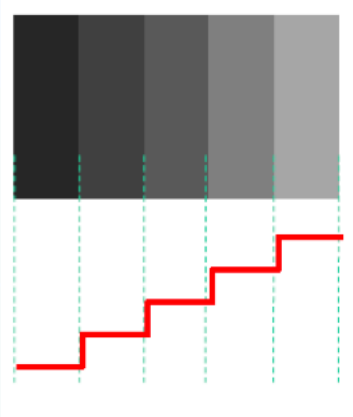
Edge
Assignment 6

Gwenaelle Cunha Sergio
ABR Lab – BEP
KNU 2014.1

Edge

- Edge: Part of the image that has a sharp **gradient** (change in intensity/contrast)

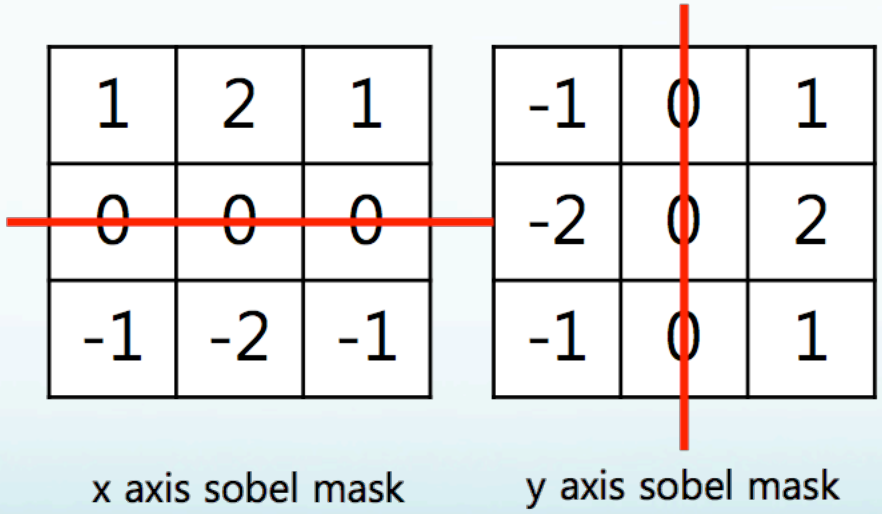


- Edge detection: used to **extract information** about the image (location of objects in the image, size, shape, image sharpening and enhancement).

- **Applications**: enhancement of noisy images (such as satellite images, x-rays and medical images); text detection; mapping of roads; video surveillance; and remote sensing images.

Sobel

- Edge Detection Method
- Uses the following masks:



Sobel - Code

```
int kernel_size = 3;
int radius = 1, kernelArea = kernel_size*kernel_size;
int pixelValue, kerneli, kernelj;

kernel = Mat::ones(kernel_size, kernel_size, CV_32F);
kernel.at<float>(0,0) = 1; kernel.at<float>(0,1) = 2; kernel.at<float>(0,2) = 1;
kernel.at<float>(1,0) = 0; kernel.at<float>(1,1) = 0; kernel.at<float>(1,2) = 1;
kernel.at<float>(2,0) = -1; kernel.at<float>(2,1) = -2; kernel.at<float>(2,2) = -1;

for (int i = 0; i < mat.rows; i++){
    for (int j = 0; j < mat.cols; j++){
    pixelValue = 0, kerneli = 0;
    for (int ki = i-radius; ki <= i+radius; ki++){
        kernelj = 0;
        for (int kj = j-radius; kj <= j+radius; kj++){
            if ((ki < 0 || kj < 0) || (ki >= mat.rows || kj >= mat.cols)) {
                pixelValue += 0;
            } else {
                pixelValue += (int) kernel.at<float>(kerneli,kernelj)*(int)mat.at<uchar>(ki,kj);
            }
            kernelj++;
        }
        kerneli++;
    }
    filter3x3X.at<uchar>(i,j) = abs(pixelValue/kernelArea);
    }
}
cv::normalize(filter3x3X, filter3x3X, 0, 255, CV_MINMAX);
```

