Image & Video Processing: Assignment 2

Kyla Kaplan
April 2024

Contents

1 Local Operations

1 Linear Motion Blur Filter

The Gaussian filter exhibits a "symmetrical decay" in all directions, meaning it decreases uniformly regardless of the axis along which it is measured. The exercise asks us to implement a function that computes an anisotropic Gaussian kernel for blurring the image along the angle of the direction of motion, with inputs of angle of motion (θ), σ_x and σ_y; then we should demonstrate on the input image `graz.png`.

```matlab
function [blurred, h] = anisotropic(img, theta, sigx, sigy)
    angle = deg2rad(theta);

    % calculate size of filer
    sigma = [sigx, sigy];
    filter_size = 4*sigma+1;
    filter_size = abs([cos(angle) sin(angle)]) .* filter_size(1) + abs([sin(angle)
        cos(angle)]) .* filter_size(2);

    % create grid and rotate ellipsoid
    [X, Y] = meshgrid(-filter_size:filter_size, -filter_size:filter_size);
    X_rotated = -X * cos(angle) + Y * sin(angle);
    Y_rotated = X * sin(angle) + Y * cos(angle);

    % create anisotropic gf
    h = exp(-(X_rotated.^2 / (2 * sigx^2) + Y_rotated.^2 / (2 * sigy^2)));
    h = h / sum(h, 'all'); % normalize to add up to 1

    % apply with imfilter
    blurred = imfilter(img, h, 'conv', 'replicate');
end
```

The anisotropic function above take an input image, angle (in degrees), and two sigmas, x and y respectively; firstly, it calculates the size of the filter based on the inputted σ's (using the formulated example as shown in class), creates a "grid", and rotates it. Then, it is assigned to the kernel as per the formula:

$$h = \exp\left(-\frac{X^2_{\text{rotated}}}{2\sigma_x^2} - \frac{Y^2_{\text{rotated}}}{2\sigma_y^2}\right)$$

Followed by a normalization of the kernel, and lastly, it is applied to the image. The function thus returns the blurred image, and the used kernel.

I did not use the `conv2` function towards the end, and opted for `imfilter`, as I found the prior to leave an odd darkened-border artifact.

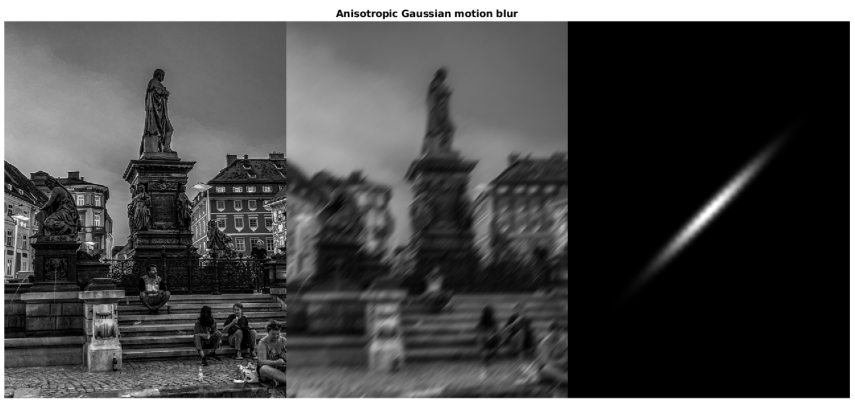

Figure 1: Original image (left), motion blur (middle), applied kernel (right)

After some experimenting, I found an optimal σ_x and σ_y to be:

```
sigx = 7;     % Standard deviation along the x
sigy = 0.7;   % Standard deviation along the y
```

2 Iterative Filtering

Iterative filtering involves approximating a larger kernel filter by repeatedly applying a smaller kernel filter. With Gaussian blur, this technique uses smaller filters multiple times to mimic the effect of a larger Gaussian kernel. Each application of the filter alters pixel values based on spatial proximity.

