Detailed Technical Documentation, README, and SOP for the Code

---

Technical Documentation

1. Overview

The provided project processes drone-captured images to:

1. Convert them into gridded grayscale images.
2. Identify potential hazards in the images based on dynamic brightness thresholds.
3. Group hazards into clusters using connected grid logic.
4. Plan drone paths to efficiently navigate through identified hazard clusters.
5. Visualize the paths and generate animations of drone navigation.

---

2. Key Components

main.py

Handles the orchestration of image processing:

- Reads images from the specified folder.
- Converts them to grayscale with overlaid grids.
- Dynamically calculates brightness thresholds to identify potential hazards.
- Groups hazards into clusters and plans drone paths using the `ClusterPathPlanner`.
- Visualizes and saves drone paths.

grid_and_grayscale.py

Defines the `DefineGrayScale` class:

- Converts an image to 16-bit grayscale.
- Overlays a grid on the image for hazard analysis.

red_hazards.py

(Not explicitly shown but assumed based on usage in `main.py`.)

- Highlights grid cells with hazards using calculated thresholds.
- Extracts hazard grid information for clustering.

neighbors.py

Defines the `IdentifyNeighbors` class:

- Identifies neighboring grid cells to determine connected clusters of hazards.
- Computes centers of these clusters for drone path planning.

path_planning.py

Defines the `ClusterPathPlanner` class:

- Uses KMeans clustering to group hazard clusters into manageable groups.
- Plans efficient drone paths using a nearest-neighbor algorithm.
- Visualizes and animates the planned paths.

---

3. Input and Output

Input

- Images from the `drone_images` folder (16-bit PNG format).
- Grid size and dynamic thresholds defined in `main.py`.

Output

- **Grayscale Images**: Saved in the `grayscale_drone_images` folder.
- **Hazard Images**: Saved in the `potential_hazards` folder.
- **Hazard Grid Coordinates**: Text files in the `hazard_grid_coordinates` folder.
- **Drone Paths**: Images and animations in the `drone_paths` folder.

---

4. Functions and Parameters

Main Functions

1. **`process_image_files()`**
    - Processes all images, detects hazards, and plans paths.
    - Inputs:
        - Image folders, grid size, and other parameters.
    - Outputs:
        - Grayscale images, hazard maps, cluster information, and drone paths.
2. **`calculate_dynamic_thresholds()`**

   - Dynamically calculates brightness thresholds based on image average brightness.
3. **`ClusterPathPlanner`**
   - `split_clusters()`: Groups centroids into manageable clusters using KMeans.
   - `nearest_neighbor_path()`: Computes efficient paths using a nearest-neighbor algorithm.
   - `plot_paths()`: Visualizes planned paths on an image.
   - `animate_paths()`: Creates an animation of drone navigation paths.

---

5. Requirements

Python Libraries

- **Core Libraries**: `os`, `math`, `io`, `random`.
- **External Libraries**:
  - `numpy`: For numerical operations.
  - `Pillow`: For image processing.
  - `matplotlib`: For plotting and animations.
  - `scikit-learn`: For KMeans clustering.

File Requirements

- Input folder: `drone_images/`.
- Output folders: Created automatically if not present.

---

6. Edge Cases

- If no images are found in the input directory, the script exits gracefully.
- Non-PNG images are ignored.
- Handles scenarios where clusters are too few for the requested number of groups.

---

7. Performance Considerations

- KMeans clustering uses multiple initializations (`n_init=10`) for stability.

- Dynamic threshold calculations ensure robust hazard detection under varying lighting conditions.

README.md

Drone Image Processing and Path Planning

Overview
This project processes drone-captured images to:
1. Convert images to gridded grayscale.
2. Identify hazards based on brightness thresholds.
3. Cluster hazards and plan efficient drone paths.
4. Visualize and animate drone paths.

Features
- Dynamic Hazard Detection: Adjusts brightness thresholds based on image properties.
- Clustered Path Planning: Groups hazards and computes paths using KMeans and nearest-neighbor algorithms.
- Visualization and Animation: Saves static and animated visualizations of drone paths.

Installation

1. Clone the repository:

```bash
git clone <repository-url>
cd <repository-folder>
```

2. Install dependencies:
   pip install -r requirements.txt

Usage

1. Place drone images in the `drone_images/` folder (16-bit PNG format).

   Run the script:
   python main.py

2.
3. Results are saved in:
   - Grayscale images: `grayscale_drone_images/`.
   - Hazard images: `potential_hazards/`.

    - Hazard coordinates: `hazard_grid_coordinates/`.
    - Drone paths: `drone_paths/`.

Requirements

- Python 3.8+
- External Libraries:
    - `numpy`
    - `Pillow`
    - `matplotlib`
    - `scikit-learn`

Output

- Grayscale images with grids.
- Hazard-highlighted images and coordinates.
- Visualized and animated drone paths.

License

SOP (Standard Operating Procedure)

1. Preparing Inputs

1. Install Python 3.8 or later.
2. Clone the repository and navigate to its folder.
3. Install the required Python libraries using:
```bash
pip install -r requirements.txt
```

---

2. Preparing Inputs

1. Ensure drone images are saved in the `drone_images/` folder.
2. Images must be 16-bit grayscale PNG files.

---

3. Running the Script

1. Open a terminal or command prompt.
2. Run the script:
   ```
   python main.py
   ```

---

4. Analyzing Outputs

1. **Grayscale Images**:
   - Located in `grayscale_drone_images/`.
2. **Hazard Maps**:
   - Hazard-highlighted images in `potential_hazards/`.
   - Hazard coordinates in text files within `hazard_grid_coordinates/`.
3. **Drone Paths**:
   - Images and GIFs in `drone_paths/`.

---

5. Troubleshooting

- **Issue**: Missing output folders.
  - **Solution**: The script automatically creates necessary folders.
- **Issue**: Unexpected errors.
  - **Solution**: Ensure input images are valid 16-bit PNG files.

---

6. Maintenance

Regularly update libraries using:

```
pip install --upgrade -r requirements.txt
```

- Back up output folders before running new analyses.

---

requirements.txt (create a file called this and have the below text inside)

```
numpy
Pillow
matplotlib
scikit-learn
```