Water Body (Rivers, Lakes, Reservoirs) Detection & Mapping

Muhammad Aarish Mughal
Reg Id. **223570**
Department of Computer Science
Multan, Punjab, Pakistan.
aarishmughal21@gmail.com

Aymen Majid
Reg Id. **223574**
Department of Computer Science
Multan, Punjab, Pakistan.
aymenmajid440@gmail.com

ABSTRACT

This project focuses on detecting and mapping water bodies such as rivers, lakes, and reservoirs using satellite imagery and digital image processing techniques. Traditional manual mapping methods are time-consuming and less accurate, especially in changing environmental conditions. By applying filters, thresholding, and segmentation methods like NDWI and U-Net, we aim to automate the identification of water regions in satellite images. The system also computes the area of detected water bodies, which supports resource management, flood monitoring, and environmental planning. The project combines remote sensing, image analysis, and machine learning to deliver a fast and efficient solution with potential for real-time monitoring and future enhancements.

INTRODUCTION

Detecting and mapping water bodies helps in monitoring floods, managing resources, and planning cities. Using satellite images and remote sensing makes this faster and more accurate.

PROBLEM STATEMENT

Traditional water body mapping relies heavily on manual efforts and field surveys, which are time-consuming and limited in scope. Moreover, environmental challenges such as seasonal changes, cloud cover, and sediment interference further complicate manual analysis. With the increasing availability of satellite imagery, there is a pressing need to develop automated, accurate, and scalable solutions for detecting and mapping water bodies in various terrains and climatic conditions.

OBJECTIVES

1. To apply filtering concepts learned in our course Digital Image Processing.
2. To automate the detection of water bodies using satellite/aerial imagery.
3. To compute the area of water bodies via computers.

RELATED WORK

Researchers have explored a variety of approaches for detecting water bodies in satellite imagery. One of the most widely used techniques is the **Normalized Difference Water Index (NDWI)**, proposed by McFeeters, which enhances water features by leveraging the difference between near-infrared and green spectral bands. Variants like **Modified NDWI (MNDWI)** have also been developed to improve accuracy in urban and turbid regions.

Beyond index-based methods, **machine learning algorithms** such as Support Vector Machines (SVM), Random Forests (RF), and k-Nearest Neighbours (k-NN) have been applied to classify water and non-water pixels based on multispectral features. These models require manually labelled training data and often depend on handcrafted features, which may limit their scalability and generalizability across diverse terrains.

In recent years, **deep learning approaches** have gained significant traction due to their superior performance in image segmentation tasks. Architectures like **U-Net**, **SegNet**, and **DeepLabV3+** have been employed to segment water bodies from high-resolution satellite images. These models are capable of learning complex spatial features automatically and typically outperform

traditional methods in terms of accuracy and robustness.

Several studies have also incorporated **multitemporal analysis** to track seasonal changes in water extent, and **SAR (Synthetic Aperture Radar)** data has been integrated for better detection under cloud cover or during flood events.

Overall, the trend in recent literature indicates a shift toward deep learning methods, especially in cases where large, annotated datasets are available, due to their ability to generalize across varied geographies and imaging conditions.

THEORY

NDWI Formula:

$$NDWI = \frac{Green - NIR}{Green + NIR}$$

Basic Concepts:

- NDWI highlights water in images.
- U-Net is a deep learning model used for image segmentation.
- Accuracy is checked using Precision, Recall, and IoU.

DESIGN

Flowchart

A Flowchart of the initial design about our project is given in the appendix.

Method Used:

Filtering & Geometry.

Tools:

GitHub, Python, OpenCV, TensorFlow, ReactJs, Flask API, Bootstrap, VSCode, Git Copilot, OpenAI's ChatGPT, Anthropic's Claude 4 Sonnet.

Evaluation:

Detect water bodies via Satellite Images and Computing the area of these water bodies via filtering.

IMPLEMENTATION

We have developed this project application and structured into three abstract-level Python modules. We have implemented a few features related to our project for now. Those features are as follows:

- Contour Detection
- Save Filtered Image
- Dynamic Blue Range Tuning
- Pixel Information of the Image
- Percentage of Water Body Pixels

Unlike last time, our application now has a UI based on the JavaScript (web) framework: **ReactJs**. This has made the application run smoothly and act responsively. This opens the application to even more users who are on mobile and other platforms if they have access to the internet.

Each module has a distinct role in the architecture of the application.

- App.py
- Processing.py
- Utils.py
- UpdateForm.js
- App.js

These files contain different pieces of python code that work together to make this application work. These files are also attached separately with the submission. To provide a preview of the implementation, small snippets of the code from each module are given in the appendix.

DEBUGGING-TEST-RUN

At first, the first problem we faced was that the input image could not be opened. After a few trial runs with the input image being in the parent directory of the file **app.py,** we settled on allowing the user to choose the input image file via the Windows Explorer which proved to be very convenient.

