Image Restoration using ROF Model: Analysis Report

Dimtri Bolt

May 4, 2025

1 Brief Explanation of Methods

1.1 ROF Discretization and Difference Scheme

The Rudin-Osher-Fatemi (ROF) model aims to restore an image u from a degraded observation f by minimizing the functional:

$$\mathcal{F}(u) = \int \sqrt{\epsilon^2 + |\nabla u|^2}\, dx\, dy + \frac{\lambda}{2} \int (u - f)^2\, dx\, dy$$

where λ is the fidelity parameter balancing data consistency and smoothing, and ϵ is a regularization parameter ensuring differentiability.

The corresponding Euler-Lagrange PDE is:

$$\frac{u - f}{\lambda} - \nabla \cdot \left(\frac{\nabla u}{\sqrt{\epsilon^2 + |\nabla u|^2}} \right) = 0$$

subject to Neumann boundary conditions ($\frac{\partial u}{\partial n} = 0$) on the image domain boundaries.

This implementation solves the PDE using an iterative fixed-point method based on its discretized form. The discretization employs:

- **Finite Differences:** Standard central differences are used implicitly through forward/backward schemes.
  - The gradient $\nabla u = (u_x, u_y)$ components are approximated using *forward differences*:
  $$(u_x)_{i,j} \approx u_{i,j+1} - u_{i,j}$$
  $$(u_y)_{i,j} \approx u_{i+1,j} - u_{i,j}$$
  - The divergence term $\nabla \cdot \mathbf{p}$ (where $\mathbf{p} = (p_x, p_y) = \frac{\nabla u}{\sqrt{\epsilon^2 + |\nabla u|^2}}$) is approximated using *backward differences*:
  $$(\nabla \cdot \mathbf{p})_{i,j} \approx (p_x)_{i,j} - (p_x)_{i,j-1} + (p_y)_{i,j} - (p_y)_{i-1,j}$$
- **Neumann Boundary Conditions:** These are handled by using symmetric padding (‘padarray(..., 'symmetric')‘) before computing spatial differences near the boundaries. This effectively mirrors the image values across the boundary, enforcing a zero gradient perpendicular to the edge.
- **Iterative Solver:** The discretized PDE is rearranged into a fixed-point iteration scheme:
  $$u^{(k+1)} = f - \lambda \cdot \nabla \cdot \left(\frac{\nabla u^{(k)}}{\sqrt{\epsilon^2 + |\nabla u^{(k)}|^2}} \right)$$
  where $u^{(k)}$ is the solution at iteration k. The iteration starts with $u^{(0)} = f$ and continues until the relative change between successive iterations ($||u^{(k+1)} - u^{(k)}||/||u^{(k)}||$) falls below a specified tolerance (`tol = 1e-4`) or a maximum number of iterations (`max_iter = 100`) is reached.

This scheme is implemented in the MATLAB function `smooth_image_rof.m`.

2 Interpretation of Results

2.1 MSD Plot Analysis

The plot 'Stacked MSD Surfaces for R, G1, G2, B Planes' (shown in Figure 1) displays the Mean Square Difference (MSD) between the original noisy color plane (f) and the ROF-smoothed version (u) as a function of the parameters λ and ϵ. The MSD is calculated as:

$$\text{MSD}(f, \lambda, \epsilon) = \sqrt{\frac{\sum_{i,j}(u_{i,j} - f_{i,j})^2}{H \cdot W}}$$

Each colored surface represents one of the four color planes (Red, Green1, Green2, Blue) extracted from the Bayer mosaic. The surfaces are stacked vertically with transparency to allow comparison.

