Beach Crowd Counter

Technical Summary: Algorithms, Implementation, and Performance

1. Introduction

This project implements a **people counter for beach images** using only **classical computer vision** in a Jupyter notebook (beach_crowd_detection_interactive.ipynb). The goal is to estimate how many people appear in fixed camera images of a beach.

The system does **not** use deep learning. Instead, it uses a simple but effective pipeline:

- contrast enhancement with CLAHE and gamma correction,
- spatial and color masking to remove sky and sand,
- grayscale, morphology, and adaptive thresholding to obtain a binary mask,
- blob detection with geometric filters to approximate each person as one blob,
- and global metrics (MAE and error percentage) to evaluate performance.

All steps are implemented directly in the notebook and can run on a normal CPU.

---

2. Pipeline Overview

The method works image by image. For each beach image, the notebook applies the same sequence of operations:

1. **Load image and ground truth** (manual head annotations).
2. **Preprocess image**: CLAHE, top mask, gamma, Gaussian blur.
3. **Remove sand** using an HSV-based color mask.
4. **Create a binary mask** with morphology and adaptive thresholding.
5. **Detect blobs** with SimpleBlobDetector.
6. **Count blobs** and compare with ground-truth people count.

Parameters (gamma, mask thresholds, morphology size, blob filters) are first explored with an **interactive widget** for a few images, only to find a reasonable set of values. Then a **separate loop cell** applies these fixed parameters to all images and computes the final performance.

---

3. Algorithms and Implementation

This section explains the main algorithms used in the notebook and how they are implemented.

3.1 Image Loading and Ground Truth

For each image file:

- The image is read with OpenCV and converted from BGR to RGB (for correct display with Matplotlib).
- Ground-truth annotations are stored in a CSV file with one point per person (x, y coordinate of the head).
- The **ground-truth count** (C_{gt}) for that image is simply the number of rows in the CSV corresponding to that file.

These counts are used later to compute the error of the detector.

3.2 CLAHE (Contrast Enhancement)

The first processing step is **CLAHE** (Contrast Limited Adaptive Histogram Equalization) applied to a grayscale version of the image:

- Convert the RGB image to grayscale.
- Apply CLAHE with a clipLimit parameter.
- Use the CLAHE result as an enhanced version of the luminance.

Reasoning: beach images often have **bright sand** and **dark shadows**. CLAHE increases local contrast in dark areas (where people may be) without producing too much noise. This helps the following steps to separate people from the background.

In the notebook, clahe_clip is a parameter that can be changed with a slider during the exploration phase.

3.3 Top Mask (Remove Sky and Horizon)

The next step is a simple **top mask**:

- Let top_mask_percent be a value such as 0.40.
- Compute the corresponding row index h * top_mask_percent (where h is image height).
- Replace all pixels above this row with a constant gray value.

Reasoning: in typical beach cameras, people appear mainly in the **lower part** of the image. The top part usually contains **sky, sea, and distant buildings** that are not relevant and may create false detections. Masking this area simplifies the problem and reduces noise.

3.4 Gamma Correction

After masking, the notebook applies **gamma correction** to adjust brightness:

- It uses a power-law transformation with a gamma value (0 < < 1), for example (= 0.35).
- When (< 1), dark pixels are brightened more than bright ones.

Reasoning: many people stand in **shadowed regions** (under umbrellas, close to walls, or far from the camera). Gamma correction makes these people more visible while keeping bright sand under control.

In the notebook, gamma is a slider in the interactive cell. The best value is found by trial and error on a few sample images.

3.5 Gaussian Blur

To reduce noise, a **Gaussian blur** with a small kernel is applied:

- If gaussian_size is positive (e.g. 3 or 5), the image is blurred with that kernel size.
- If gaussian_size is 0, this step is skipped.

Reasoning: sand produces many **small bright and dark dots**. Gaussian blur smooths this high-frequency texture but keeps the general shape of people. This makes later thresholding and blob detection more stable.

3.6 HSV Sand / Non-Sand Mask

The blurred image is converted to **HSV** color space to separate sand from other objects:

- hsv_s_max is the **maximum saturation** considered as sand (sand is usually low-saturation beige).
- hsv_v_min is the **minimum brightness** considered as sand (sand is relatively bright).

Pixels with (S <) and (V >) are classified as **sand**. All other pixels are **non-sand**.

The pipeline keeps only the non-sand region for the next steps. Sand pixels are removed or darkened.

