Efficient Road Quality Estimation Using Image Processing and Digital Elevation Models

Siddhant Paliwal

June 11, 2024

Abstract

This research aims to identify the most efficient method for estimating road quality using machine learning techniques. Initially, the task was broad, requiring us to define the specific aspects of road quality. Key indicators of road quality include potholes, crevices, distributed gravel, and tar, which can be identified through photographs. However, manually scanning extensive road networks is impractical. Therefore, our objective was to leverage machine learning and image analysis to quickly and accurately assess road quality. We narrowed our focus to gravel roads, commonly used and presenting various challenges. Additionally, we employed digital elevation models (DEMs) to gain insights into road defects like potholes, crevices, and elevation changes. This led to the primary focus of this research: the efficient analysis of DEMs to identify sections of poor road quality.

Data collection presented another challenge. We utilized a dataset from Vernal Pools near Merced, consisting of drone-photographed roads. Our research aims to propose a more efficient method for analyzing DEMs and to highlight parts of the dataset indicative of poor road quality. By demonstrating the use of image processing techniques and Python for DEM analysis, this research contributes to the development of automated systems for road quality assessment. Ultimately, our project seeks to enhance the efficiency and accuracy of road maintenance, improving travel safety and infrastructure management.

1 Introduction

Efficient road quality estimation is crucial for infrastructure maintenance and travel safety. Traditional methods of road inspection are labor-intensive and time-consuming. This research seeks to automate the process using machine learning and digital elevation models (DEMs). By focusing on gravel roads, which present unique challenges, we aim to develop a robust system for identifying road defects such as potholes and crevices through image analysis.

2 Procedure

2.1 Data Collection

A dataset from Vernal Pools near Merced was acquired using drone flights. The data consisted of images of gravel roads captured from various angles. These images were processed to create Digital Elevation Models (DEMs), which provide a visual representation of elevation changes, with darker shades indicating higher elevations and lighter shades indicating lower elevations. Approximately ten DEMs were generated and subsequently converted to JPEG format for further processing.

2.2 Image Processing

The DEMs were subjected to a Python script designed to slice each model into multiple segments. Each DEM was divided into 10-12 slices, carefully aligned with the edges of the road. This slicing was performed intelligently to ensure that, regardless of the road's orientation, vertical, horizontal, and diagonal slices were accurately extracted. The Python code used for this slicing is as follows:

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

# Load the image
image_path = 'section10.png'
image = cv2.imread(image_path)
```

```
# Convert to grayscale
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

# Apply thresholding to create a binary image
_, binary = cv2.threshold(gray, 1, 255, cv2.THRESH_BINARY)

# Find contours
contours, _ = cv2.findContours(binary, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

# Create a mask of the road
mask = np.zeros_like(gray)
cv2.drawContours(mask, contours, -1, (255), thickness=cv2.FILLED)

# Extract the road using the mask
road_only = cv2.bitwise_and(image, image, mask=mask)

# Get the bounding box of the road
x, y, w, h = cv2.boundingRect(contours[0])

# Crop the image to the bounding box
cropped_image = road_only[y:y+h, x:x+w]

# Define the number of slices
num_slices = 12
slice_height = w // num_slices

# Slice the cropped image into multiple vertical sections without background
for i in range(num_slices):
    start_col = i * slice_height
    end_col = (i + 1) * slice_height if i != num_slices - 1 else w

    # Crop each vertical slice
    slice_img = cropped_image[:, start_col:end_col]

    # Remove any remaining white background around the road in each slice
    gray_slice = cv2.cvtColor(slice_img, cv2.COLOR_BGR2GRAY)
    _, binary_slice = cv2.threshold(gray_slice, 1, 255, cv2.THRESH_BINARY)
    slice_contours, _ = cv2.findContours(binary_slice, cv2.RETR_EXTERNAL, cv2.CHAI

    if slice_contours:
        x_slice, y_slice, w_slice, h_slice = cv2.boundingRect(slice_contours[0])
        slice_img = slice_img[y_slice:y_slice+h_slice, x_slice:x_slice+w_slice]

