Image Analysis

Edge Detection

K. Buza, Lars Schmidt-Thieme

Information Systems and Machine Learning Lab (ISMLL)
Institute of Economics and Information Systems
& Institute of Computer Science
University of Hildesheim
http://www.ismll.uni-hildesheim.de

Edge Detection

<u>**What is Edge Detection?**</u>

Gradient for Edge Detection

Convolution

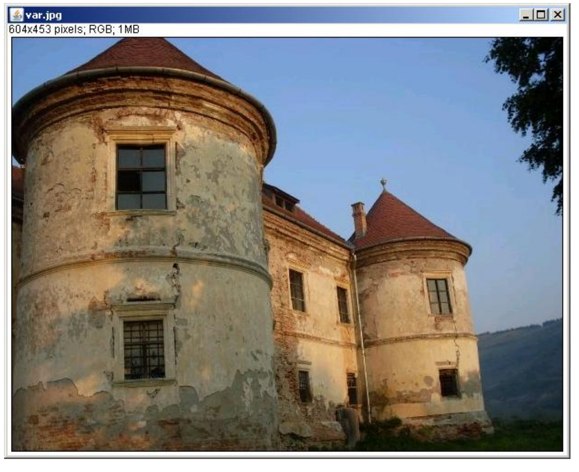
Marr-Hildreth

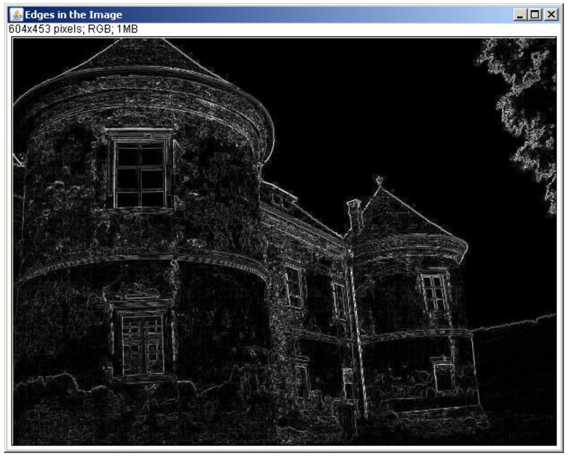
Canny

Hough Transformation

What is Edge Detection?

- “Low level“ edge detection
  - Which pixels belong to an edge?
  - What is the (local) orientation of those pixels?
- “High level” edge detection
  - Characterize edges in the picture
  - For example: lines by their offset and slope





Edge Detection

What is Edge Detection?

Gradient for Edge Detection

Convolution

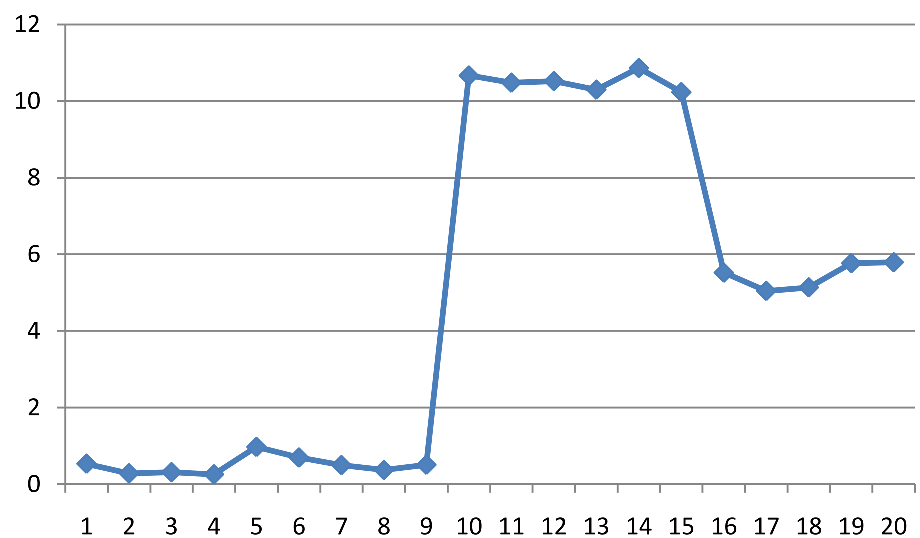
Marr-Hildreth

Canny

Hough Transformation

Gradient for Edge Detection

- What is an edge?
  - Significant change in intensity
- If we are given a function, how can we detect significant changes, i.e. regions where the function is “steep”?



Gradient for Edge Detection

- Derivation of two-dimensional function
  - One can derive the function along a chosen dimension (variable): $\frac{\partial f(x,y)}{\partial x}$ $\frac{\partial f(x,y)}{\partial y}$
  - Gradient: $\nabla f(x,y) = \left(\frac{\partial f(x,y)}{\partial x}, \frac{\partial f(x,y)}{\partial y} \right)$
  - The magnitude of the gradient:

$$G_{mag} = \sqrt{\left(\frac{\partial f(x,y)}{\partial x} \right)^2 + \left(\frac{\partial f(x,y)}{\partial y} \right)^2}$$

Gradient for Edge Detection

- Approximations for discrete functions (pictures):

$$\frac{\partial f(x,y)}{\partial x} \approx f(x,y) - f(x-1,y)$$

$$\frac{\partial f(x,y)}{\partial y} \approx f(x,y) - f(x,y-1)$$

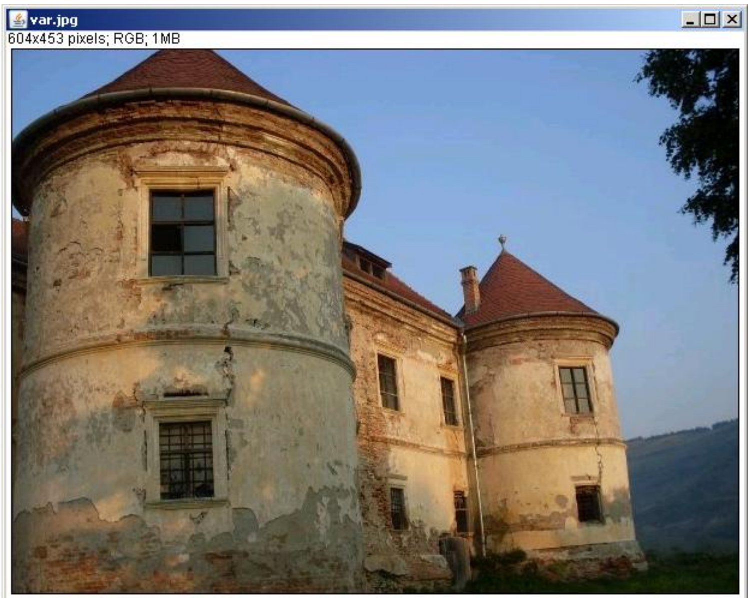
$$G_{mag} \approx \sqrt{\left(f(x,y) - f(x-1,y)\right)^2 + \left(f(x,y) - f(x,y-1)\right)^2}$$

