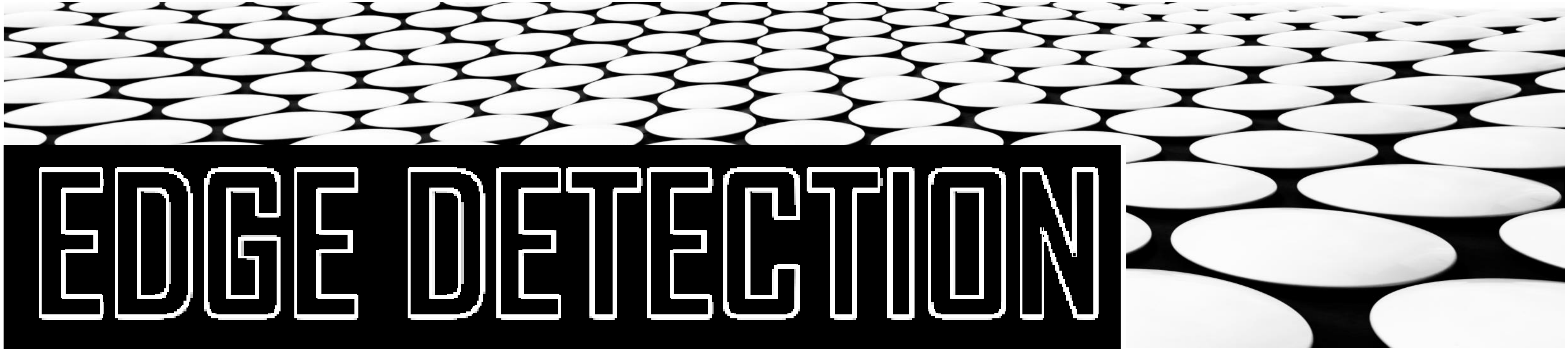
INTELLIGENT SYSTEMS FOR
PATTERN RECOGNITION
A.Y. 2020/2021

MIDTERM 1
ASSIGNMENT 6

ELIA PICCOLI

[HTTPS://GITHUB.COM/ELIAPICCOLI/ISPR-PROJECTS](https://github.com/eliapiccoli/ISPR-projects)



EDGE DETECTION CODE

```
% read the image and take its gray representation
T = rgb2gray(imread('./dataset/2_13_s.bmp'));

% ------- Filtering
% Reduce possible noise in the image
% Trade-off between edge strength and noise reduction
GF = getfilter('gaussian');
TGF = convolution(T, GF);

% ------- Enhancement
% Determine changes in intensity in the neighbourhood of a point
% Compute the gradient magnitude, using the gradients along X-Y
SFX = getfilter('sobel', 'X');
SFY = getfilter('sobel', 'Y');
TSFX = convolution(T, SFX);
TSFY = convolution(T, SFY);
MTSF = uint8(sqrt(double((TSFX.^2)+(TSFY.^2))));

% ------- Detection
% Filter out only strong edges
RMTSF = rescale(MTSF,'InputMin',0,'InputMax',255);
threshold = 180/255;
TRMTSF = RMTSF;
TRMTSF(RMTSF < threshold) = 0;
```

GETFILTER CODE

```
function [F] = getfilter(varargin)
%GETFILTER Return the requested filter
%   Give the name of the filter return the matrix of its representation.
%   This version produces filters of defualt size [3x3].
%   Certain parameters only apply to certain filters.
%Input:
% - FN : string, filter name
% - A : string, for gradient filters specify the axis (X, Y)
% - S : flaot, for gaussian filter specify standard deviation
% - N : integer, specify filter dimension
%Output:
% - F : NxN matrix representing the filter

    % input checking
    [FN, A, S, N] = parseparameters(varargin{:});
    % Assign to F the correct filter
    switch FN
        case 'roberts'
            if A == 'X'; F = [1 0 ; 0 -1]; else; F = [0 -1 ; 1 0]; end
        case 'sobel'
            if A == 'X'; F = [1 0 -1; 2 0 -2; 1 0 -1]; else; F = [1 2 1; 0 0 0; -1 -2 -1]; end
        case 'prewitt'
            if A == 'X'; F = [-1 0 1; -1 0 1; -1 0 1]; else; F = [1 1 1; 0 0 0; -1 -1 -1]; end
        case 'average'
            F = ones(N)/(N*N);
        case 'gaussian'
            r = N;c = N;
            [row, col] = meshgrid(-(r-1)/2:(r-1)/2, -(c-1)/2:(c-1)/2);
            F = exp(-(row.^2+col.^2)/(2*S^2));
            F = F./sum(F(:));
        case 'log' % Laplacian of Gaussian
            % gaussian filter
            r = N;c = N;
            [row, col] = meshgrid(-(r-1)/2:(r-1)/2, -(c-1)/2:(c-1)/2);
            F = exp(-(row.^2+col.^2)/(2*S^2));
            F = F./sum(F(:));
            % laplacian filter
            F1 = F.*((row.*row + col.*col - 2*S^2)/(S^4));
            F = F1 - sum(F1(:))/prod(r*c);
    end
end
```