    # Save or display the slices
    slice_path = f'slice_{i+108}.png'
    cv2.imwrite(slice_path, slice_img)
    plt.imshow(cv2.cvtColor(slice_img, cv2.COLOR_BGR2RGB))
    plt.title(f'Slice {i+1}')
    plt.show()
```

The sliced images were then categorized into three folders based on their orientation: horizontal, vertical, and diagonal.

2.3 Color Shift Analysis

Each categorized image was analyzed to detect color shifts. Horizontal slices were examined for vertical color shifts, vertical slices for horizontal color shifts, and diagonal slices for shifts along a 45-degree angle. This analysis is crucial because significant changes in elevation, as indicated by color shifts, are indicative of road defects.

The following Python script was employed for color shift analysis:

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

# Constants
DIAGONAL_DIR = 'diagonal/'
VERTICAL_DIR = 'vertical/'
IMAGE_SIZE = (224, 224)  # Example size, adjust based on your requirements

# Load and preprocess image
def load_and_preprocess_image(file_path):
    image = cv2.imread(file_path, cv2.IMREAD_UNCHANGED)  # Load with transparency
    if image.shape[2] == 4:  # Check if image has an alpha channel
        image = cv2.cvtColor(image, cv2.COLOR_BGRA2RGBA)  # Ensure 4 channels (RGB
    image = cv2.cvtColor(image, cv2.COLOR_RGBA2RGB)  # Convert to RGB
    return image

# Create a mask to identify non-transparent regions
def create_mask(image):
    alpha_channel = image[:, :, 3] if image.shape[2] == 4 else None
    if alpha_channel is not None:
        mask = alpha_channel > 0
    else:
        gray = cv2.cvtColor(image, cv2.COLOR_RGB2GRAY)
        _, mask = cv2.threshold(gray, 1, 255, cv2.THRESH_BINARY)
    return mask

# Crop the image to the bounding box of the non-transparent regions
def crop_to_non_transparent(image, mask):
    coords = np.column_stack(np.where(mask > 0))
    y0, x0 = coords.min(axis=0)
    y1, x1 = coords.max(axis=0) + 1  # slices are exclusive at the top
    cropped_image = image[y0:y1, x0:x1]
    return cropped_image

# Detect main diagonal color shifts (top-left to bottom-right)
def detect_diagonal_color_shifts(image):
    height, width, _ = image.shape
    color_shifts = []

    for d in range(-height + 1, width):
        diag = np.diagonal(image, offset=d)
        diag_shifts = np.mean(diag[1:] != diag[:-1], axis=1)
        color_shifts.extend(diag_shifts)

    return np.array(color_shifts)

# Detect horizontal color shifts
def detect_horizontal_color_shifts(image):
    height, width, _ = image.shape
    color_shifts = np.mean(image[:, 1:] != image[:, :-1], axis=1)
    return color_shifts

# Process directory for color shifts and plot individual graphs
def process_directory(directory, direction='diagonal'):
    for filename in os.listdir(directory):
        file_path = os.path.join(directory, filename)
        image = load_and_preprocess_image(file_path)

        # Create mask to identify non-transparent regions
        mask = create_mask(image)

        # Crop the image to the bounding box of the non-transparent regions
        cropped_image = crop_to_non_transparent(image, mask)

        # Detect color shifts based on the specified direction
        if direction == 'diagonal':
            color_shifts = detect_diagonal_color_shifts(cropped_image)
        elif direction == 'horizontal':
            color_shifts = detect_horizontal_color_shifts(cropped_image)
        else:
            raise ValueError("Invalid direction. Use 'diagonal' or 'horizontal'.")

        # Only take data from 200 to 500 pixels
        if len(color_shifts) > 500:
            color_shifts = color_shifts[200:500]
        else:
            color_shifts = color_shifts[200:]

        # Plot the individual graph for color shifts
        plt.figure(figsize=(10, 5))
        plt.plot(color_shifts, color='green' if direction == 'diagonal' else 'blue
        plt.title(f'{direction.capitalize()} Color Shifts in {filename}')
        plt.xlabel('Pixels')
        plt.ylabel('Color Shift')
        plt.show()

# Process diagonal images for diagonal color shifts
```

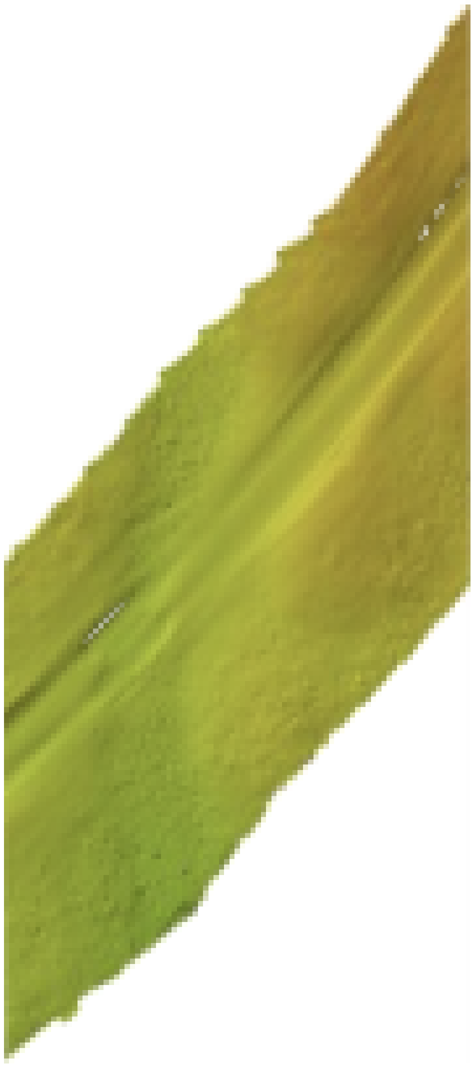
```
process_directory(DIAGONAL_DIR, direction='diagonal')

# Process vertical images for horizontal color shifts
process_directory(VERTICAL_DIR, direction='horizontal')
```

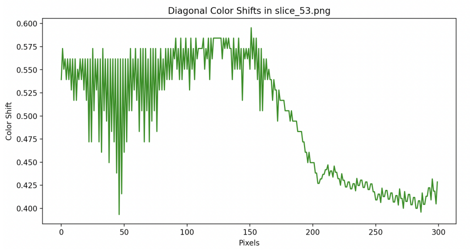
2.4 Graphical Analysis and Interpretation

Graphs were plotted for the color shifts of each slice. High variance in these graphs indicated significant color shifts, correlating with notable elevation changes in the road cross-sections. Additionally, darker spots in the images were associated with high color shifts, suggesting the presence of potholes. There is a noticeable example on the last page.

2.5 Objective and Future Work

The primary objective is to model the data and attribute it to each road section, determining their quality ranking and indexing. The goal is to fully automate this process and expand it to analyze more datasets, enhancing its applicability and robustness.


(a) Image 1 description



(b) Image 2 description

Figure 1: An example of diagonal color shift seen with graph.