Sobel - Result

original

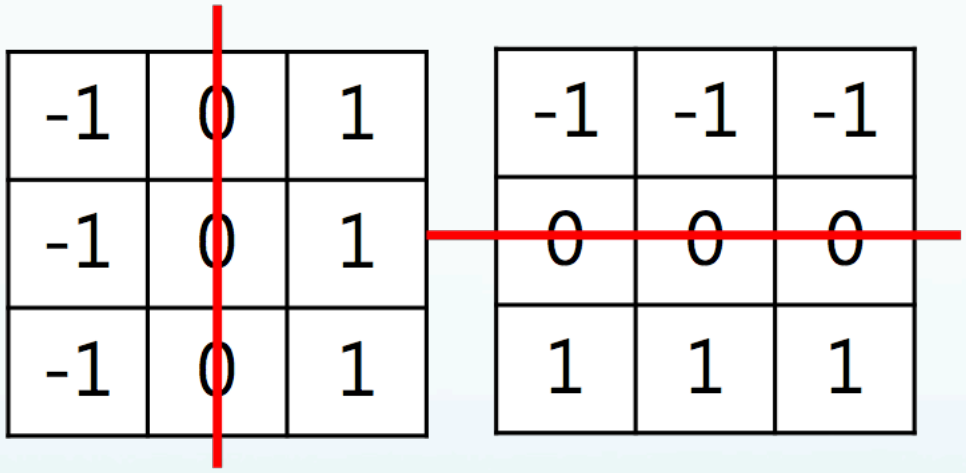
x+y

x

y

Prewitt

- Like Sobel, but uses the following masks:



y axis prewitt mask

x axis prewitt mask

Prewitt - Code

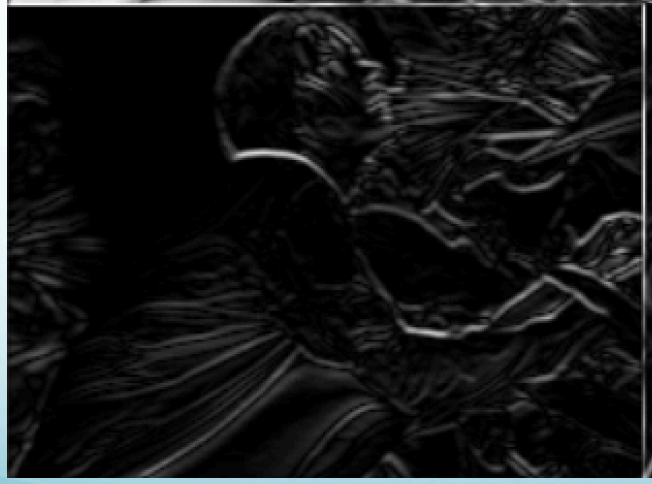
```
int kernel_size = 3;
int radius = 1, kernelArea = kernel_size*kernel_size;
int pixelValue, kerneli, kernelj;

kernel = Mat::ones(kernel_size, kernel_size, CV_32F);
kernel.at<float>(0,0) = -1; kernel.at<float>(0,1) = -1; kernel.at<float>(0,2) = -1;
kernel.at<float>(1,0) = 0; kernel.at<float>(1,1) = 0; kernel.at<float>(1,2) = 0;
kernel.at<float>(2,0) = 1; kernel.at<float>(2,1) = 1; kernel.at<float>(2,2) = 1;

for (int i = 0; i < mat.rows; i++){
    for (int j = 0; j < mat.cols; j++){
    pixelValue = 0, kerneli = 0;
    for (int ki = i-radius; ki <= i+radius; ki++){
        kernelj = 0;
        for (int kj = j-radius; kj <= j+radius; kj++){
            if ((ki < 0 || kj < 0) || (ki >= mat.rows || kj >= mat.cols)) {
                pixelValue += 0;
            } else {
                pixelValue += (int) kernel.at<float>(kerneli,kernelj)*(int)mat.at<uchar>(ki,kj);
            }
            kernelj++;
        }
        kerneli++;
    }
    filter3x3X.at<uchar>(i,j) = abs(pixelValue/kernelArea);
    }
}
cv::normalize(filter3x3X, filter3x3X, 0, 255, CV_MINMAX);
```

