Implementation of 2D Optimal Edge Detector Recursively Implemented.

ABSTRACT

To implement the recursive 2D Optimal Edge Detector by using the recursive filtering mechanism reducing the computational effort and obtaining the first derivatives which can be used further for edge detection.

-Supervised by: Professor Rachid Deriche

-Submitted by: Cyril Naves

References:
[1] Rachid Deriche "Fast Algorithms for Low-Level Vision" in IEEE Transaction on Pattern Analysis and Machine Intelligence Vol12, Jan 1990

[2] Rachid Deriche "Using Canny's criteria to derive a recursively implemented Optimal Edge Detector" in International Journal of Computer Vision 1987

Abstract: To implement a recursive filtering structure that reduces the computational effort for smoothing and then perform the first and second derivatives of an image. These operations are done with fixed number of multiplication and additions per output point independent of size of neighborhood considered. Here we apply a recursive filter which is first derived so as to avoid the two steps of smoothing and then find the derviatives.

General Survey of the Reference Paper [1] and [2] for implementing Edges from first derivatives recursively:

1) In [1] the recursive filtering is explained as a convolution operation relating the input sequence to the output sequence.
   $Y(i) = \sum_{k=0}^{n-1} h(k)\, x(i-k)$.
2) We know that to get the first order derivatives we need to convolve each point of the image.
3) We use the following 2D Smoothing and Derivatives Operators [1] to get the first order derivatives in a recursive implementation.
   2D Smoothing Filter is given by:
   $$SS(m,n) = k(\alpha|m| + 1)\, e^{-\alpha|m|} k(\alpha|n| + 1) e^{-\alpha|n|}$$
   Then by derivative rule of convolution we get the x and y derivatives by convolving with the following operators.
   $$SS_x(m,n) = k' m e^{-\alpha|m|} k(\alpha|n| + 1) e^{-\alpha|n|}$$

$$SS_y(m, n) = k(\alpha|m| + 1)e^{-\alpha|m|}k'ne^{-\alpha|n|}$$

4) Here K and K' are given by:

$$k = \frac{(1 - e^{-\alpha})^2}{1 + 2\alpha e^{-\alpha} - e^{-2\alpha}}.$$

$$k = -\frac{(1 - e^{-\alpha})^2}{e^{-\alpha}}.$$

5) We follow the recursive filtering scheme here by the following equations:

a) We convolve image x(m,n) along x -direction:

$$y_1(m, n) = a_1x(m, n) + a_2x(m, n - 1) + b_1y_1(m, n - 1) + b_2y_1(m, n - 2)$$
$$\text{for } n = 1, \cdots, N \text{ and } m = 1, \cdots, M$$

With boundary conditions:

$$x(m, 0) = 0,$$
$$y_1(m, 0) = y_1(m, -1) = 0 \quad \text{for } m = 1, \cdots, M$$

b) Then obtain the following:

$$y_2(m, n) = a_3x(m, n + 1) + a_4x(m, n + 2) + b_1y_2(m, n + 1) + b_2y_2(m, n + 2)$$
$$\text{for } n = N, \cdots, 1 \text{ and for } m = 1, \cdots, M$$

With boundary conditions:

$$x(m, N + 1) = x(m, N + 2) = 0 \text{ and } y_2(m, N + 1) = y_2(m, N + 2) = 0 \quad \text{for } m = 1, \cdots, M$$

c) Finally calculate r(m,n):

$$r(m, n) = c_1(y_1(m, n) + y_2(m, n))$$
$$\text{for } n = 1, \cdots, N \text{ and for } m = 1, \cdots, M$$

d) Then apply to the above result in the vertical direction y(m,n)

$$y_1(m, n) = a_5r(m, n) + a_6r(m - 1, n) + b_1y_1(m - 1, n) + b_2y_1(m - 2, n)$$
$$\text{for } m = 1, \cdots, M \text{ and for } n = 1, \cdots, N$$

With boundary conditions:

$$r(0, n) = 0,$$
$$y_1(0, n) = y_1(-1, n) = 0 \quad \text{for } n = 1, \cdots, N$$

e) Calculate y2(m,n) :

$$y_2(m, n) = a_7 r(m + 1, n) + a_8 r(m + 2, n) + b_1 y_2(m + 1, n) + b_2 y_2(m + 2, n)$$

$$\text{for } m = M, \cdots, 1 \text{ and for } n = 1, \cdots, N$$

With boundary conditions:

$$r(M + 1, n) = r(M + 2, n) = 0 \text{ and}$$

$$y_2(M + 1, n) = y_2(M + 2, n) = 0 \quad \text{for } n = 1, \cdots, N$$

$$y(m, n) = c_2(y_1(m, n) + y_2(m, n))$$

$$\text{for } n = 1, \cdots, N \text{ and for } m = 1, \cdots, M.$$

f) Here the coefficients are given by:

$$a_1 = 0;\ a_2 = 1;\ a_3 = -1;\ a_4 = 0;\ c_1 = -(1 - e^{-\alpha})^2;$$

$$a_5 = k;\ a_6 = ke^{-\alpha}(\alpha - 1);\ a_7 = ke^{-\alpha}(\alpha + 1);$$

$$a_8 = -ke^{-2\alpha};\ c_2 = 1;$$

Similarly, for the y derivatives can be obtained by swapping the following: a_i with a_{i+4} and c_1 with c_2.