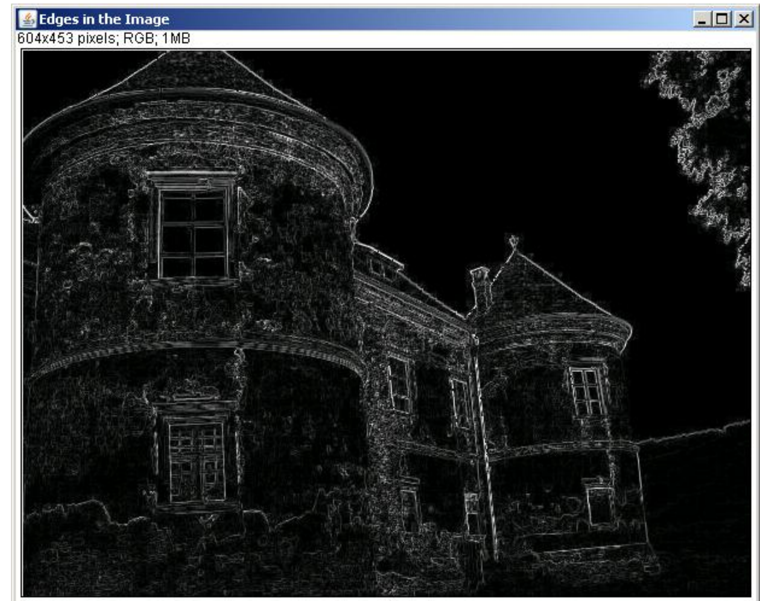
Gradient for Edge Detection

- Edge detection:
  - Calculate the gradient of an image

```
function gradient(Image image) {
    for (int x=1;x<image.getWidth();x++) {
        for (int y=1;y<image.getHeight();y++) {
            intensity1 = image.getIntensity(x, y);
            intensity2 = image.getIntensity(x-1, y);
            intensity3 = image.getIntensity(x, y-1);
            gradient = sqrt( (intensity1-intensity3)*(intensity1-intensity3) +
                             (intensity1-intensity2)*(intensity1-intensity2));
            edge_image.setIntensity(x, y, gradient);
        }
    }
    return edge_image;
}
```

Gradient for Edge Detection





Edge Detection

What is Edge Detection?

Gradient for Edge Detection

Convolution

Marr-Hildreth

Canny

Hough Transformation

Can we formulate gradient as Convolution?

$$\frac{\partial f(x,y)}{\partial x} \approx f(x,y) - f(x-1,y) \qquad K_x = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & -1 \\ 0 & 0 & 0 \end{bmatrix}$$

$$\frac{\partial f(x,y)}{\partial y} \approx f(x,y) - f(x,y-1) \qquad K_y = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -1 & 0 \end{bmatrix}$$

$$G_{mag} \approx \sqrt{\left(f(x,y) - f(x-1,y)\right)^2 + \left(f(x,y) - f(x,y-1)\right)^2}$$

$$K_{grad} = ?$$

Sobel Edge Detector

- Calculate *x* and *y* components separately by convolving the image with S_x and S_y and use them in the gradient formula

$$S_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \quad \rightarrow \quad f'_x(x,y)$$

$$S_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} \quad \rightarrow \quad f'_y(x,y)$$

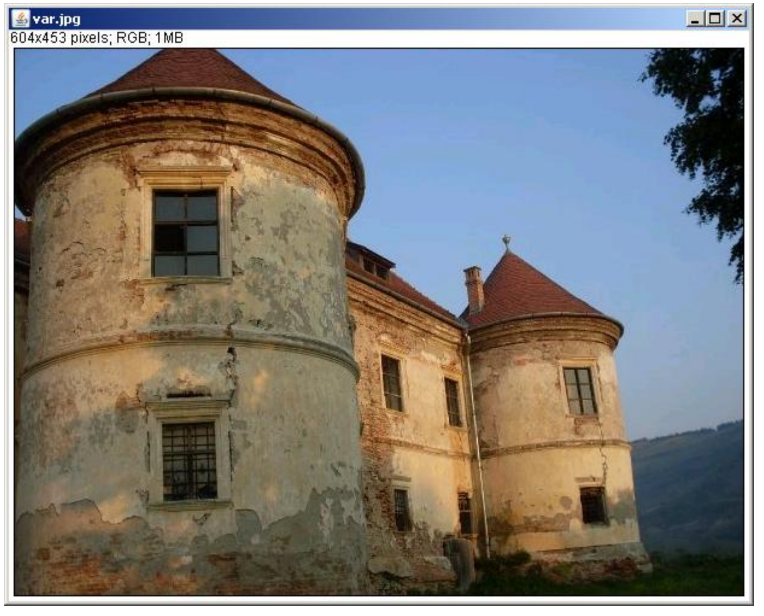
$$G^*_{mag} \approx \sqrt{\left(f'_x(x,y)\right)^2 + \left(f'_y(x,y)\right)^2}$$

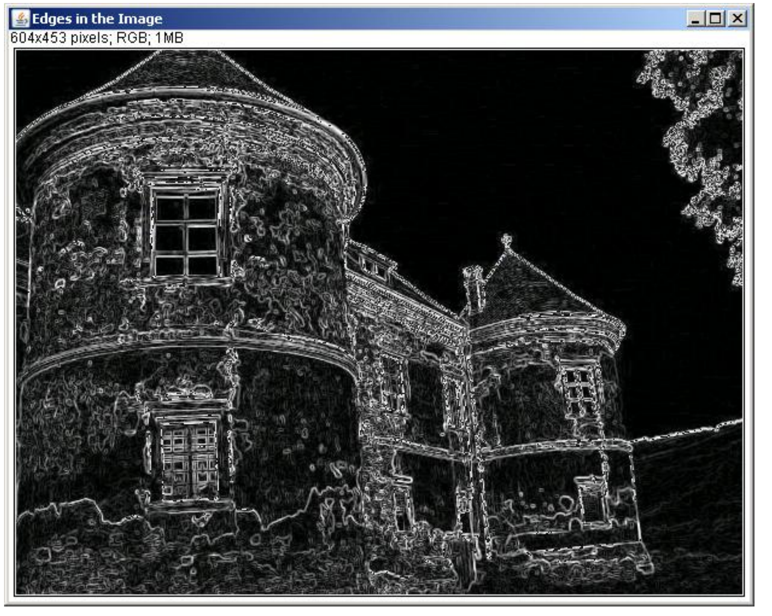
Sobel Edge Detector

```
function sobel(Image image) {
    int[][] Sx = { {-1,0,1},  {-2,0,2}, {-1,0,1} };    int[][] Sy = { {-1,-2,-1}, {0,0,0}, {1,2,1} };

    for (int x=1;x<image.getWidth()-1;x++)  {
        for (int y=1;y<image.getHeight()-1;y++)  {
            int intensity_sum_x=0, intensity_sum_y=0;
            for (int i=-1;i<=1;i++) {
                for (int j=-1;j<=1;j++) {
                    int intensity = image.getintensity(x+i, y+j);
                    intensity_sum_x+=(intensity*Sx[1-j][1-i]);
                    intensity_sum_y+=(intensity*Sy[1-j][1-i]);
                    // 1st index of the array Sx and Sy --> row of the kernel matrix
                    // 2nd index of the array Sx and Sy --> column of the kernel matrix
                }
            }
        int new_intensity = sqrt((intensity_sum_x)*(intensity_sum_x)+(intensity_sum_y)*(intensity_sum_y));
        edges_image.setIntensityPixel(x, y, new_intensity);
        }
    }
    return edges_image;
}
```