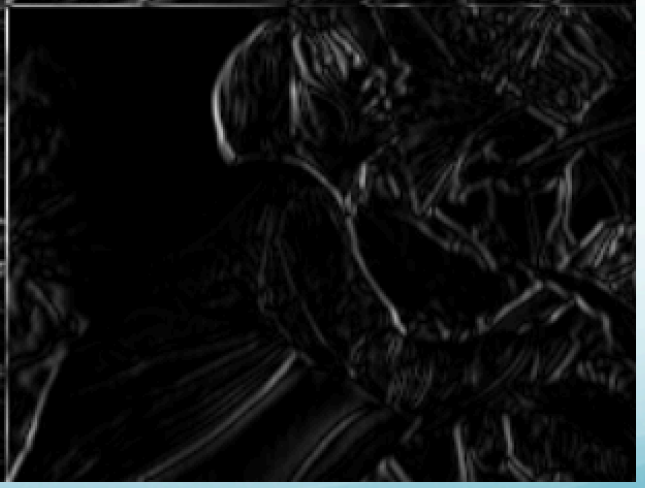
Prewitt - Result

original

x+y

x

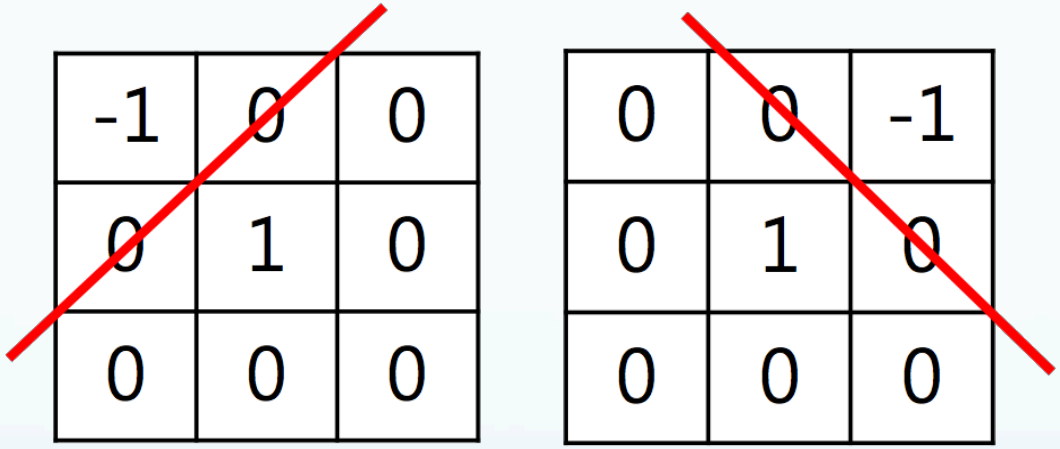




y

Roberts

- Faster than Sobel and Prewitt
- Small mask



- Pixel being calculated can't be average
- Sensitive to noises

Roberts - Code

```
int kernel_size = 3;
int radius = 1, kernelArea = kernel_size*kernel_size;
int pixelValue, kerneli, kernelj;

kernel = Mat::ones(kernel_size, kernel_size, CV_32F);
kernel.at<float>(0,0) = -1; kernel.at<float>(0,1) = 0; kernel.at<float>(0,2) = 0;
kernel.at<float>(1,0) = 0; kernel.at<float>(1,1) = 1; kernel.at<float>(1,2) = 0;
kernel.at<float>(2,0) = 0; kernel.at<float>(2,1) = 0; kernel.at<float>(2,2) = 0;

for (int i = 0; i < mat.rows; i++){
    for (int j = 0; j < mat.cols; j++){
    pixelValue = 0, kerneli = 0;
    for (int ki = i-radius; ki <= i+radius; ki++){
        kernelj = 0;
        for (int kj = j-radius; kj <= j+radius; kj++){
            if ((ki < 0 || kj < 0) || (ki >= mat.rows || kj >= mat.cols)) {
                pixelValue += 0;
            } else {
                pixelValue += (int) kernel.at<float>(kerneli,kernelj)*(int)mat.at<uchar>(ki,kj);
            }
            kernelj++;
        }
        kerneli++;
    }
    filter3x3X.at<uchar>(i,j) = abs(pixelValue/kernelArea);
    }
}
cv::normalize(filter3x3X, filter3x3X, 0, 255, CV_MINMAX);
```