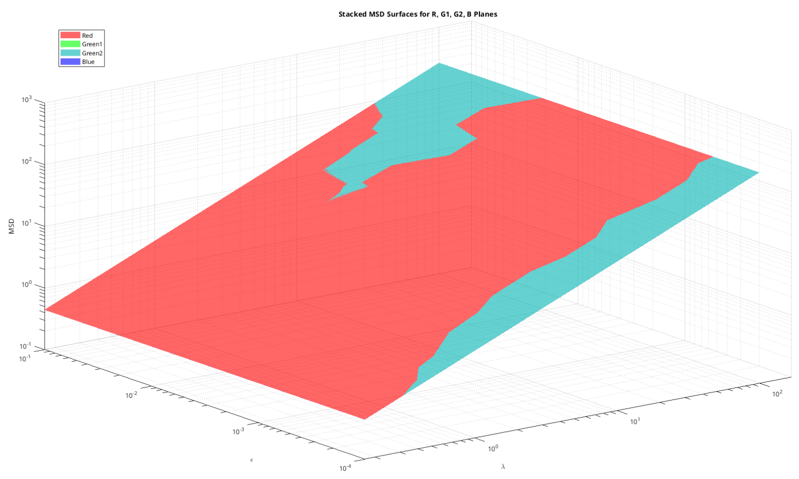

Figure 1: Stacked MSD Surfaces vs. λ and ϵ for R, G1, G2, B planes.

Dependence on λ: The parameter λ controls the weight of the data fidelity term $\frac{1}{2}\int(u-f)^2$.

- For *small* λ, the gradient term $\int \sqrt{\epsilon^2 + |\nabla u|^2}$ dominates. The model prioritizes smoothness, potentially oversmoothing the image (u becomes very different from f), leading to a *higher* MSD.
- For *large* λ, the fidelity term dominates. The model forces the solution u to be very close to the noisy input f. This results in less smoothing and a *lower* MSD, as the difference $u - f$ approaches zero.

Therefore, the MSD is expected to decrease as λ increases. The plot likely shows this trend, though calling it strictly "linear" might be an oversimplification; the relationship is generally monotonic decreasing but non-linear, especially when viewed on logarithmic scales. The rate of decrease diminishes as u gets very close to f.

Dependence on ϵ: The parameter ϵ regularizes the gradient term, essentially setting a threshold below which gradient magnitudes do not heavily influence the cost.

- For *very small* ϵ, the term $\sqrt{\epsilon^2 + |\nabla u|^2} \approx |\nabla u|$ (Total Variation). The model strongly penalizes any gradients, leading to significant smoothing and potentially piecewise constant results. This strong smoothing might increase the MSD if it removes actual image features along with noise.
- For *large* ϵ, the term $\sqrt{\epsilon^2 + |\nabla u|^2} \approx \epsilon + \frac{|\nabla u|^2}{2\epsilon} + \ldots$. The gradient's influence is down-weighted relative to the constant ϵ. The model allows steeper gradients with less penalty, resulting in less smoothing. As ϵ becomes very large, the first term's contribution to the functional becomes less dependent on ∇u, and the fidelity term (scaled by λ) dictates the solution more, potentially making u closer to f.

The observation that there appears to be "no dependence on ϵ" in the plot might be due to several factors:

1. The chosen range for ϵ might be too high, placing the analysis in a regime where MSD is less sensitive to ϵ.
2. The variation in MSD due to ϵ might be much smaller than the variation due to λ, making it less apparent on the combined plot, especially with logarithmic axes potentially compressing the visual effect.
3. There might be an interaction effect, where the influence of ϵ is only significant for certain ranges of λ.

Fundamentally, MSD *should* depend on ϵ, as ϵ directly influences the degree of smoothing applied, which in turn affects the difference $u - f$. A lack of apparent dependence warrants closer inspection of the parameter ranges and plot scales.

3 Discussion of Noise Differences Between Color Planes

3.1 Measuring Noise using ROF

In this context, the "noise" is interpreted as the high-frequency variations present in the observed image f that are removed by the ROF smoothing process to produce the restored image u. The ROF model, particularly for appropriate λ and ϵ, aims to separate the underlying structure (u) from the noise ($n \approx f - u$). The Mean Square Difference (MSD) serves as a quantitative measure of the "amount" of information removed during smoothing. By finding the parameters $(\lambda_{opt}, \epsilon_{opt})$ that minimize the MSD for a given plane, we find the ROF solution u_{opt} that is closest (in the mean square sense) to the original f. While not a direct measure of noise power in the traditional signal processing sense, the minimum MSD achieved provides a relative measure of the noise level removed by ROF across the different color planes. A lower minimum MSD suggests either less noise was present initially, or the noise structure was more amenable to removal by ROF smoothing.