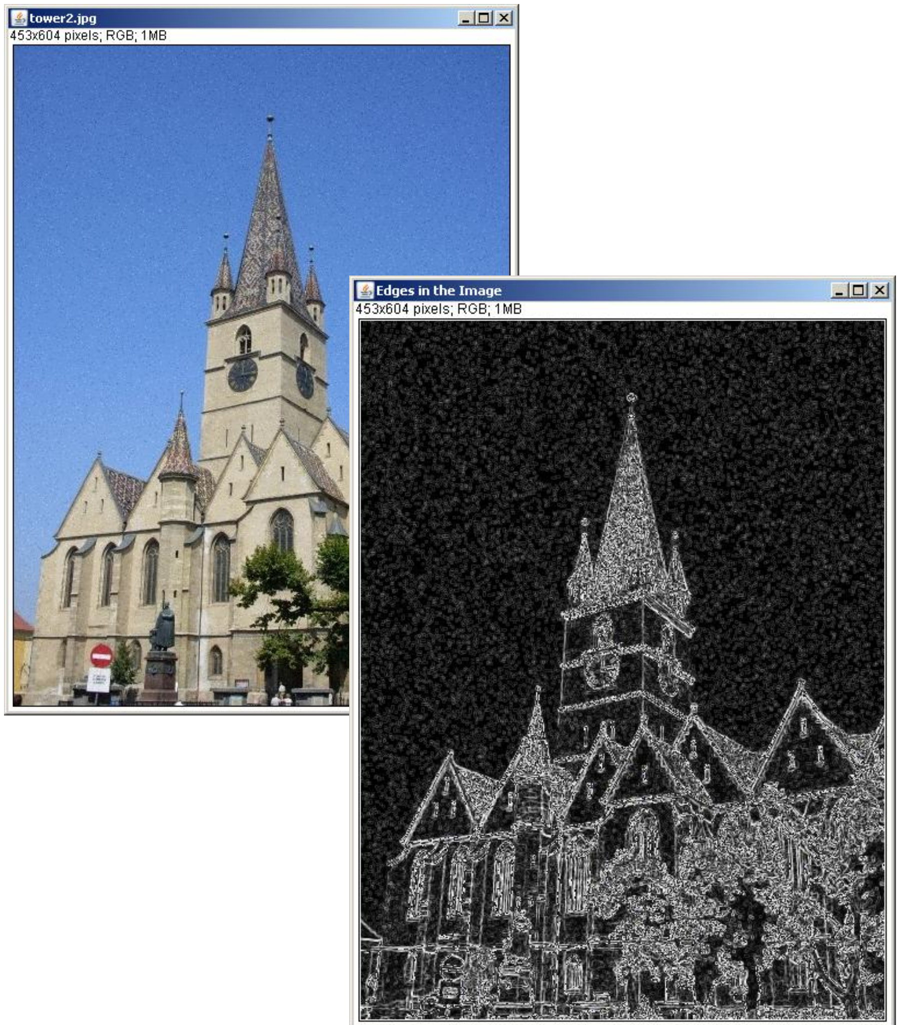
Sobel Edge Detector

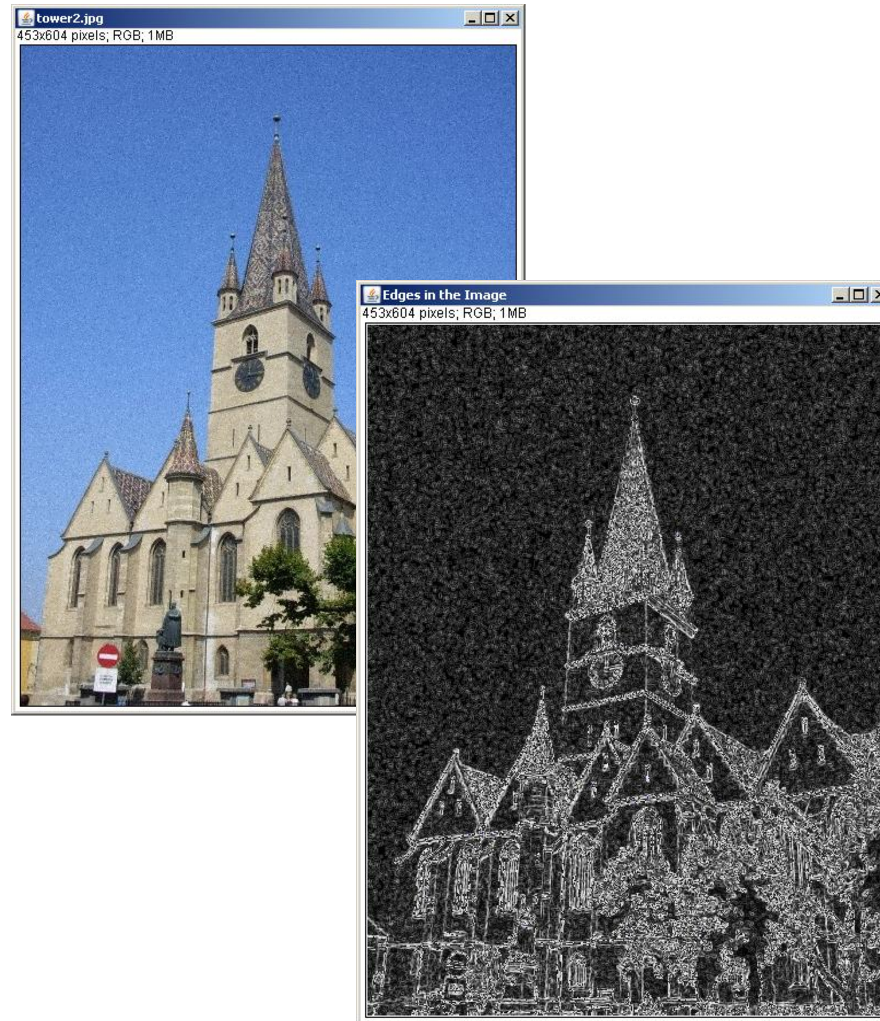




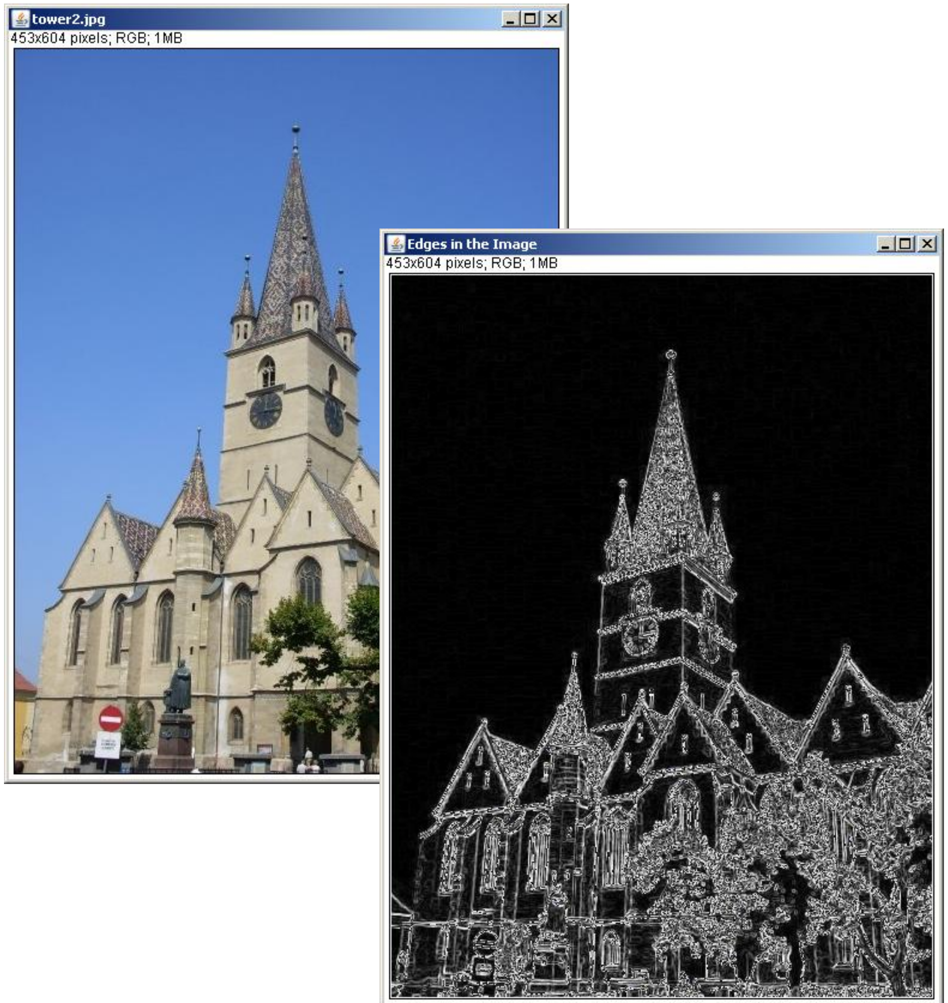
- Better or worse than gradient?

Sobel Edge Detector under noise





Presence and absence of noise



- Edge detectors seen so far, work well if there is no noise
- In case of noise, use noise filter first
- More robust edge detection (next sections)

Edge Detection

What is Edge Detection?

Gradient for Edge Detection

Convolution

<u>Marr-Hildreth</u>

Canny

Hough Transformation

Marr-Hildreth Edge Detection

- Gaussian filter for noise reduction
- Edge → Extrem value in the first derivative of the intensity → **zero crossing** in the second derivative
- Second derivative (Laplacian) of an image is:

$$\nabla^2 f(x,y) = \frac{\partial^2}{\partial x^2} f(x,y) + \frac{\partial^2}{\partial y^2} f(x,y)$$

- Laplacian is defined as the divergence of the gradient.
  (Divergence: sum of partial derivatives → scalar)
  This is equivalent with the above formula.
  The Laplacian consists of scalar values!
- **Be careful**: the gradient of an image consists of vectors (not scalars!) These vectors have some length, called magnitude, of course, which is scalar.

Marr-Hildreth Edge Detection

- Laplacian of an image

$$\nabla^2 f(x,y) = \frac{\partial^2}{\partial x^2} f(x,y) + \frac{\partial^2}{\partial y^2} f(x,y)$$