Roberts - Result

original

x+y

x

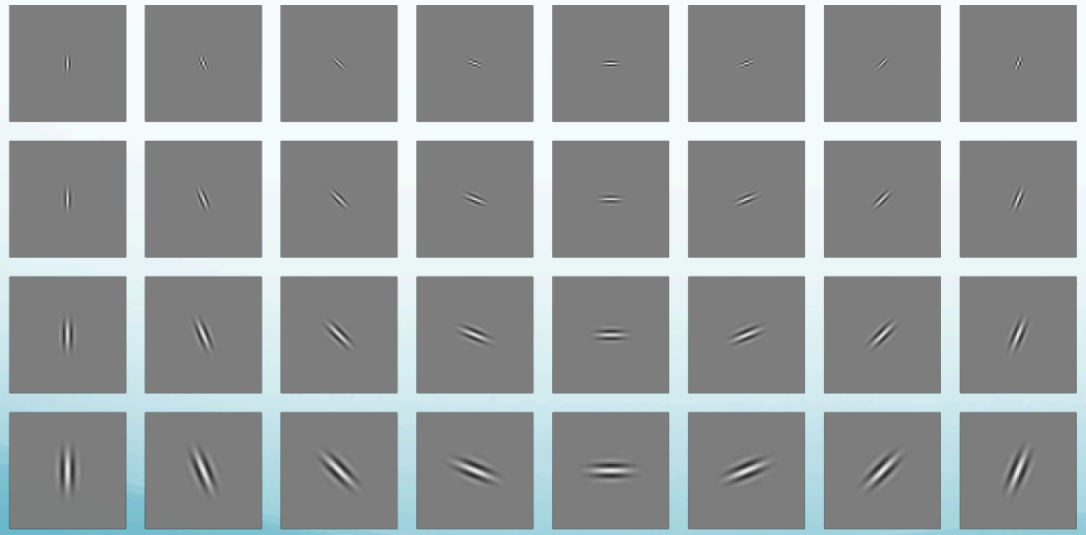
y

Orientation

- Detects the orientation of the edges in the image

-1	0	1
-2	0	2
-1	0	1

90^0 orientation



1	2	1
0	0	0
-1	-2	-1

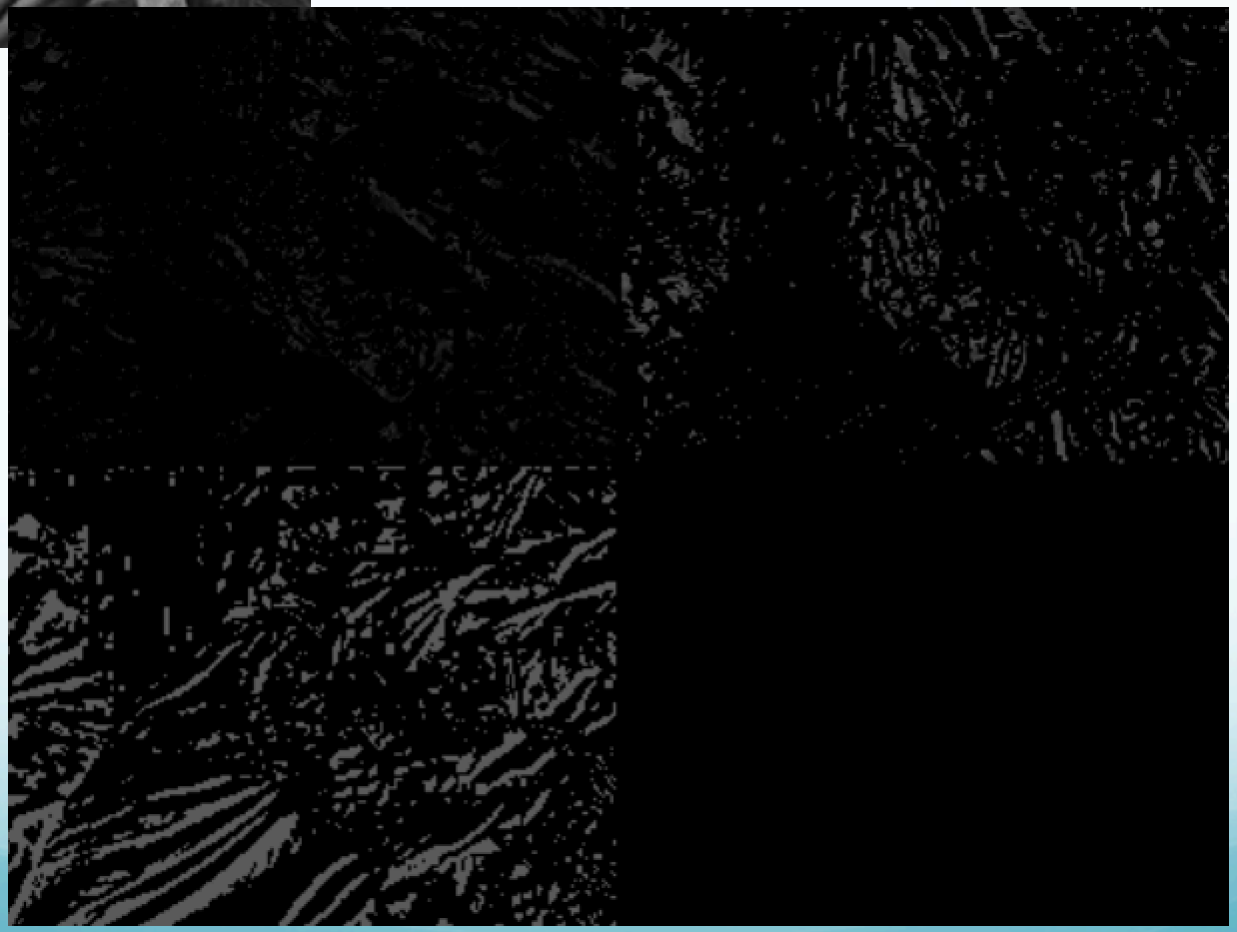
180^0 orientation

Orientation - Code

```
//Detect edge on the x-axis
kernel_size = 3;
kernel = Mat::ones(kernel_size, kernel_size, CV_32F);
kernel.at<float>(0,0) = 1; kernel.at<float>(0,1) = 2; kernel.at<float>(0,2) = 1;
kernel.at<float>(1,0) = 0; kernel.at<float>(1,1) = 0; kernel.at<float>(1,2) = 1;
kernel.at<float>(2,0) = -1; kernel.at<float>(2,1) = -2; kernel.at<float>(2,2) = -1;
filter2D(mat, edgeX, ddepth, kernel, anchor, delta, BORDER_DEFAULT);

//Detect edge on the y-axis
kernel = Mat::ones(kernel_size, kernel_size, CV_32F);
kernel.at<float>(0,0) = -1; kernel.at<float>(0,1) = 0; kernel.at<float>(0,2) = 1;
kernel.at<float>(1,0) = -2; kernel.at<float>(1,1) = 0; kernel.at<float>(1,2) = 2;
kernel.at<float>(2,0) = -1; kernel.at<float>(2,1) = 0; kernel.at<float>(2,2) = 1;
filter2D(mat, edgeY, ddepth, kernel, anchor, delta, BORDER_DEFAULT);

//Check, by using arctan, the orientation of the edge
for (int i = 0; i < mat.rows; i++){
    for (int j = 0; j < mat.cols; j++){
        uchar val = atan2((double) edgeY.at<uchar>(i,j), (double) edgeX.at<uchar>(i,j))*180 / CV_PI;
        if (val < 45) edge0.at<uchar>(I,j) = val;
        else if (val < 90) edge45.at<uchar>(I,j) = val;
        else if (val < 135) edge90.at<uchar>(I,j) = val;
        else edge135.at<uchar>(I,j) = val;
    }
}
```

