Vector Valued Image Regularization with PDE

- Aneesh Garg (180050007)
- Dhananjay Singh (180050029)

Introduction

Regularization Algorithms

- What are they?
- Where are they used?
- How do they work?

Image Regularization

- **Functional minimization:** Euler-Lagrange equations

$$\underbrace{\min}_{I:\Omega\to R^n} \int_{\Omega} \phi(N(I))d\Omega$$

- **Divergence expression:** Diffusion of pixel values from high to low concentration

$$\frac{\partial I}{\partial t} = div(D\nabla I_i)$$

- **Oriented Laplacians:** Image smoothing in eigenvector directions weighted by corresponding eigenvalue.

$$\frac{\partial I}{\partial t} = c_1 I_{uu} + c_2 I_{vv}$$

Structure Tensor

$$\mathbf{G} = \sum_{i=1}^{n} \nabla I_i \nabla I_i^T = \sum_{i=1}^{n} \left(\begin{bmatrix} I_x \\ I_y \end{bmatrix} \begin{bmatrix} I_x & I_y \end{bmatrix} \right)$$

- Nonlinear regularization PDE's to vector-valued images
- Eigenvalues and Eigenvectors
- Corners and Edges

Geometric Meaning of Oriented Laplacians

$$\frac{\partial I_i}{\partial t} = c_1 \, I_{i\xi\xi} + c_2 \, I_{i\eta\eta} = \mathbf{trace}\,(\mathbf{T}\mathbf{H}_i) \qquad (i = 1..n),$$

- 2D image regularization as juxtaposition of oriented laplacians
- H is the Hessian matrix and T is a 2x2 Tensor matrix
- Solution for above PDE is :

$$I_{i_{(t)}} = I_{i_{(t=0)}} \; * \; G^{(\mathbf{T},t)} \qquad (i = 1..n),$$

Tensor Field

$$\mathbf{T} = f_-\left(\sqrt{\lambda_+^* + \lambda_-^*}\right)\theta_-^*{\theta_-^*}^T + f_+\left(\sqrt{\lambda_+^* + \lambda_-^*}\right)\theta_+^*{\theta_+^*}^T.$$

- $\boldsymbol{\lambda}$ and $\boldsymbol{\theta}$ are defined to be the spectral elements of a Gaussian smoothed version of the structure tensor G
- This allows us to retrieve a more coherent vector geometry and giving a better approximation of the vector discontinuities directions

$$f_+(s) = \frac{1}{1+s^2} \quad \text{and} \quad f_-(s) = \frac{1}{\sqrt{1+s^2}}.$$

Oriented Gaussian Kernel

$$G^{(\mathbf{T},t)}(\mathbf{x}) = \frac{1}{4\pi t} \exp\left(-\frac{\mathbf{x}^T \mathbf{T}^{-1} \mathbf{x}}{4t}\right) \qquad \text{with} \qquad \mathbf{x} = (x \quad y)^T.$$

- This is the G obtained in the previous solution
- When T is not constant (which is generally the case), the previous PDE becomes nonlinear
- And can be viewed as the application of temporally and spatially varying local masks G over the image I

Implementation (Method)

Creation of Binary Mask

```
%% Creating Masks
mask_glasses_new = zeros(size(mask_glasses,1),size(mask_glasses,2));
for i = 1:size(mask_glasses,1)
    for j = 1:size(mask_glasses,2)
        if(mask_glasses(i,j,1)==255 && mask_glasses(i,j,2)==255 && mask_glasses(i,j,3)==255)
            mask_glasses_new(i,j)=1;
        end
    end
end
mask_glasses_new = double(mask_glasses_new);
```

Image Gradients

```
[imx(:,:,1), imy(:,:,1)] = imgradientxy(final(:,:,1),'sobel');
[imx(:,:,2), imy(:,:,2)] = imgradientxy(final(:,:,2),'sobel');
[imx(:,:,3), imy(:,:,3)] = imgradientxy(final(:,:,3),'sobel');
```

$$G_x = \begin{bmatrix} +1 & 0 & -1 \\ +2 & 0 & -2 \\ +1 & 0 & -1 \end{bmatrix} * A \quad \text{and} \quad G_y = \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} * A$$

- Computed gradients across x and y direction using sobel operator
- This will be used to compute structure tensor at each point

Evaluating Structure Tensor

```
gauss = fspecial('gaussian',3,1);
```

Evaluated structure tensor at the hole and then convolved it with gaussian mask to get the gaussian smoothed structure tensor

```
imrxx = imx(i,j,1)*imx(i,j,1);
imrxy = imx(i,j,1)*imy(i,j,1);
imryy = imy(i,j,1)*imy(i,j,1);
imbxx = imx(i,j,2)*imx(i,j,2);
imbxy = imx(i,j,2)*imy(i,j,2);
imbyy = imy(i,j,2)*imy(i,j,2);
imgxx = imx(i,j,3)*imx(i,j,3);
imgxy = imx(i,j,3)*imy(i,j,3);
imgyy = imy(i,j,3)*imy(i,j,3);

ixx = (imrxx+imbxx+imgxx);
ixy = (imrxy+imbxy+imgxy);
iyy = (imryy+imbyy+imgyy);

G = [ixx ixy; ixy iyy];
G = imfilter(G,gauss);
```