$$\frac{\partial^2}{\partial x^2} f(x,y) = \frac{\partial}{\partial x}\left(\frac{\partial}{\partial x} f(x,y) \right)$$

$$\frac{\partial f(x,y)}{\partial x} \approx f(x,y) - f(x-1,y) \qquad K_x = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & -1 \\ 0 & 0 & 0 \end{bmatrix}$$

$$\frac{\partial f(x,y)}{\partial x} \approx f(x+1,y) - f(x,y) \qquad K_x^* = \begin{bmatrix} 0 & 0 & 0 \\ 1 & -1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

Marr-Hildreth Edge Detection

- Second derivative in the *x* direction

$$f_{xx}''(x,y) \approx f_x'(x+1,y) - f_x'(x,y) \approx$$
$$\big(f(x+1,y) - f(x,y)\big) - \big(f(x,y) - f(x-1,y)\big) =$$
$$f(x+1,y) - 2f(x,y) + f(x-1,y)$$

- Corresponding convolution kernel

$$K_{xx} = \begin{bmatrix} 0 & 0 & 0 \\ 1 & -2 & 1 \\ 0 & 0 & 0 \end{bmatrix}$$

Marr-Hildreth Edge Detection

- Second derivative in the *x* direction

$$K_{xx} = \begin{bmatrix} 0 & 0 & 0 \\ 1 & -2 & 1 \\ 0 & 0 & 0 \end{bmatrix}$$

- Second derivative in the *y* direction

$$K_{yy} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & -2 & 0 \\ 0 & 1 & 0 \end{bmatrix}$$

- Convolution matrix for the Laplacian (L):

$$L = K_{xx} + K_{yy} = \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

Marr-Hildreth Edge Detection

- A possible gaussian noise filter:

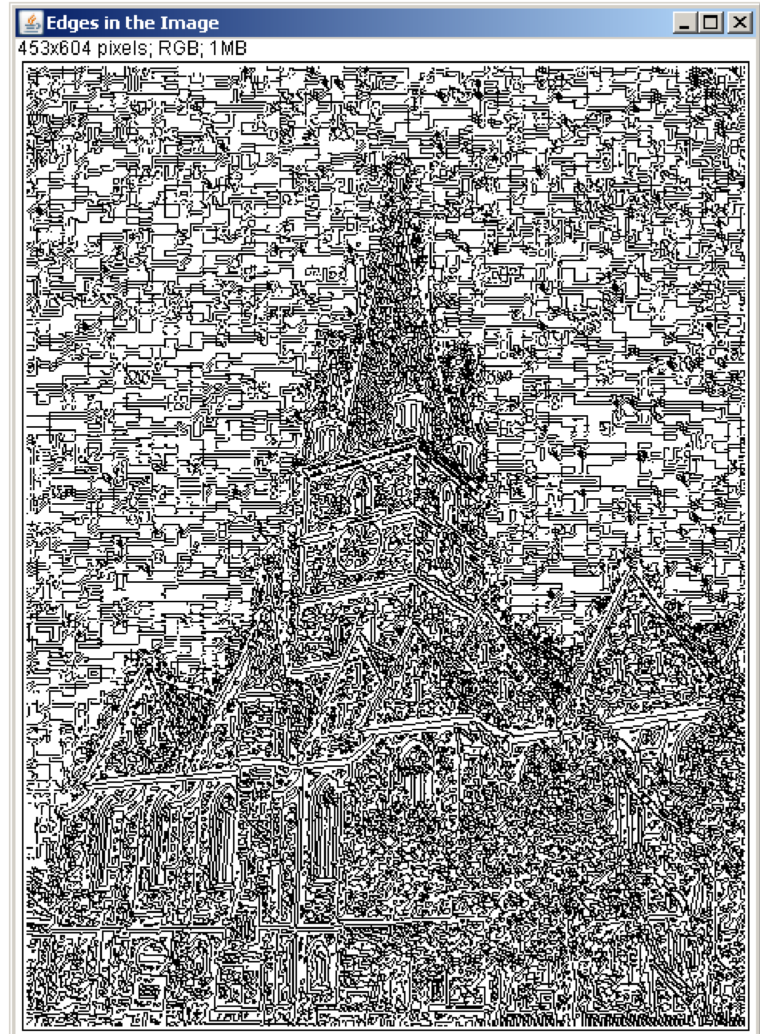
$$G = \begin{bmatrix} 0.0625 & 0.125 & 0.0625 \\ 0.125 & 0.25 & 0.125 \\ 0.0625 & 0.125 & 0.0625 \end{bmatrix}$$