Orientation - Result

original



$\theta < 45^{o}$

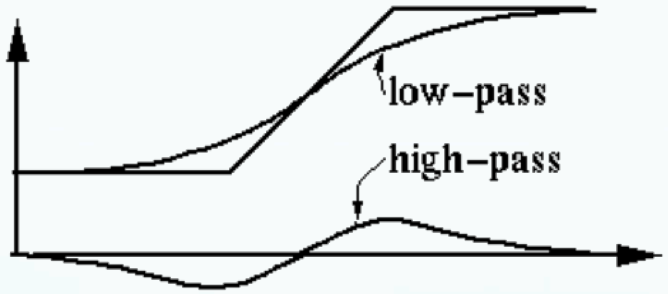
$\theta < 90^{o}$

$\theta < 135^{o}$

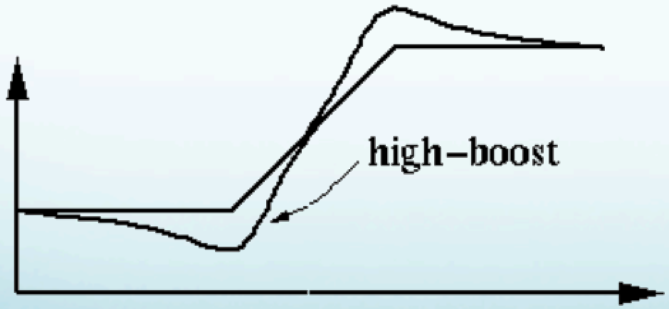
$\theta < 180^{o}$

Highboost-spatial

- High-pass filter with a boost, like the low-pass one



$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 4+c & -1 \\ 0 & -1 & 0 \end{bmatrix}$$



$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8+c & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

Highboost-spatial - Code

```
kernel_size = 3;
int radius = 1, kernelArea = kernel_size*kernel_size;
int pixelValue, kerneli, kernelj;

kernel = Mat::zeros(kernel_size, kernel_size, CV_32F);
kernel.at<float>(0,1) = -1; kernel.at<float>(1,0) = -1; kernel.at<float>(1,2) = -1;
kernel.at<float>(2,1) = -1; kernel.at<float>(1,1) = 4+c;

for (int i = 0; i < mat.rows; i++){
    for (int j = 0; j < mat.cols; j++){
        pixelValue = 0, kerneli = 0;
        for (int ki = i-radius; ki <= i+radius; ki++){
            kernelj = 0;
            for (int kj = j-radius; kj <= j+radius; kj++){
                if ((ki < 0 || kj < 0) || (ki >= mat.rows || kj >= mat.cols)) {
                    pixelValue += 0;
                } else {
                    pixelValue += (int)kernel.at<float>(kerneli,kernelj) * (int)mat.at<uchar>(ki,kj);
                }
                kernelj++;
            }
            kerneli++;
        }
        edge.at<uchar>(i,j) = abs(pixelValue/kernelArea);
    }
}
cv::normalize(edge, edge, 0, 255, CV_MINMAX);
```

Highboost-spatial - Result

original mask 1 mask 2