Iterative bilateral filtering may introduce aliasing artifacts due to increased smoothing with each application of the filter, resulting in a loss of detail. The bilateral filter's edge-preserving nature can lead to sharper edges but also potentially more pronounced aliasing artifacts.

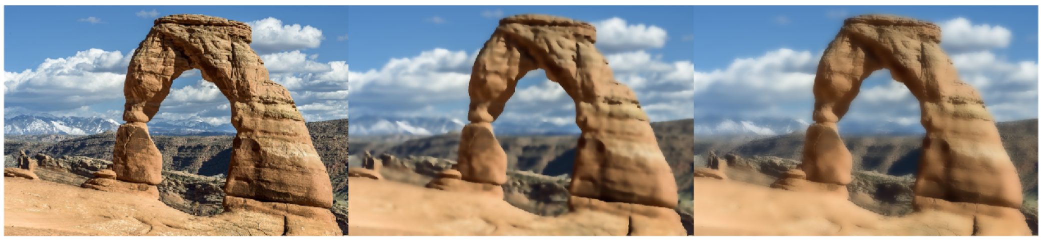

Figure 2: Original (left), Several smaller bilateral filters (middle), Large bilteral filter (right)

The fully implemented function can be found in ex2.m. The code demonstrates how to achieve iterative filtering using bilateral filtering, applying smaller kernels multiple times to approximate the effect of a larger kernel. I set the σ space and range for the smaller bilateral filter to the following, as well as the number of iterations:

```
num_iterations = 3;
sigma_space = 3;
sigma_range = 0.1;
```

And for the larger bilateral turns to be:

```
big_sigma_space = sigma_space * num_iterations; % = 9
big_sigma_range = sigma_range * num_iterations; % = 0.3
filtered_image_big_kernel = imbilatfilt(I, big_sigma_range, big_sigma_space);
```

The function utilizes the in-built function imbilatfilt, and as mentioned, I have set the parameters DegreeOfSmoothing and SpatialSigma accordingly.

```
filtered_image_small_kernel = I;
for i = 1:num_iterations
    filtered_image_small_kernel = imbilatfilt(filtered_image_small_kernel, sigma_range,
        sigma_space);
end
```

As can be seen in Figure 2, with delicate_arch.jpg used, applying multiple smaller filters (meaning with small spatial or range σ) seems to preserve features more and blur the total image less, when compared to using a single larger bilateral filter. This seems to be because a larger σ value causes the filter to consider a wider range of pixel values and distances, making it less sensitive to small intensity differences or "weaker" features. Perhaps tweaking the σ space (which controls the spatial extent of the filter) and σ range (defines the intensity range) values could achieve a much closer result, and no significant aliasing artifacts are seen.

3 Image Stylization

The desired image shown in the assignment, that features a "cartoonish" filter has a few noticeable effects that need to be mimicked in order to achieve the same result; firstly, the blurring of the underlying image is of note, and secondly and most importantly, the bolded thick lines that create the aforementioned "cartoonish" features. Figre 3 below depicts the results I achieved.