CONVOLUTION CODE

```
function [C] = convolution(I, K, P)
%CONVOLUTION Compute 2D-Convolution between an image I and kernel K
% Computes convoltion betweem the Image and the Kernel. If provided the
% algorithm uses Matlab ParPool to compute the result.
%Input:
% - I : matrix [ N x M ], Image
% - K : matrix [ N x N ], Kernel
% - P : boolean, parallel computation
%Output:
% - C : Convolution Matrix

% cast I to int32 to avoid data type problems during computation
I = cast(I, 'int32');

% Create result matrix
[IR,IC] = size(I);
[KR,KC] = size(K);
C = zeros(1, IR*IC);

% Compute padding of original image (replicate)
Xpad = floor(KR/2);
Ypad = floor(KC/2);
upad = repmat(I(1, :), Xpad, 1);
bpad = repmat(I(end, :), Xpad, 1);
PI = [upad ; I ; bpad];
lpad = repmat(PI(:, 1), 1, Ypad);
rpad = repmat(PI(:, end), 1, Ypad);
PI = [lpad PI rpad];
[~, PIC] = size(PI);
```

```
% linearlize the matrixes for better parrallel computation
LPI = reshape(PI', 1, []);
LK = reshape(K', 1, []);
% reverse the kernel
LK = flip(LK);

% Compute convolution
if P
    parfor index=0:IR*IC-1
        C(index+1) = computeconv(index, IC, PIC, LPI, KR, KC, LK, Xpad, Ypad);
    end
else
    for index=0:IR*IC-1
        C(index+1) = computeconv(index, IC, PIC, LPI, KR, KC, LK, Xpad, Ypad);
    end
end

% un-linearize C
C = reshape(C, [IC, IR])';
```

```
function[sum] = computeconv(index, IC, PIC, LPI, KR, KC, LK, XP, YP)
    x = floor(index/IC);
    y = mod(index, IC);
    sum = 0;
    for i=0:KR-1
        for j=0:KC-1
            ImX = x + i - floor(KR/2);
            ImY = y + j - floor(KC/2);
            sum = sum + LK(i*KR+j+1)*LPI((ImX+XP)*PIC+ImY+YP+1);
        end
    end
end
```