- Apply Gaussian filter first, then calculate the Laplacian of the image:

$$f_1(x,y) = G * f(x,y)$$
$$f_2(x,y) = L * f_1(x,y)$$

- Laplacian of the Gaussian (LoG)
  - The both above operations as one convolution matrix
  - Homework: calculate the LoG matrix

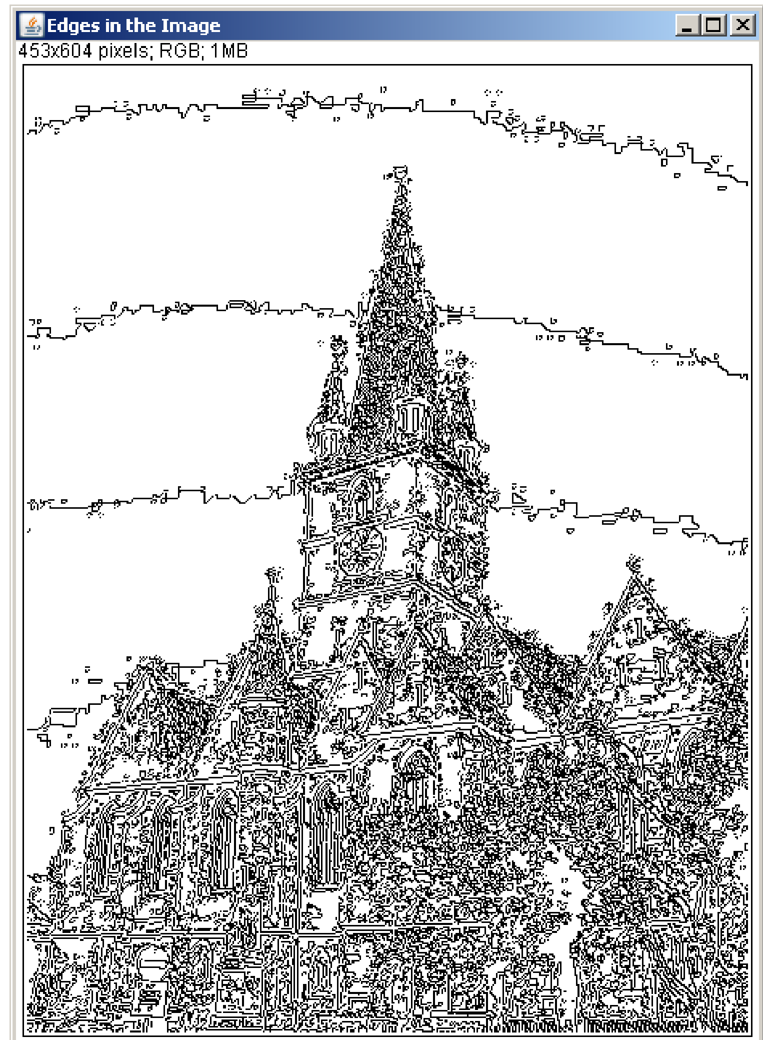
Marr-Hildreth Edge Detection



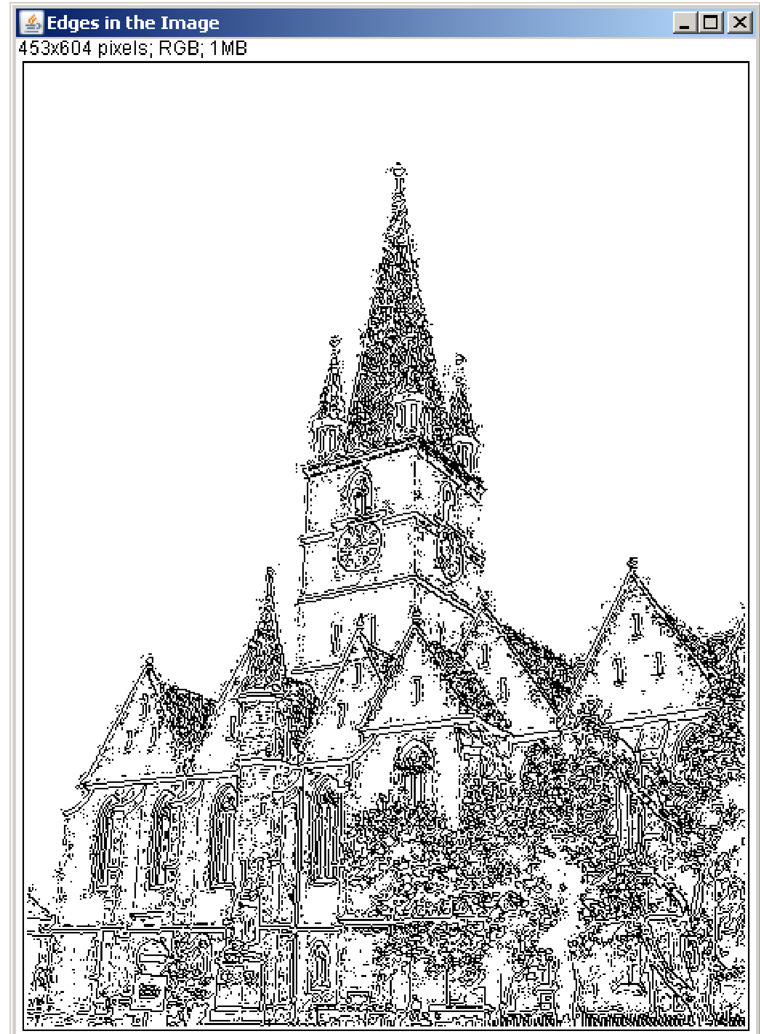
Marr-Hildreth Edge Detection

- Why do things go wrong?
  - For example sky pixels are not homogenous blue. If a pixel is a „little bit“ different, than the others around it → an edge will be detected
- What can be done?
  - Reduce the number of gray levels used in the calculation of the LoG of the image
  - Take only “drastic” zero crossings into account when determining where are the edges