3.2 Analysis of Noise Statistics (via MSD)

By examining the stacked surfaces in Figure 1, we can compare the MSD values across the R, G1, G2, and B planes.

- **Relative Noise Levels:** The plane whose surface sits consistently lower than the others, especially in the region of minimum MSD values, can be inferred to be the "least noisy" as interpreted by the ROF model. Conversely, the plane with the highest surface suggests it contains more noise (or features interpreted as noise by ROF). Based on typical sensor characteristics and the Bayer pattern, one might expect the Green channels (G1, G2) to potentially exhibit lower noise or different noise characteristics compared to Red and Blue, as there are twice as many green sensors, often contributing more to luminance detail. Visual inspection of the plot is required to confirm which planes yield lower minimum MSD values.
- **Bayer Pattern Speculation (Two Green Planes):** Digital camera sensors use a Color Filter Array (CFA), most commonly the Bayer pattern (RGGB), to capture color information with a single sensor layer. The human visual system is more sensitive to green light, which carries most luminance information. Having twice as many green pixels (G1, G2) as red or blue allows the camera to capture finer luminance detail and potentially reduce luminance noise through averaging during the demosaicing process (which reconstructs the full-color image). While G1 and G2 pixels capture the same color, their surrounding neighbors in the RGGB grid are different, which can lead to slight variations in their values after processing or potential differences in noise characteristics depending on sensor readout patterns or demosaicing algorithms, although ideally they should represent similar noise levels from the same (green) spectral band. Comparing the MSD surfaces for G1 and G2 can reveal if they exhibit significantly different noise levels in the raw data.

4 Software Structure

4.1 Software Components

The image restoration software consists of the following MATLAB script files:

- `basic_script.m`: Performs initial setup. Reads the raw image file (e.g., `DSC00099.ARW`), extracts metadata, and separates the Bayer mosaic data into four distinct color planes (R, G1, G2, B) stored in the variable `Iplanar`. It also includes visualization steps.
- `smooth_image_rof.m`: Implements the core ROF algorithm. Takes a single image plane (f) and parameters (λ, ϵ) as input and returns the smoothed image (u). Handles scalar or vector parameters, Neumann boundary conditions, GPU acceleration (if available), and parallel CPU processing (using `parfor` for vector parameters if no GPU).
- `calculate_msd.m`: Computes the Mean Square Difference (MSD) between an input image plane (f) and its ROF-smoothed version. It calls `smooth_image_rof` internally to get the smoothed image u for the given parameters (λ, ϵ) before calculating the MSD.
- `run_rof_analysis.m`: The main driver script that orchestrates the entire analysis. It calls `basic_script.m`, defines parameter ranges, manages the processing of each color plane (including chunking for GPU memory management), calls `calculate_msd` for parameter sweeps, collects results, and generates the final stacked MSD surface plot.

4.2 How to Launch the Software

To run the analysis:

1. Ensure all MATLAB script files (`run_rof_analysis.m`, `basic_script.m`, `smooth_image_rof.m`, `calculate_msd.m`) are in the MATLAB path or the current working directory.
2. Ensure the raw image file (`DSC00099.ARW`) is located in the path expected by `basic_script.m` (e.g., in a subdirectory named `images`).
3. Open MATLAB.
4. Run the main driver script by typing its name in the Command Window or by opening it in the Editor and pressing the 'Run' button:

```
>> run_rof_analysis
```

The script will then execute the full analysis and produce the MSD plot. Progress messages will be displayed in the Command Window.