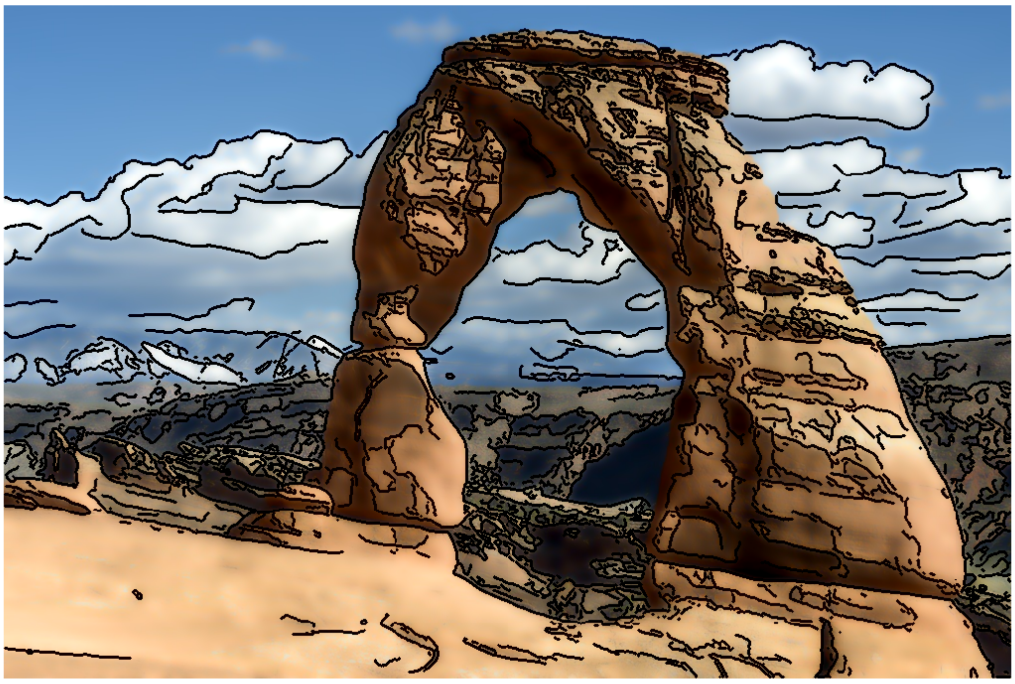

Figure 3: Resulting stylized image

The full code can be found in `ColorHandling.m`. Prior to everything the image was initially converted to gray-scale. Firstly (similarly to the previous assignment), I applied a bilateral filter in order to blur the image while retaining as much outline/edge information. Following that, the edges are extracted from the blurred image (as seen in Figure 4), using an "edge detection" function `edge`. This function utilizes a `'Canny'` algorithm (it seems to look better than the `'Prewitt'` one). The result is a binary image where pixels corresponding to edges are set to 1 (white), and non-edge pixels are set to 0 (black) [Fig. 4]. By utilizing the `strel` function in order to create a structural element for the proceeding morphological operation, and the `imdilate` function, which in this case is responsible for morphologically dilating regions of high intensities (in this case white lines) in `I_edge`.

Both intermediate steps are shown below. The final image is then re-colored with the given driver code and is shown above.

```
...
sigma_space = 5;
sigma_range = 0.1;
I_smooth = imbilatfilt(I_gray, sigma_range, sigma_space);

I_edge = edge(I_smooth, 'Canny');

structuring_element = strel('disk', 1);
I_edge_dilated = imdilate(I_edge, structuring_element);
...
```

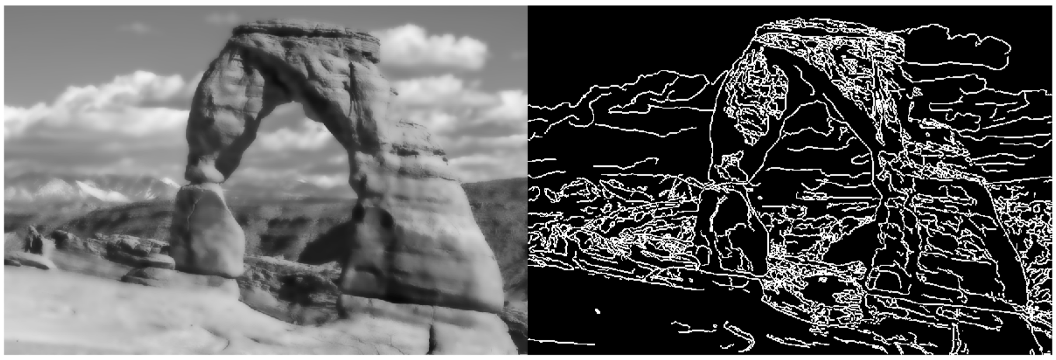

Figure 4: Detected edges

As a note, in my resulting I observed a minor ”halo effect” (especially visible in the stone closest to the camera), but it seems to be minor.