**Abc Image analysis using edge detection**

There are number of edge detection operators available, each designed to be sensitive to certain type of edges. The quality of edge detection can be measured from several criteria objectively. Some criteria are proposed in terms of mathematical measurement, some of them are based on application and implementation requirements .In all five cases a quantitative evaluation of performance requires use of images where the true edges are known.

* **Good detection**: There should be minimum number of false edges. Usually, edges are detected after a threshold operation. The high threshold will lead to less false edges, but it also reduces the number of true edges detected.
* **Noise sensitivity**: The algorithm can detect edges in certain acceptable noise environments.   Good localization: The edge location must be reported as close as possible to the correct possible position, i.e. edge localization accuracy.
* **Orientation Sensitivity**: The operator not only detects edge magnitude, but it also detects edge orientation correctly. Orientation can be used in post processing to connect edge segments , reject noise and suppress non-maximum edge magnitude
* **Speed and efficiency**: The algorithm should be fast enough to be usable in an image processing system. An algorithm that allows recursive implementation or separately processing can greatly improve efficiency.

**Techniques of Edge Detection:**

**Robert operator:**

It is the gradient operator. The simple 2\*2 Roberts operators were one of the earliest methods employed to detect edges. The Roberts operator is implemented using two convolution masks/kernels, each designed to respond maximally to edges running at ±45º to the pixel grid, which return the image x-derivative and y derivative, Gx and Gy respectively. This method is not preferred for today’s technology as it is highly sensitive to noise and not compatible with all the elements.

**Sobel Operator:**

The Sobel operator performs a 2-D spatial gradient measurement on an image. It uses a pair of 3×3 convolution masks, one estimating the gradient in the x-direction (columns) and the other estimating the gradient in the y-direction (rows). These kernels are designed to respond maximally to edges running vertically and horizontally relative to the pixel grid, one kernel for each of the two perpendicular orientations. The kernels can be applied separately to the input image, to produce separate measurements of the gradient component in each orientation (Gx and Gy). Errors due to effects of noise are reduced by local averaging within the neighborhood of the mask.

**The Prewitt filter:**

The Prewitt filter is very similar to Sobel operator. The 3x3 total convolution masks is used to detect gradient in the X, Y directions .Prewitt filter is a fast method for edge detection The Prewitt/Sobel kernels are generally preferred to the Roberts approach because the gradient is not shifted by half a pixel in both directions and extension to larger sizes (for filter neighbor hoods greater than 3\*3) is not readily possible with the Roberts operators. The key difference between the Sobel and Prewitt operators is that the Sobel kernel implements differentiation in one direction and (approximate) Gaussian averaging in the other. . It is only suitable for well-contrasted noiseless images. The advantage of this is that it smoothes the edge region, reducing the likelihood that noisy or isolated pixels will dominate the filter response.

In the following code, The Prewitt filter method is implemented in which an image is being taken from a camera and edge detection is applied onto it. The name of the image is “abc.jpg”

function varargout = abc(varargin)

% Begin initialization code

gui\_Singleton = 1;

gui\_State = struct('gui\_Name', mfilename, ...

'gui\_Singleton', gui\_Singleton, ...

'gui\_OpeningFcn', @abc\_OpeningFcn, ...

'gui\_OutputFcn', @abc\_OutputFcn, ...

'gui\_LayoutFcn', [] , ...

'gui\_Callback', []);

if nargin && ischar(varargin{1})

gui\_State.gui\_Callback = str2func(varargin{1});

end

if nargout

[varargout{1:nargout}] = gui\_mainfcn(gui\_State, varargin{:});

else

gui\_mainfcn(gui\_State, varargin{:});

end

% End initialization code

% --- Executes just before abc is made visible.

function abc\_OpeningFcn(hObject, eventdata, handles, varargin)

data1=imread('abc\_pic.jpg');

axes(handles.axes2);

imshow(data1);

clear data1;

handles.output = hObject;

% Update handles structure

guidata(hObject, handles);

varargout{1} = handles.output;

% --- Executes on button press in start.

function start\_Callback(hObject, eventdata, handles)

vid=videoinput('winvideo',2,'YUY2\_640X480');

set(vid,'TriggerRepeat',Inf);

vid.returnedcolorspace='rgb';

vid.FrameGrabInterval=2;

flag=1;

while (flag==1)

data=getsnapshot(vid);

diff=im2bw(data,0.60);

diff=bwareaopen(diff,100);

sum=0;

axes(handles.axes1);

diff=imerode(diff,strel('disk',12));

diff=imerode(diff,strel('line',3,45));

bw=bwlabel(diff,8);

stats=regionprops(bw,'Area','BoundingBox','Centroid');

imshow(data);

hold on

for object=1:length(stats)

if(stats(object).Area>1300&&stats(object).Area<6500)

bb=stats(object).BoundingBox;

bc=stats(object).Centroid;

rectangle('Position',bb,'EdgeColor','Y','LineWidth',2);

x1=bc(1);

y1=bc(2);

plot(x1,y1,'o');

sum=sum+stats(object).Area;

% display(stats(object).Area);

end

end

hold off

density=sum/768;

ncars=round(density/3.2);

set(handles.text3,'String',num2str(density));

set(handles.text6,'String',num2str(ncars));

red=0; y=2; grn=0;

if(ncars>=0&&ncars<2)

red=7; grn=3;

end

if(ncars>=2&&ncars<5)

red=5; grn=4;

end

if(ncars>=5&&ncars<9)

red=3; grn=7;

end

if(ncars>9)

red=1; grn=10;

end

set(handles.text10,'BackgroundColor','Red');

while(red>0)

set(handles.edit1,'String',num2str(red));

red=red-1;

pause(1);

end

red=2;

set(handles.text10,'BackgroundColor','Yellow');

while(red>0)

set(handles.edit1,'String',num2str(red));

red=red-1;

pause(1);

end

set(handles.text10,'BackgroundColor','Green');

while(grn>0)

set(handles.edit1,'String',num2str(grn));

grn=grn-1;

pause(1);

end

stx=get(handles.text4,'String');

if strcmp(stx,'Approximate Cars')

flag=1;

else

flag=2;

end

if(flag==2)

close all;

clear all;

flag=2;

end

end

function stop\_Callback(hObject, eventdata, handles)

set(handles.text4,'String','Stopped');

disp('END');

function edit1\_Callback(hObject, eventdata, handles)

function edit1\_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))

set(hObject,'BackgroundColor','white');

end