

Thermal Image Analysis Script

by IDEAS Science (Béla Mihalik and Györgyi Bela)

version: 20/03/2025

Purpose

This script processes thermal orthophotos (top-down images) together with a color palette to estimate the temperature of different surfaces (e.g., roofs, streets). It clusters these surfaces by color/material and calculates statistical information about their temperatures. Finally, it generates graphs to illustrate the distribution of temperatures across different surface classes.

Table of Contents

1. [Overview of Processing Steps](#)
 2. [Detailed Explanation by Code Section](#)
 3. [Script Outputs](#)
 4. [Usage Example](#)
 5. [System Requirements](#)
 6. [Conclusion](#)
-

Overview of Processing Steps

1. Reading and Validating Input

- Accepts paths for:
 - Orthophoto (visual spectrum image).
 - Thermal photo.
 - Structure mask (buildings, streets).
 - Color palette (vertical strip of colors → temperature range).
- Takes numeric parameters for temperature range, shadow threshold, and number of clusters.

2. Preprocessing and Masking

- Ensures uniform image dimensions (resizing if needed).
- Extracts building and street masks from `structure_mask`.
- Brightness normalization for orthophoto areas (roofs, streets).

3. Shadow Processing (Optional)

- Identifies shadowed pixels on roofs via brightness thresholding.
- Excludes those shadowed regions from further analysis.

4. Thermal-to-Temperature Conversion

- Uses a color palette image to map each thermal pixel to a temperature value.
- Saves this “temperature map” into a cache file (`.npy`) to avoid recalculating.

5. Automatic Color Clustering (K-Means)

- Clusters the normalized orthophoto pixels into `num_clusters` classes for roofs, and similarly for streets.
- Each class roughly corresponds to a color/material group.

6. Statistical Computation

- For each cluster: calculates min, max, mean, median, std. dev. of temperature.
- Maps cluster color → descriptive name.

7. Graph Generation

- **Temperature Distribution:** Bar chart of mean temperature (with error bars for std. dev.).
- **Coverage:** Shows coverage percentage of each class and their mean temperatures.

8. Optional Debug Outputs

- Saves intermediate images (masks, normalized regions, shadow visuals) when `--save-debug-images` is specified.

9. Results and Logging

- Logs steps, warnings (e.g., image resizing).
- Saves the final graphs (PNG) and optionally debug images.

Detailed Explanation by Code Section

1. Imports and Initial Setup

```
import os
```

```
import argparse
```

```
import time
```

```
import logging
```

```
from datetime import datetime
```

```
import hashlib
```

```
import math
```

```
import cv2
```

```
import numpy as np
```

```
import matplotlib.pyplot as plt
```

```
from sklearn.cluster import KMeans
```

```
from scipy.spatial import distance
```

- **Key libraries:**
 - `OpenCV (cv2)` for image I/O and operations.
 - `numpy` for array handling.
 - `matplotlib` for plotting.
 - `sklearn.cluster.KMeans` for color clustering.
 - `scipy.spatial.distance` for distance metrics.
 - `tqdm` (used later) for progress bars.

2. Color Classification Functions

- **`rgb_to_hsv_roof_name(rgb)`**
Converts an RGB triplet → HSV, then infers a descriptive roof category (e.g., "Asphalt-Black", "Burgundy", "Cardinal-Red") based on hue, saturation, value, and an RGB standard deviation metric.
- **`rgb_to_hsv_street_name(rgb)`**
Similar logic but labels street/road surfaces according to standard marking or surface types (e.g., "Yellow Centerline", "Fresh Asphalt", "Weathered Concrete").
- **`hsv_to_rgb(hsv)`**
Converts an HSV triplet (scaled in script) back to RGB (0–255).
- **`rgb_to_gray(rgb)`**
Approximates brightness (grayscale value) from an RGB color. Used to order clusters from darkest to lightest (albedo).

3. Image Processing Functions

- **`normalize_brightness(img)`**
Finds the max brightness in non-zero pixels and scales them so that max becomes 255. Leaves zero (or fully black) areas unchanged.
- **`create_mask_from_structure(structure_mask, color_range_bgr, name)`**
Creates a binary mask for pixels within a specified BGR color range in `structure_mask`. Useful for extracting regions like buildings or streets.

4. Shadow Processing

- **`process_shadows(orthophoto, roof_mask, shadow_level)`**
Identifies shadowed pixels (under a brightness threshold) in roof regions.
- **`create_shadows_visualization(ortho_photo, shadow_mask)`**
Overlays a semi-transparent blue color where `shadow_mask` is present for visualization.

5. Thermal Image Processing

- **`thermal_to_temperature(thermal_img, color_palette, lowest_temp, highest_temp)`**
Converts each pixel in the thermal image to a temperature by:
 1. Extracting a vertical palette strip from `color_palette`.
 2. Mapping each pixel's color to the nearest color in the palette (in RGB space).
 3. Assigning the corresponding temperature from a linear scale between `lowest_temp` and `highest_temp`.