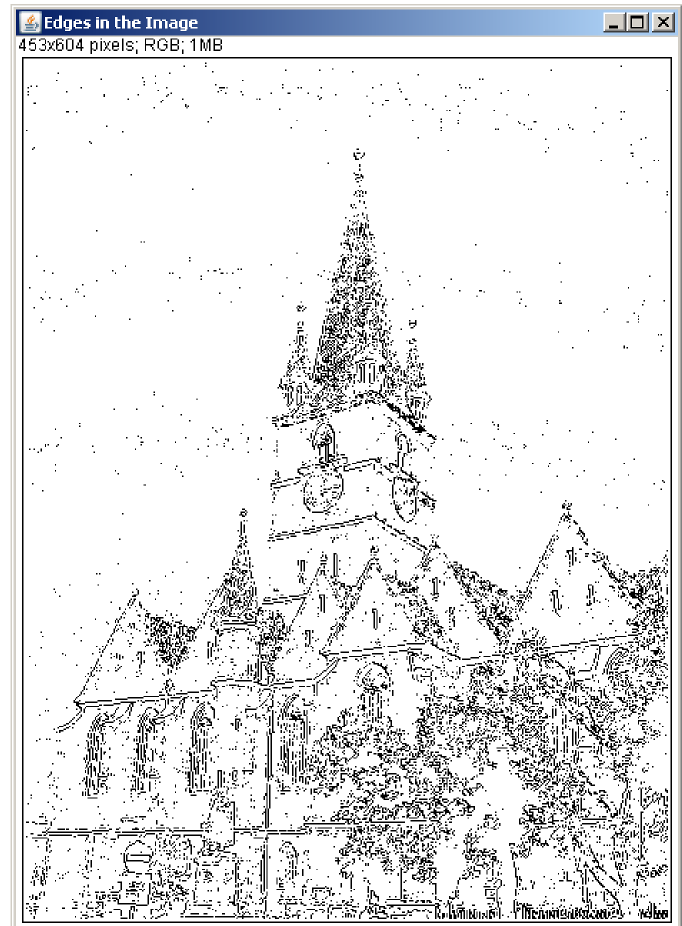
Marr-Hildreth Edge Detection



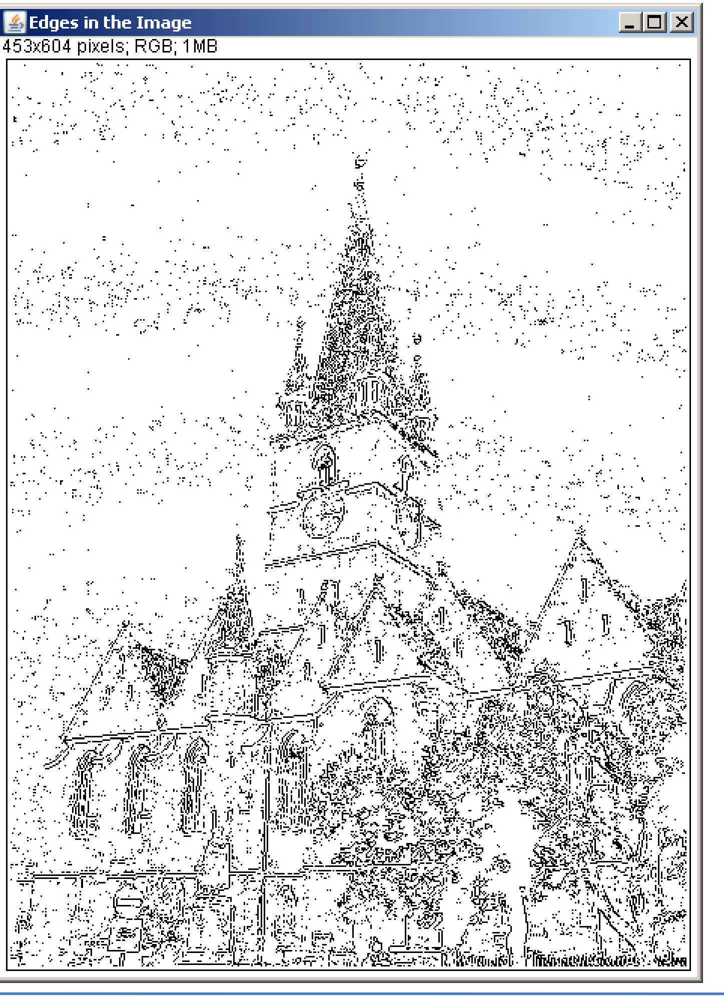
Marr-Hildreth Edge Detection



Marr-Hildreth Edge Detection in presence of noise



Marr-Hildreth Edge Detection in presence of noise



Edge Detection

What is Edge Detection?

Gradient for Edge Detection

Convolution

Marr-Hildreth

Canny

Hough Transformation

Canny Edge Detection

- Use the derivative of the Gaussian
- Speed-up edge detection process:
  - instead of convolution with a matrix (two dimensional array), two convolutions with two vectors (one dimensional arrays)
- Post-processing steps
  - Non-maximal suppression
  - Hysteresis sampling

Canny Edge Detection

- Derivative of the Gaussian

$$G = \begin{bmatrix} 0 & 0.1 & 0.2 & 0.4 & 0.2 & 0.1 & 0 \end{bmatrix}$$

$$G' = \begin{bmatrix} 0 & 0.1 & 0.1 & 0.2 & -0.2 & -0.1 & -0.1 \end{bmatrix}$$

- Steps of the algorithm (Input Image I)
  1. Convolve the image I with G → I_x
  2. Convolve the image I with G^T → I_y
  3. Convolve I_x with G' → I'_x
  4. Convolve I_y with G'^T → I'_y
  5. For each pixel (x,y) there is a vector: $R(x,y) = (I'_x\,(x,y), I'_y\,(x,y))$
  6. Perform Non-maximal suppression
  7. Perform Hysteresis Sampling

Non-maximal Suppression

- Denote the result of steps 1-5 with R.
- An edge at the pixel (x,y) in the image corresponds to a local maximum of $|R|$ at (x,y). This local maximum is meant according to direction of $R(x,y)$. For each pixel:
  - Select the "next" pixels both in the direction of $R(x,y)$ and in the opposite directions