The second issue was that the output image would not be shown. We fixed this issue by showing the output image file (the input image after processing is performed), also allowing the user to save this output image file for later analysis.

One of the few other errors included the sliders not working as expected but we fixed those with the help of our AI helper: **ChatGPT**. *(This tool is a life saver when the bugs you face aren't usual or can be fixed via **Stack overflow, etc.**)*

RESULTS ANALYSIS

When we were finished with the application, it provided robust and viable results. The satellite images were not only hard to decode but also a challenge for our application. But our application worked flawlessly and provided results. These results along with the inputs are attached in the appendix of this report.

CONCLUSION

Looking at this project, we believe that this would be impossible without the core concepts that we were taught in the **Digital Image Processing** class. Different concepts like Colour Space Conversion, Binary Masking, Edge Detection, Contour Detection, etc., were all implemented in this project which further improved our concepts.

FUTURE IMPROVEMENTS

Our future intentions with this project are to improve its user interface and improve its UI. Another option that could be implemented is the use of **Django** – the python framework for making quick web application.

GITHUB REPOSITORY

All the files of this project are maintained at the GitHub Repository accessible via this url:

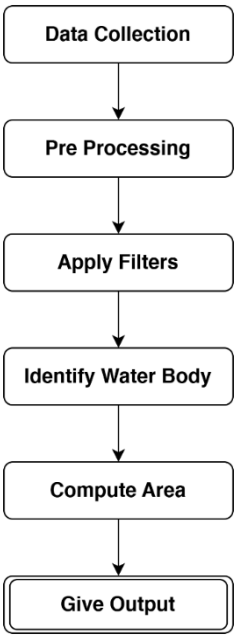
https://github.com/Aarishmughal/dip_project_water_body_detection

BIBLIOGRAPHY

- **OpenCV Documentation. (n.d.).** *Open-Source Computer Vision Library*. https://docs.opencv.org/
- **Gonzalez, R. C., & Woods, R. E. (2018).** *Digital Image Processing (4th ed.). Pearson.*
- **Shapiro, L. G., & Stockman, G. C. (2001).** *Computer Vision. Prentice Hall.*

APPENDIX

Flowchart of the Project (initial)



Code Snippets

App.py

```
from flask import Flask, request,
jsonify, send_file
from flask_cors import CORS
import numpy as np
import cv2
import io
import os
from PIL import Image
from processing import
detect_water_and_contours

app = Flask(__name__)
CORS(app)

@app.route('/process',
methods=['POST'])
def process_image():
    file = request.files['image']
    hsv_values = request.form

    img =
Image.open(file.stream).convert("RGB
")
    img_np = np.array(img)
    img_cv = cv2.cvtColor(img_np,
cv2.COLOR_RGB2BGR).....
```

Processing.py

```python
import cv2
import numpy as np
import os

def
detect_water_and_contours(image_bgr,
hsv_range):
    hsv = cv2.cvtColor(image_bgr,
cv2.COLOR_BGR2HSV)

    lower_blue, upper_blue =
hsv_range
    mask = cv2.inRange(hsv,
lower_blue, upper_blue)

    kernel = np.ones((5, 5),
np.uint8)
    mask_cleaned =
cv2.morphologyEx(mask,
cv2.MORPH_OPEN, kernel)
    mask_cleaned =
cv2.morphologyEx(mask_cleaned,
cv2.MORPH_CLOSE, kernel)

    water_pixel_count =
cv2.countNonZero(mask_cleaned)
    total_pixels =
mask_cleaned.shape[0] *
mask_cleaned.shape[1]
    percentage = (water_pixel_count
/ total_pixels) * 100

    water_only =
cv2.bitwise_and(image_bgr,
image_bgr, mask=mask_cleaned)
```

Utils.py

```python
import cv2
from PIL import Image, ImageTk

def convert_cv_to_tk(cv_img):
    img_rgb = cv2.cvtColor(cv_img,
cv2.COLOR_BGR2RGB)
    pil_img =
Image.fromarray(img_rgb)
    return
ImageTk.PhotoImage(pil_img)
```

UpdateForm.js

```javascript
import { useState, useRef } from
"react";
import axios from "axios";
import
"bootstrap/dist/css/bootstrap.min.cs
s";
import
"bootstrap/dist/js/bootstrap.bundle.
min.js";
import "bootstrap-
icons/font/bootstrap-icons.css";
function UploadForm() {
    const [image, setImage] =
useState(null);
    const [processedImage,
setProcessedImage] = useState(null);
    const [showEdges, setShowEdges]
= useState(false);
    const [stats, setStats] =
useState(null);
    const imageSrc = showEdges
        ?
"http://localhost:5000/edges"
        :
"http://localhost:5000/processed";

    const [hsv, setHSV] = useState({
        h_min: 85,
        h_max: 135,
        s_min: 30,
        s_max: 255,
        v_min: 20,
        v_max: 255,
    });
```

Results Analysis

Input Image 01

Output Image 01

Input Image 02

Output Image 02

Screenshot of App