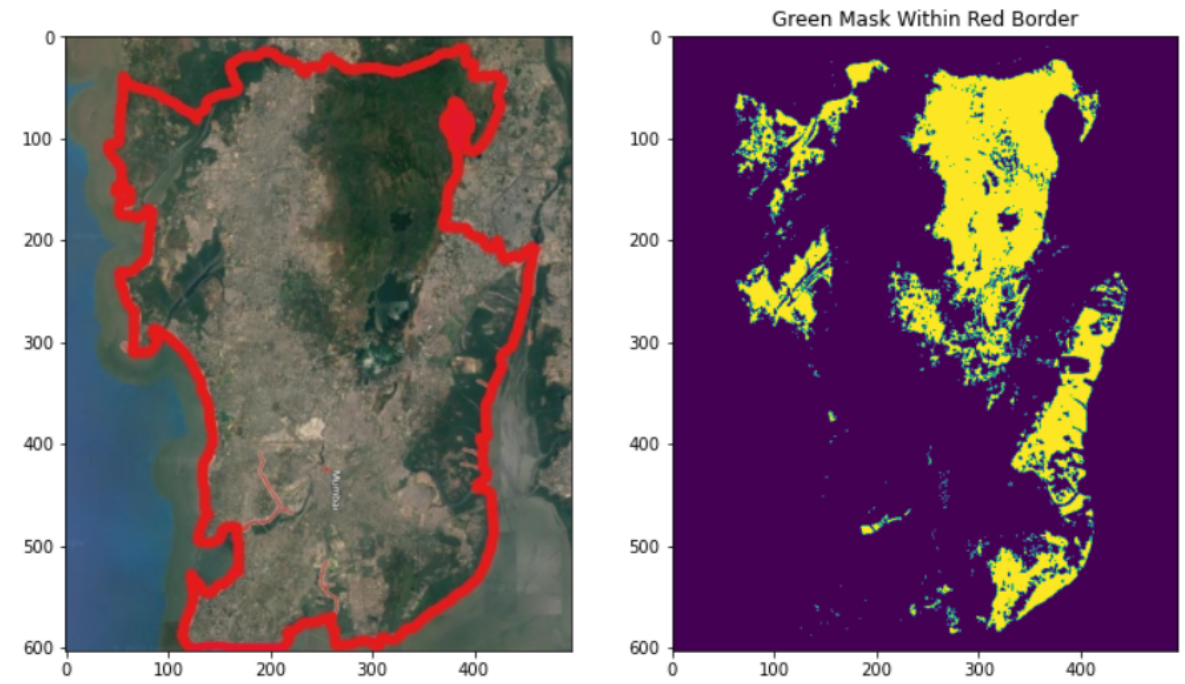
Code

```
# import cv2
import matplotlib.pyplot as plt
import numpy as np

# Load the image using OpenCV
image = cv2.imread(r"C:\Users\Shubham Mohapatra\Desktop\green area~2.jpg")
# Convert the image to grayscale
gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
rgb= cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

# Define the lower and upper bounds for green and red colors in BGR format
lower_green = np.array([0, 50, 0], dtype=np.uint8)
upper_green = np.array([70, 255, 70], dtype=np.uint8)
lower_red = np.array([0, 0, 50], dtype=np.uint8)
upper_red = np.array([50, 50, 255], dtype=np.uint8)

# Create masks for green and red areas based on color ranges
green_mask = cv2.inRange(image, lower_green, upper_green)
red_mask = cv2.inRange(image, lower_red, upper_red)

# Find contours in the red mask
contours, _ = cv2.findContours(red_mask, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)

# Get the largest contour (red border)
if contours:
    largest_contour = max(contours, key=cv2.contourArea)

# Create a mask for the red border
red_contour_mask = np.zeros_like(gray_image)
cv2.drawContours(red_contour_mask, [largest_contour], -1, (255), thickness=cv2.FILLED)

# Apply the green mask within the red border
green_mask_within_red = cv2.bitwise_and(green_mask, red_contour_mask)
```