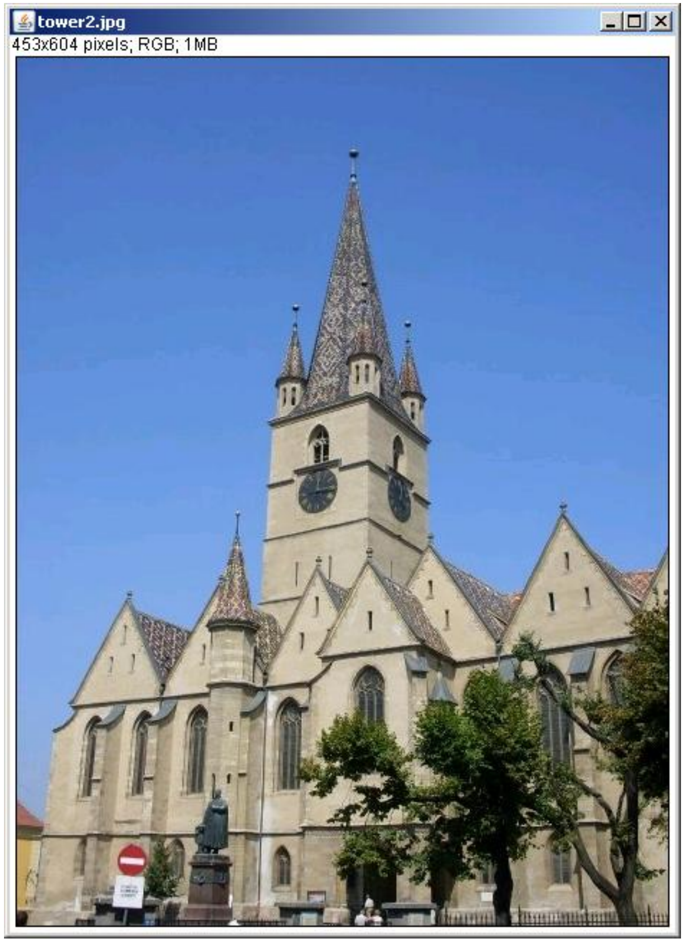
(linear Interpolation)

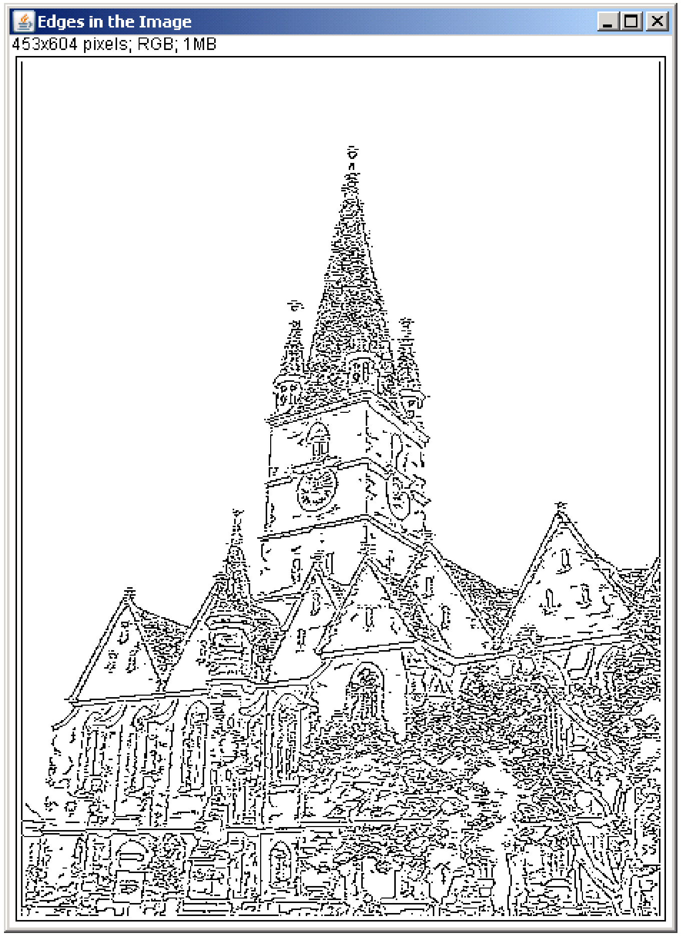
  - Check if there is a local maximum at (x,y) and indicate non-local maxima in R as non-edge pixels, i.e. set $R(x,y)=0$, if (x,y) is not a local maximum

Hysteresis sampling

- Given two thresholds T_{high} and T_{low}
- Mark all pixels as non-edge, if $R(x,y) < T_{low}$
- Mark all pixels as edge, if $R(x,y) > T_{high}$
- For the other pixels:
  - Mark the neighbors of edge pixels as edge-pixel iteratively

Canny Edge Detection





Edge Detection

What is Edge Detection?

Gradient for Edge Detection

Convolution

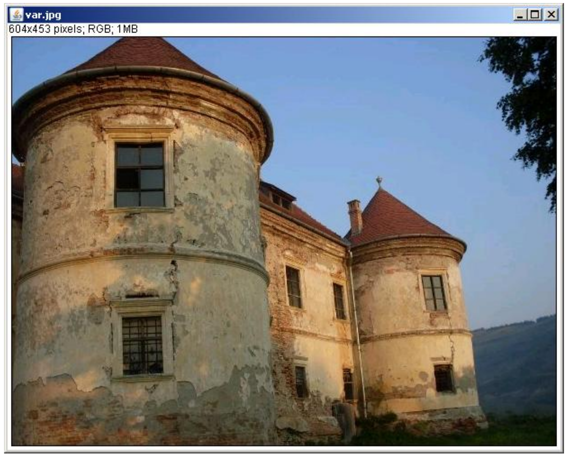
Marr-Hildreth

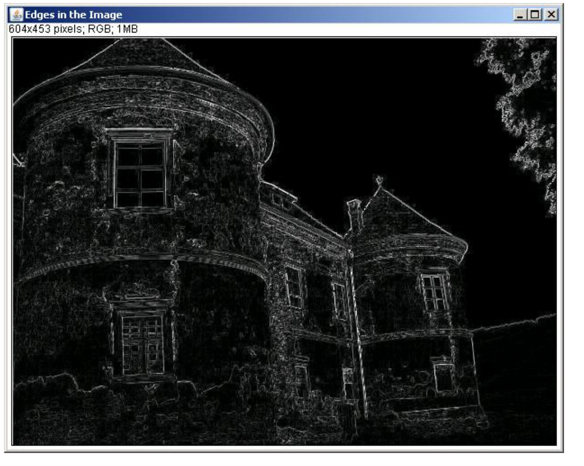
Canny

Hough Transformation

What is Edge Detection?