6. Clustering and Class Masks

- **`create_automatic_class_masks(normalized_orthophoto_masked, num_clusters)`**
Runs K-Means to create `num_clusters` classes from the non-zero orthophoto pixels. Returns:
 - A list of binary masks, one per class.
 - The RGB cluster center for each class.

7. Statistical Computations

- **`calculate_temperature_statistics(temperature_map, class_masks, class_color_centers, color_to_name_func)`**
For each class (mask), gathers all pixel temperatures and computes:
 - min, max, mean, median, std. dev.

- cluster color name (using `color_to_name_func` like `rgb_to_hsv_roof_name`).
- logs the results.
- **`calculate_structure_statistics(structure_mask)`**
Measures coverage of buildings and streets by:
 1. Counting all colored pixels.
 2. Creating building/street masks, counting each separately.
 3. Returning % coverage.

8. Creating Graphs

- **`create_temperature_graph(class_temp_stats, mask_stats, scenario_name, output_name)`**
Creates a bar chart of mean temperatures per class (with error bars for std. dev.), labeling min/max, pixel counts, etc. Saves to a PNG file.
- **`create_coverage_graph(class_temp_stats, percentage_all_classes, scenario_name, output_name)`**
Shows each class's coverage percentage (with a bar) along with mean temperatures. Also saves to a PNG file.
- **`calculate_possible_temperature_rediction(class_temp_stats, scenario_name)`**
Logs a rough estimate of how much the average temperature might be reduced if higher-temperature classes were replaced by the class with the lowest mean temperature.

9. Main Processing Flow (`main()`)

1. **Argument Parsing**
Reads arguments like `--orthophoto`, `--thermalphoto`, `--lowest-temperature`, `--num-clusters`, etc.
2. **Image Reading & Dimension Handling**
Reads the four images. Resizes thermal and structure mask to match the orthophoto if needed.
3. **Mask Extraction**
Uses `create_mask_from_structure` to isolate *building* (blue) and *street* (yellow) areas.
4. **Applying Masks & Normalizing**
Focuses on building/street regions in both orthophoto and thermal image, normalizing brightness of the orthophoto sections.

5. **Shadow Removal** (if `shadow_level > 0`)

Identifies roof shadows and removes them from further processing.

6. **Temperature Map Generation**

- Either loads a cached `.npy` (if it exists) or computes a new temperature map via `thermal_to_temperature`.
- Uses a hash to detect changes in input images or parameters.

7. **Analysis for Roofs**

- K-Means cluster the normalized building region.
- Compute temperature stats.
- Generate temperature distribution and coverage graphs.
- Log possible temperature reduction scenario.

8. **Analysis for Streets**

- Repeat clustering and stats for the street region.
- Generate graphs and logs for streets.

9. **Debug Outputs**

- If `--save-debug-images` is on, saves intermediate images.

10. **Final Output**

- Saves generated graphs for both roofs and streets.
- Displays logs in console.

Script Outputs

1. **Log Messages** in console (or whichever logging handler is configured).
2. **Cached Temperature Map** (in `cache/` directory) with a unique `.npy` filename based on a hash.
3. **Graphs** as PNG images:
 - Temperature distribution graphs (e.g., `_stat_for_roof_temperatures.png`, `_stat_for_streets_stat_for_roof_temperatures.png`).
 - Coverage graphs (e.g., `_stat_for_roof_coverage.png`, `_stat_for_streets_stat_for_roof_coverage.png`).
4. **Optional Debug Images**:

- Various intermediate images like masked orthophoto, masked thermal, shadow visualization, etc.
-

Usage Example

```
python3 thermal_analysis.py \
```

```
--orthophoto ortho_example.png \
```

```
--thermalphoto thermal_example.png \
```

```
--structuremask structure_mask.png \
```

```
--colorpalette palette.png \
```

```
--lowest-temperature 15 \
```

```
--highest-temperature 40 \
```

```
--shadow-level 30 \
```

```
--num-clusters 5 \
```

```
--output results \
```

```
--scenario-name "Downtown Survey" \
```

```
--save-debug-images
```

- Reads the specified images.
 - Treats the palette range as 15°C to 40°C.
 - Considers pixels below 30% brightness (in roofs) to be shadow.
 - Clusters each region into 5 classes (for roofs and streets separately).
 - Outputs graphs labeled with "**Downtown Survey**" in their titles.
 - Saves debug images, if requested.
-

System Requirements

- **Python 3.x**
- **Required Packages:**
 - **opencv-python** (cv2)
 - **numpy**
 - **matplotlib**

- `scikit-learn`
- `scipy`
- `tqdm`

Install with:

```
pip install opencv-python numpy matplotlib scikit-learn scipy tqdm
```

Conclusion

This script provides a complete pipeline for analyzing thermal orthophotos in conjunction with standard orthophotos and structure masks. It identifies, clusters, and compares the temperature of various urban surfaces (buildings, streets), and produces detailed statistics and visualizations. The features—shadow exclusion, brightness normalization, color-based classification, caching, and robust logging—make it well suited to repeated large-scale analysis.