g) Coefficients for b1 and b2 given by:

$$b_1 = 2e^{-\alpha};\ b_2 = -e^{-2\alpha}.$$

6) After the convolution operation recursively along the x- co-ordinates and y co-ordinates, where we derive the first derivatives, the local orientation of an edge is taken to be direction of tangent along contour in image plane by the first directional directives of x and y.
7) Gradient Magnitude is calculated by:
   For edge=Sqrt(x^2+y^2) for each point
   Also for Angle=arctg(x/y) for each point
8) Gradient magnitude is then non -maxima suppressed in exact gradient direction and thresholded where in our case we choose to be 30.
9) Edges are characterized as local maxima of gradient magnitude along eta direction.

10)After the local maxima is detected we keep higher than the threshold as output points for the First order derivative approach.

Approach implemented using Matlab Script:
(Appropriate Comments specified in the attached script)
Algorithm:

1) Input Image

```
input = imread('castle.jpg');
```

2) Convert Image to GrayScale

```
input_gray=rgb2gray(input);
[rows, columns ,depth] = size(input_gray);
x=input_gray;
```

3) Declare coefficients

Value of alpha

```
alpha =1;
```

4) Value of K:

```
knum = (1-exp(-alpha))^2;
kden= 1+(2*alpha*exp(-alpha))-exp(-2*alpha);
k = knum/kden;
```

5) Value of other coefficients:

```
a1=0; a5=k;
a2=1; a6= k* exp(-alpha)*(alpha-1);
a3=-1; a7=k*exp(-alpha)*(alpha+1);
a4=0; a8 = -k*exp(-2*alpha);
b1=2*exp(-alpha);b2=-exp(-2*alpha);
c1 = -((1-exp(-alpha))^2); c2=1;
```

6) Important Recursive Sturcture:
Start of Recursive Filtering Scheme

Derivative of X

Along Horizontal

Step1: Calculate y1

Boundary Conditions
x(m,0) = 0;
yl(m, 0), y1(m, -1) = 0;

```
y1=x;
for n=1:columns
   for m=1:rows
      if n==1
         y1(m,n)=a1*x(m,n)+a2*0+b1*0+b2*0;
      elseif n==2
         y1(m,n)=a1*x(m,n)+a2*x(m,n-1)+b1*y1(m,n-1)+b2*0;
      else
         y1(m,n)=a1*x(m,n)+a2*x(m,n-1)+b1*y1(m,n-1)+b2*y1(m,n-2);
      end
   end
end
```

Step2: Calculate y2
Boundary Conditions
x(m,N+1),x(m,N+2)= 0
y2(m,N+1),y2(m,N+2) = 0

```
y2=x;
for n=columns:-1:n
   for m=1:rows
      if n==columns
         y2(m,n)=a3*0+a4*0+b1*0+b2*0;
      else

y2(m,n)=a3*x(m,n+1)+a4*x(m,n+2)+b1*y2(m,n+1)+b2*y2(m,n+2);
      end
   end
end
%Step3:Calculate R(m,n)
r=x;
for n=1:columns
   for m=1:rows
```

```
    r(m,n) =c1*(y1(m,n)+y2(m,n));
  end
end
```

Along Vertical
Step4: Calculate y3
Boundary Conditions
r(0, n) = 0;
y1(0, n) ,y1(-1,n) = 0;

```
y3=x;
for m=1:rows
  for n=1:columns
    if m==1
      y3(m,n)=a5*r(m,n)+a6*0+b1*0+b2*0;
    elseif m==2
      y3(m,n)=a5*r(m,n)+a6*r(m-1,n)+b1*y1(m-1,n)+b2*0;
    else
      y3(m,n)=a5*r(m,n)+a6*r(m-1,n)+b1*y1(m-1,n)+b2*y1(m-2,n);
    end
  end
end
```

Step5: Calculate y4
Boundary Conditions
r(M + 1, n) = r(M + 2, n) = 0 and
y2(M + 1, n) = y2(M + 2, n) = 0

```
y4=x;
for m=rows:-1:m
  for n=1:columns
    if m==rows
      y4(m,n)=a7*0+a8*0+b1*0+b2*0;
    else

y4(m,n)=a7*r(m+1,n)+a8*r(m+2,n)+b1*y2(m+1,n)+b2*y2(m+2,n);
    end
  end
end
```

Step5: Calculate Xfinal

```
xfinal=x;
for n=1:columns
   for m=1:rows
      xfinal(m,n)=c2*(y3(m,n)+y4(m,n));
   end
end
```