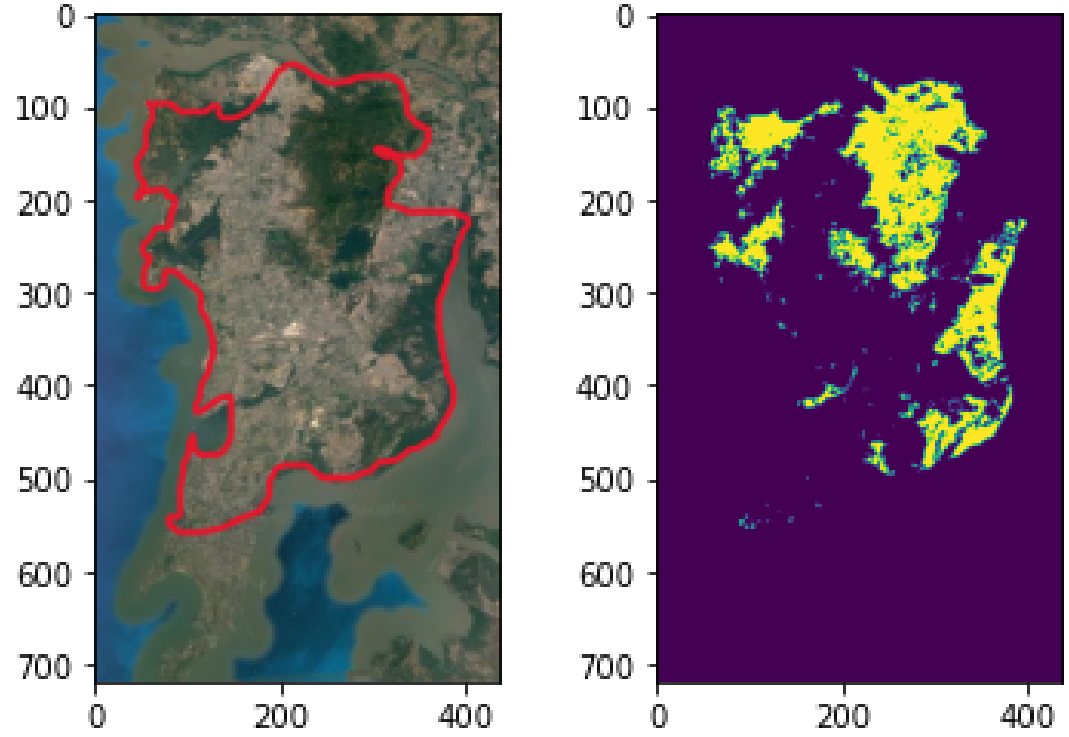
```
# Calculate the total number of green and red pixels within the red border
total_green_pixels = cv2.countNonZero(green_mask_within_red)
total_red_pixels = cv2.countNonZero(red_contour_mask)

# Calculate the percentage of green cover within the red border
if total_red_pixels > 0:
    green_percentage_within_red = (total_green_pixels / total_red_pixels) * 100
else:
    green_percentage_within_red = 0

# Display the original image and the green mask within the red border
plt.figure(figsize=(12, 8))
plt.subplot(1, 2, 1)
plt.imshow(rgb)
plt.subplot(1, 2, 2)
plt.imshow(green_mask_within_red)
plt.title("Green Mask Within Red Border")
plt.show()

print(f"Percentage of green cover within red border: {green_percentage_within_red:.2f}%")
```

Results

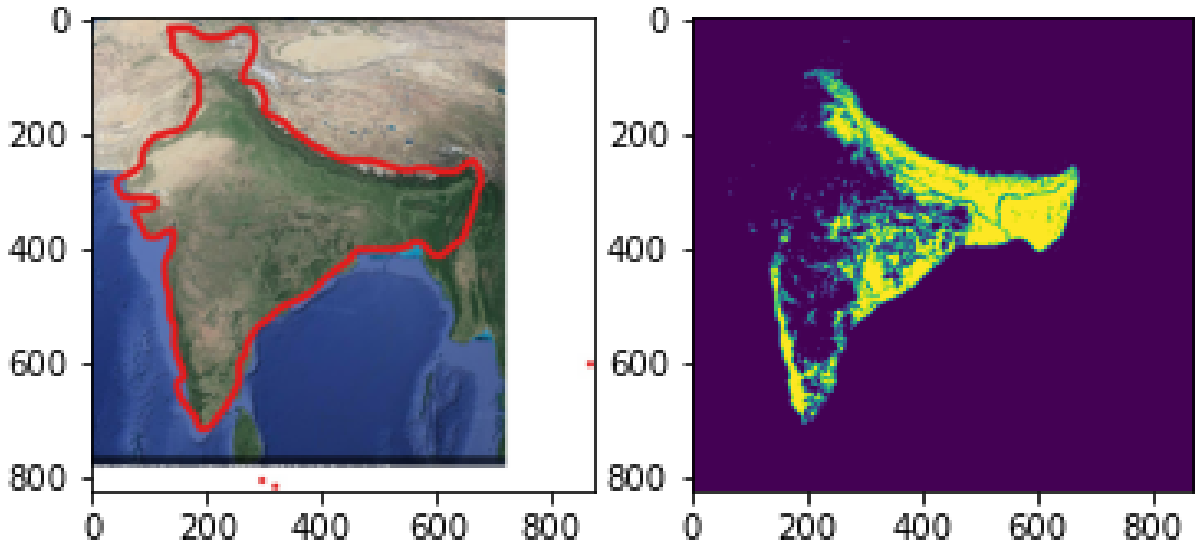
Percentage of green cover within the red border of the image provided : **24.87%**



Percentage of green cover on Mumbai in 2023 : **27.16%**



Percentage of green cover of India in 2023: **34.45%**



Conclusion

We have successfully identified the red border and applied a green mask to it. Hence, calculated the green percent within the red border and showed the output image. Similarly, we can identify different areas and even classify further vegetation in a given area.

Application

As stated above, DIP has many applications in many fields like Remote sensing, medical fields, space exploration, chemistry, textiles, etc. Color segmentation and identifying contours. You can use color segmentation in a wide variety of machine vision applications, such as the following:

1. Inspection: Partition an image into different regions based on the colour of the part in each region.
2. Counting: Segment an image to quickly count the number of objects with a particular colour composition.

References

1. Digital Image Processing, 3rd Edition- By Gonzalez
2. Google Earth
3. https://stackoverflow.com/