Evaluating Tensor field

Evaluated Eigenvalues and eigenvectors of structure tensor and created tensor field

```
[V, D] = eig(G);
[D,I] = sort(diag(D));
D = flip(D);
D = diag(D);
V =  V(:, flip(I));
thieta_plus = V(:,1);
thieta_minus = V(:,2);
lambda_plus = D(1,1);
lambda_minus = D(2,2);
T = (1/sqrt(1+(lambda_plus+lambda_minus)))*(thieta_minus*thieta_minus');
T = T + (1/(1+(lambda_plus+lambda_minus)))*(thieta_plus*thieta_plus');
T = inv(T);
```

Applying $G^{(T,t)}(x)$ locally

- Took weighted average of neighbouring intensities.
- Weights being determined by $G^{(T,t)}(x)$
- Thus, computed the new intensity at the hole at i,j

```
for z = 1:3
    temp=0;
    temp2=0;
    for k = max(i-w,1):min(i+w,size(X,1))
        for k1 = max(j-w,1):min(j+w,size(X,2))
            %                                   if(k~=i && k1~=j)
            x = [k-i;k1-j];
            temp2 = temp2 + (exp(-1*(x'*T*x)/(4*t))*final(k,k1,z));
            if(final(k,k1,z)~=0)
                temp = temp + (exp(-1*(x'*T*x)/(4*t)));
            end
            %                                   end
        end
    end
    if(temp>0)
        final(i,j,z)=temp2/temp;
    end
end
```

Gaussian Smoothening

Smoothening the edges created by completely filling the missing parts through inpainting

```
for i = 2:size(final,1)-1
    for j = 2:size(final,2)-1
        if(mask_greenparrot_new(i,j+1)||mask_greenparrot_new(i+1,j+1)||mask_greenp
            w = floor(size(filter,1)/2);

            for z = 1:3
                temp=0;
                temp2=0;
                for k = max(i-w,1):min(i+w,size(final,1))
                    for k1 = max(j-w,1):min(j+w,size(final,2))
                        if(k~=i && k1~=j)
                        temp2 = temp2 + filter(k-i+w+1,k1-j+w+1)*final(k,k1,z);
                        if(final(k,k1,z)~=0)
                            temp = temp + filter(k-i+w+1,k1-j+w+1);
                        end
                        end
                    end
                end
                if(temp>0)
                    final2(i,j,z)=temp2/temp;
                end
            end
        end
```

Applications

Inpainting - 1




Parameters : Neighbourhood = 5x5, t=100, maxiter = 1

Inpainting - 2

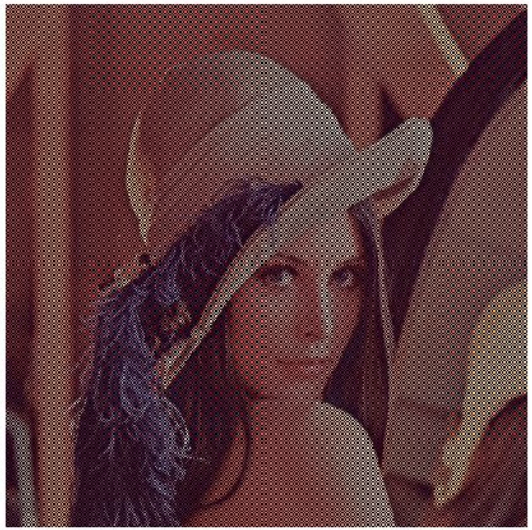



Parameters : Neighbourhood = 11x11, t=100, maxiter = 1

Inpainting - 3

Parameters : Neighbourhood = 11x11, t=100, maxiter = 1

Image Restoration - 1



Parameters : neighbourhood = 11x11, t=1000, maxiter = 1
SSIM of the final inpainted image and the original image = 0.969360

Image Restoration - 2

Parameters : neighbourhood = 11x11, t=1000, maxiter=1
SSIM of the final inpainted image and the original image = 0.922200

Observations

- As the width of the mask increases, the effectiveness of our algorithm decreases. This is evident from the fact that as the width increases there is a greater loss of information which makes it hard to reconstruct.
- Our results are very sensitive to the neighbourhood size, i.e., they increase/decrease very rapidly with change in neighbourhood size.
- Our results do depend on the number of iterations, but they start to converge as the number of iterations reaches a high value.

Observations

- Our results very slightly depend on the time parameter, i.e., for observing a significant change in the results, a large change in time parameter is required.
- This algorithm yields very good results in a continuous region, but at the edges we do not observe such good results. This is due to the fact, that it is hard for the algorithm to distinguish between the two sides across the edge and ends up assigning a value somewhere in between.

Thank You !