TREE

ORIGINAL

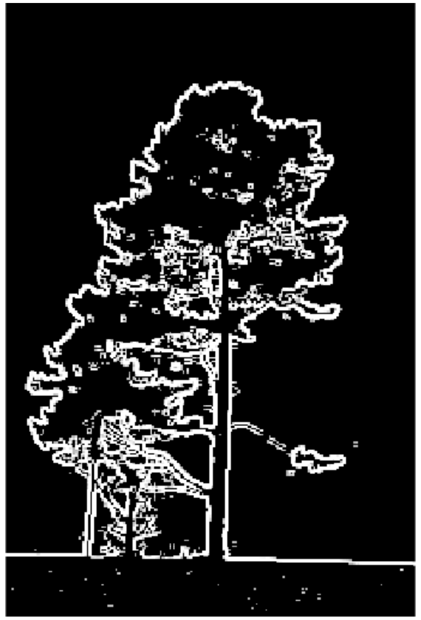
SOBEL GRADIENT X

SOBEL GRADIENT Y

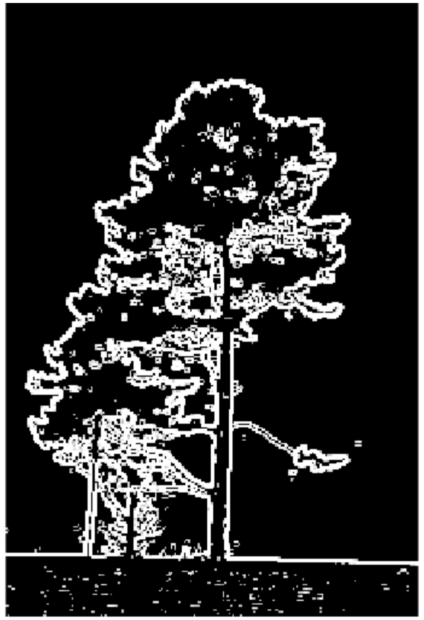
SOBEL GRADIENT MAGNITUDE

ROBERTS

PREWITT



SOBEL

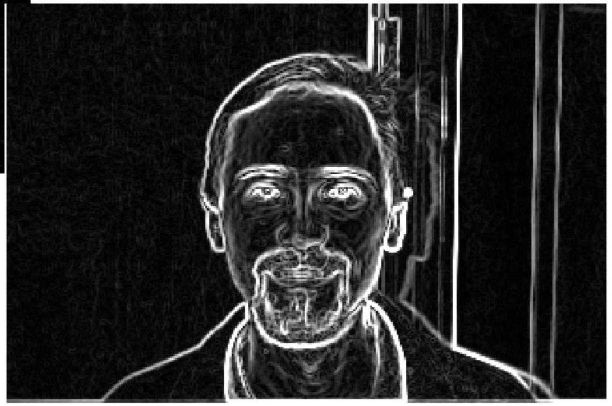


FACE

ORIGINAL

SOBEL GRADIENT X

SOBEL GRADIENT Y



SOBEL GRADIENT MAGNITUDE

ROBERTS

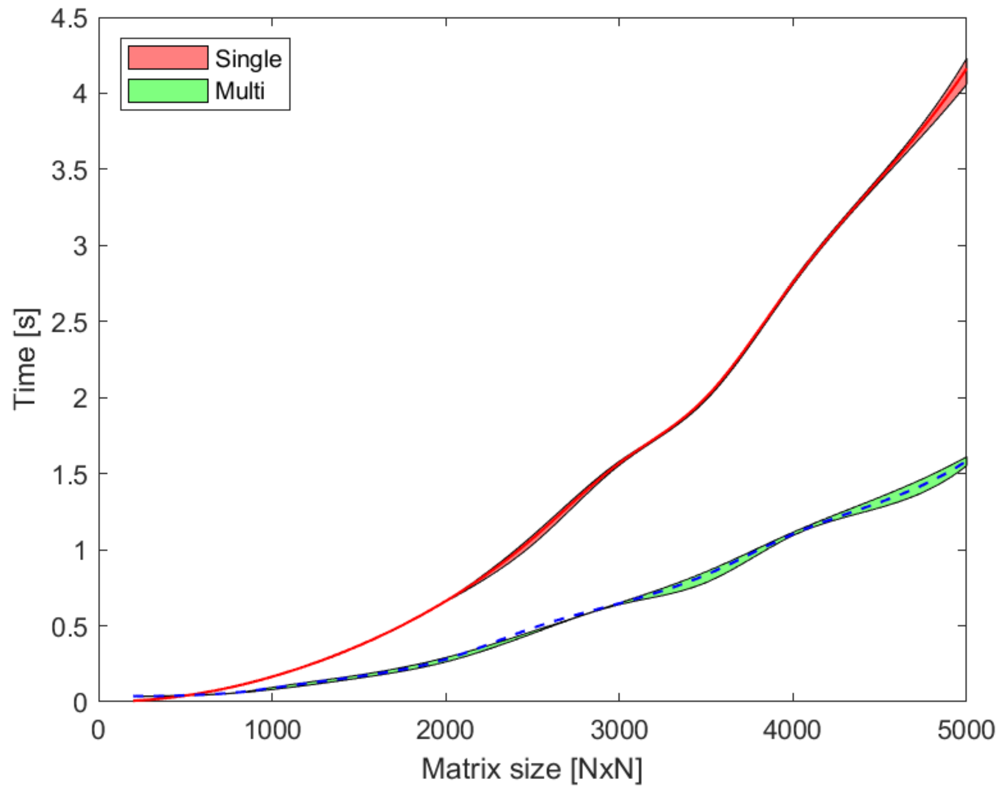
PREWITT

SOBEL

CAN WE IMPROVE?

COMPUTATIONAL POINT OF VIEW

Use parallel computation to compute
convolution of different cells at the same time.



MODEL POINT OF VIEW

We have seen only filters that compute the gradient, so they consider only the first derivative information.

What if, we try to exploit second derivative information?
Laplacian of Gaussian