4.3 Software Logic Flow

The execution proceeds as follows:

1. `run_rof_analysis.m` is executed.
2. It first calls `basic_script.m` to load the raw image data and extract the four color planes (R, G1, G2, B) into the variable `Iplanar`.
3. It defines the vectors of parameters `lambda_vec` and `epsilon_vec` over which the analysis will be performed.
4. It sets up parameters for processing the parameter grid in chunks to manage GPU memory (`combinations_per_chunk`, `num_chunks`).
5. It initializes a cell array (`msd_results`) to store the final MSD matrices for each plane.
6. It enters a loop iterating through the four color planes (j=1 to 4):
    - Extracts the current plane from `Iplanar` and converts it to `single` precision.
    - Enters a second loop iterating through the defined chunks:
        - Determines the subset of (`lambda`, `epsilon`) pairs for the current chunk.
        - Calls `calculate_msd` with the current plane and the parameter subset for this chunk.
        - Inside `calculate_msd`:

        * It calls `smooth_image_rof` with the same inputs.
        * `smooth_image_rof` performs the iterative ROF smoothing. If a GPU is available and parameters are vectors, it uses an internal batched GPU computation; otherwise, it uses `parfor` on the CPU for vector parameters.
        * `smooth_image_rof` returns the smoothed image(s) u.
    - `calculate_msd` computes the MSD between f and u for the chunk and returns the result (`msd_chunk`).
  - The main script (`run_rof_analysis.m`) carefully maps the results from `msd_chunk` back into the correct positions in the overall `msd_results{j}` matrix for the current plane.
  - (Optionally, GPU memory is reset between chunks).

7. After processing all chunks for all four planes, `run_rof_analysis.m` uses the completed `msd_results` matrices and `meshgrid/surf` to generate the final stacked 3D surface plot showing MSD vs. λ and ϵ for all planes.

5 GPU Memory Considerations

5.1 GPU Memory Issues Encountered

During the execution of the analysis with vector inputs for λ and ϵ on a machine equipped with a GPU, an "Out of memory on device" error was encountered. This occurred specifically within the `smooth_image_rof.m` function when processing multiple parameter combinations simultaneously using the GPU-accelerated code path.

The root cause lies in the strategy employed for GPU vectorization within `smooth_image_rof`. To process $K \times L$ parameter combinations efficiently, the input image f is replicated $K \times L$ times into a large 4D `gpuArray` (`f_big`). The subsequent fixed-point iteration requires numerous intermediate 4D arrays of the same large size (`[H, W, K, L]`) to be stored concurrently in GPU memory (e.g., `uk`, `up`, `ux`, `uy`, `mag`, `px`, `py`, `div_p`, `unew`). For the image size ($H \approx 2000, W \approx 3000$) and the chosen parameter grid size ($K = 15, L = 10$, total 150 combinations), the cumulative size of these temporary arrays significantly exceeded the available GPU RAM (e.g., 8.5 GB), leading to the memory allocation failure (specifically noted during a `padarray` call).

5.2 Resolution of GPU Memory Issues

The out-of-memory issue is resolved by implementing a **chunking** strategy within the main driver script, `run_rof_analysis.m`, rather than attempting to process the entire $K \times L$ parameter grid in a single call on the GPU.

The key variables controlling this strategy are:

- `num_total_combinations`: Calculated simply as $K \times L$, the total number of (λ, ϵ) pairs in the full grid.
- `combinations_per_chunk`: A user-defined parameter specifying the maximum number of parameter combinations to process in a single call to `calculate_msd` (and thus `smooth_image_rof`). This value is chosen small enough such that the peak memory usage for processing this many combinations stays within the available GPU RAM. A value of 15 was used as a starting point.
- `num_chunks`: Calculated as $\lceil$`num_total_combinations/combinations_per_chunk`$\rceil$. This determines the number of iterations required in the chunking loop to cover the entire parameter grid.

The `run_rof_analysis.m` script now includes an outer loop iterating `num_chunks` times. In each iteration, it identifies a small subset of (`lambda, epsilon`) pairs corresponding to the current chunk. It then calls `calculate_msd` with only these limited parameter vectors. Consequently, the internal GPU processing within `smooth_image_rof` only needs to handle a much smaller batch size (at most `combinations_per_chunk`), drastically reducing the peak GPU memory requirement for temporary arrays. The results from each chunk are then collected and carefully reassembled into the final full MSD matrix for each color plane. This approach effectively trades potentially longer overall execution time (due to multiple function calls and potentially less optimal GPU utilization per call) for the ability to complete the computation within the constrained GPU memory environment.