Similarly repeat this approach for getting y derviatives by reassigning value of other coefficients with swapping ai=ai+4 , c1=c2

7) Calculate Gradient Magnitude with calculated x and y derivatives

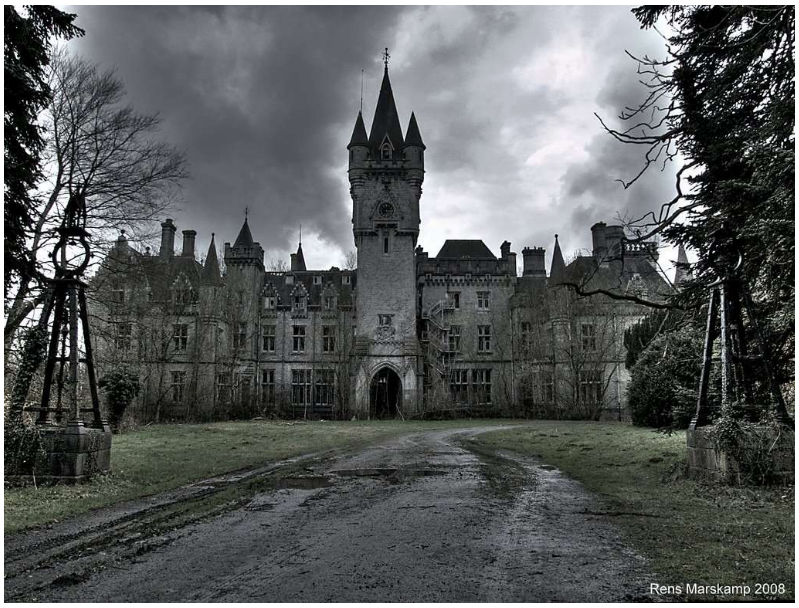
```
xfinal=double(xfinal);
yfinal =double(yfinal);
im_edge=double(x);
angle = double(x);

for m=1:rows
   for n=1:columns
      im_edge(m,n)= ((xfinal(m,n)^2+yfinal(m,n)^2)).^(0.5);
      angle(m,n)=atan2(xfinal(m,n),yfinal(m,n));
   end
end
```

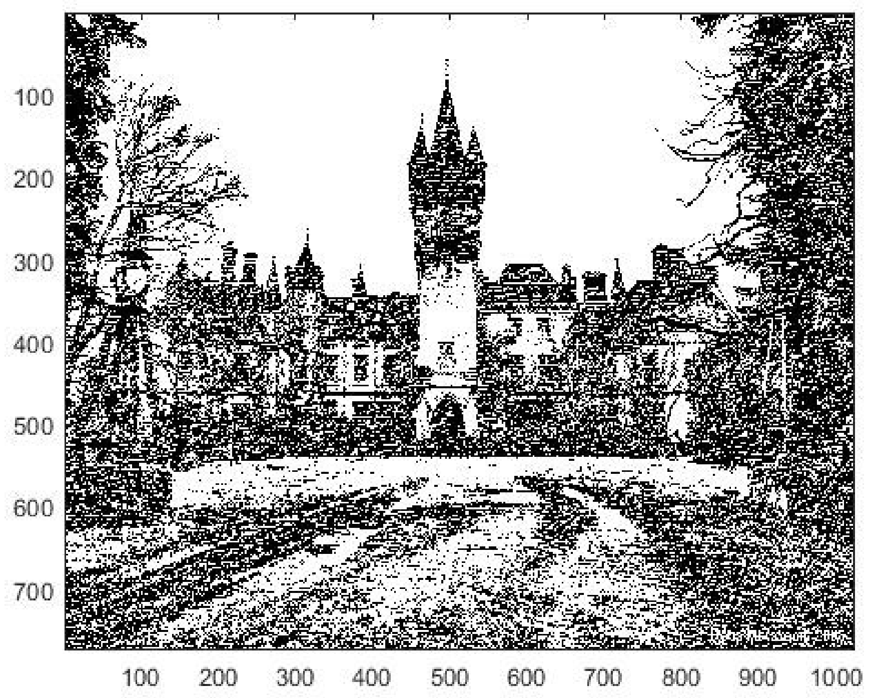
8) Calculate the Value of Non – Maxima Suppression in normal direction of edge and keep the value, otherwise set to 1
9) Then threshold of an image higher than the value 30 is then output into the final edge of the image.

Result and Output:
Here we apply the above recursive edge detector to the image:

10) **Original image:**



Recursively Edge Detected Image:



<u>Result and Observation:</u>

1) Thus through the use of recursive filtering as implemented in [1] we are able to do edge detection with less computational complexity with fixed set of multiplication and addition operation for calculating the derivatives of x and y.
2) Also here ideal value of alpha chosen is 1.0 , since values of alpha at 0.5 is unable to effectively detect the edge and for a larger value of 1.5, edges are not very clear.
3) In our case for the castle image , alpha value of 1 for recursive operator and the threshold of 30 for getting the output of the derivatives gives us the ideal edges for the images in a recursive way.