Reasoning: people, umbrellas, and towels often have **higher saturation** or different brightness than sand. Removing sand reduces many false positives caused by sand texture or small shadows.

The thresholds hsv_s_max and hsv_v_min are also controlled by sliders in the interactive phase.

3.7 Morphology and Adaptive Thresholding

From the non-sand image, the notebook creates a **binary mask** of possible people:

1. Convert the non-sand image to grayscale.
2. Apply morphological **opening** and **closing** with a square kernel of size morph_size:
   - Opening (erosion followed by dilation) removes small isolated noise.
   - Closing (dilation followed by erosion) fills small gaps inside blobs.
3. Apply **adaptive Gaussian thresholding** with parameters adaptive_block_size and adaptive_c.

Adaptive thresholding computes a local threshold for each pixel based on its neighborhood, and then classifies pixels as foreground or background.

Reasoning: lighting on the beach is not uniform; global thresholding would fail in many places. Adaptive thresholding adjusts to local brightness, keeping people visible in both bright and dark zones. Morphology smooths the shapes and reduces random artifacts, so the blobs look more like compact objects.

3.8 Blob Detection with SimpleBlobDetector

The last step is to detect blobs in the binary mask using **OpenCV's SimpleBlobDetector**. The detector uses four main filters:

- **Area** (min_area, max_area):
  - Removes very small blobs (noise) and very large blobs (e.g. umbrellas or merged groups).
- **Circularity** (min_circularity):
  - Keeps blobs that are roughly circular or elliptical, as expected for heads and upper bodies.
- **Convexity** (min_convexity):
  - Prefers compact shapes, rejects very concave ones.
- **Inertia ratio** (min_inertia):
  - Controls how elongated a blob can be, balancing standing and sitting people against flat objects like towels.

SimpleBlobDetector returns a list of keypoints, one per accepted blob. The **detected count** (C_{det}) is the number of keypoints.

All these parameters are controlled by sliders during the interactive phase to find values that work well on several images.

---

4. Interactive Tuning and Batch Evaluation

The notebook uses ipywidgets mainly as a **parameter search tool**:

- The user selects a sample image from a dropdown.
- Sliders control gamma, clahe_clip, top_mask_percent, hsv_s_max, hsv_v_min, morph_size, and the blob filters (min_area, max_area, min_circularity, min_convexity, min_inertia).
- Each time a slider changes, comprehensive_analysis runs the full pipeline for that image and shows the detections and the per-image error (absolute error and error percentage).

This interactive step is used only to **find a good set of parameters**.

Once a configuration is chosen, a **separate loop cell** processes all images in the dataset with these fixed parameters.

From results, the notebook computes global performance metrics.

---

5. Performance Metrics

The evaluation is done at the **image level**. For each image (i):

- Ground-truth count.

- Detected count.
- Absolute error.
- Error percentage.

Over a set of (N) images, the main metric is **Mean Absolute Error (MAE)**:

$$\text{MAE} = \frac{1}{N}\sum_{i=1}^{N}\left|C_{gt}^{(i)} - C_{det}^{(i)}\right|.$$

We also report the **mean error percentage**, which averages the per-image error percentages. These two metrics describe:

- how many people the system misses or over-counts **on average**, and
- how large the error is relative to the real number of people.

In the written report, you can insert the numeric values obtained in your experiments (for example, the MAE in people per image and the average error percentage).

---

6. Design Choices, Limitations, and Conclusion

The design is intentionally simple and interpretable:

- Classical computer vision is enough for this small dataset and avoids the need for training.
- All parameters have a clear meaning and can be adjusted manually.
- The whole method runs in real time on a normal CPU inside Jupyter.

However, there are some limitations:

- **Occlusion:** in very crowded regions, several people can merge into a single blob, causing under-counting.
- **Lighting extremes:** very strong shadows or reflections can break the assumptions of the pipeline.
- **Similar shapes:** some objects (umbrellas, signs, structures) may produce blobs like people and generate false positives.

Despite these issues, the approach works well for **sparse to medium** beach crowds. It provides reasonable people counts with low computational cost and full transparency of each step.

In summary, the notebook implements a complete and understandable pipeline for beach crowd counting. It combines CLAHE, gamma correction, spatial and color masks, adaptive thresholding, and blob detection, with an interactive phase to select good parameters and a batch loop to measure performance over all images using MAE and error percentage.