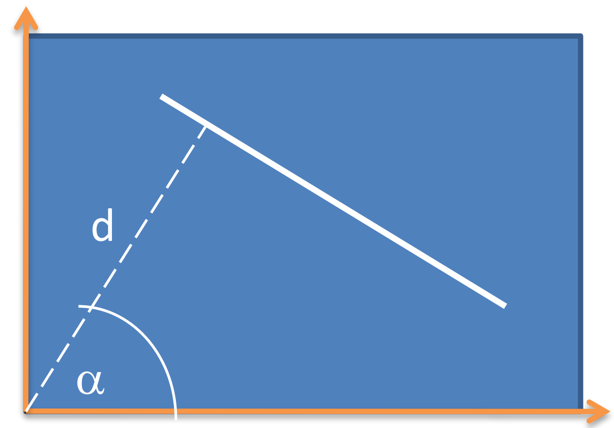
- “Low level“ edge detection
  - Which pixels belong to an edge?
  - What is the (local) orientation of those pixels?
  - Gradient, Sobel, Kirsch, Marr-Hildreth, Canny
- “High level” edge detection
  - Characterize edges in the picture
  - For example:
    - Lines by their offset and slope
    - Circles by their center and radius
    - …
  - **Hough transform**

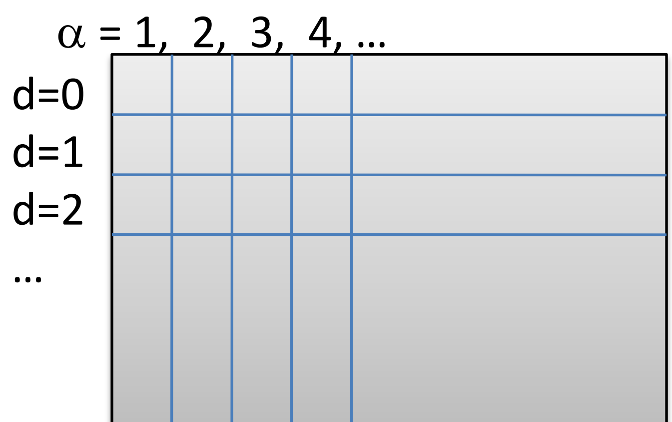




Hough Transformation: Detection of straight lines

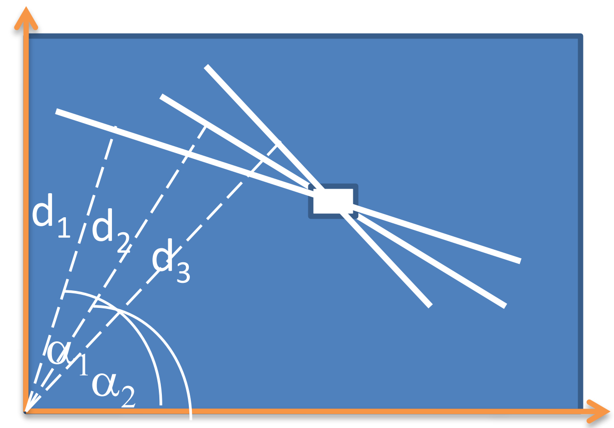
- A line can be described by *d* and α.
- Make a “catalog” of “all” the possible lines (create a counter for each of the lines)
- Traverse through the pixels of the image. For each pixel of the foreground:
  - This pixel can belong the several lines. Increment the counters of ALL of these lines
- Finally, the counters with “highest” values correspond to real lines

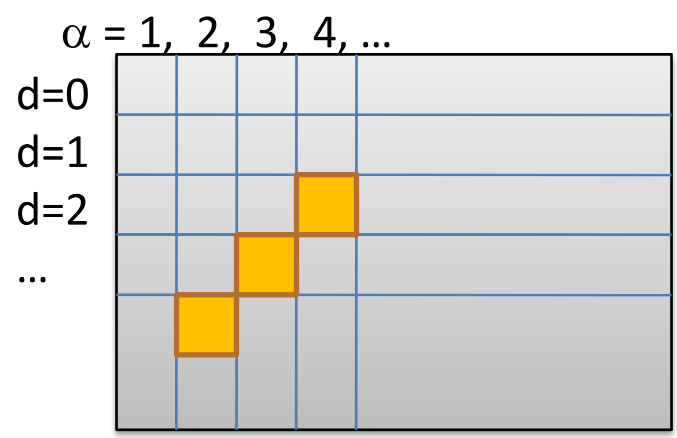




Hough Transformation: Detection of straight lines

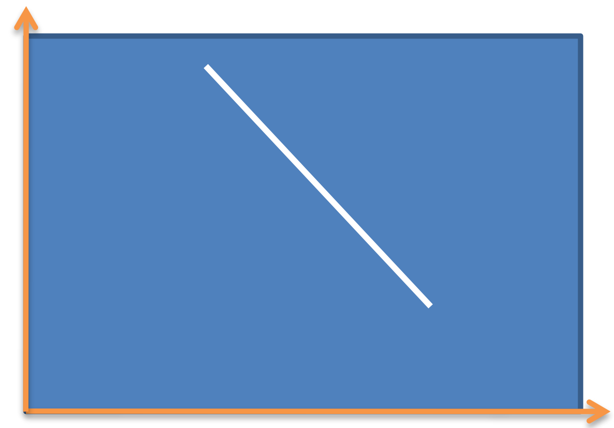
- A line can be described by *d* and α.
- Make a "catalog" of "all" the possible lines (create a counter for each of the lines)
- Traverse through the pixels of the image. For each pixel of the foreground:
  - This pixel can belong the several lines. Increment the counters of ALL of these lines
- Finally, the counters with "highest" values correspond to real lines

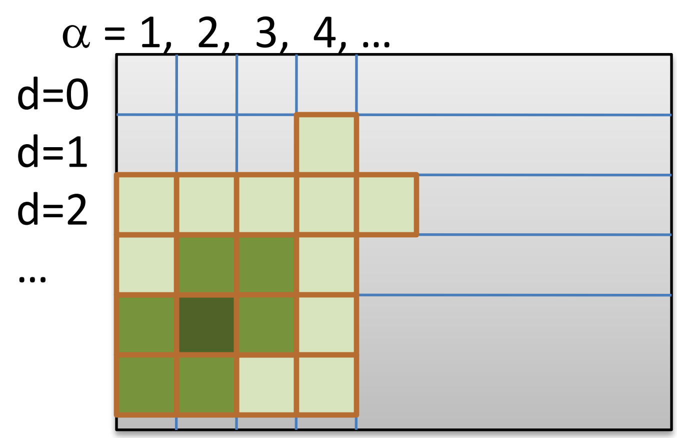




Hough Transformation: Detection of straight lines

- A line can be described by *d* and α.
- Make a “catalog” of “all” the possible lines (create a counter for each of the lines)
- Traverse through the pixels of the image. For each pixel of the foreground:
  - This pixel can belong the several lines. Increment the counters of ALL of these lines
- Finally, the counters with “highest” values correspond to real lines

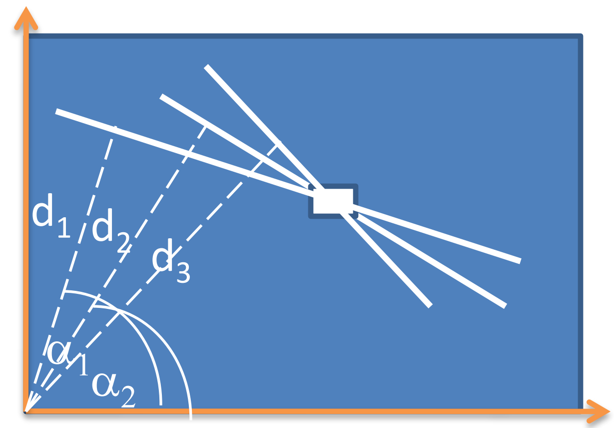


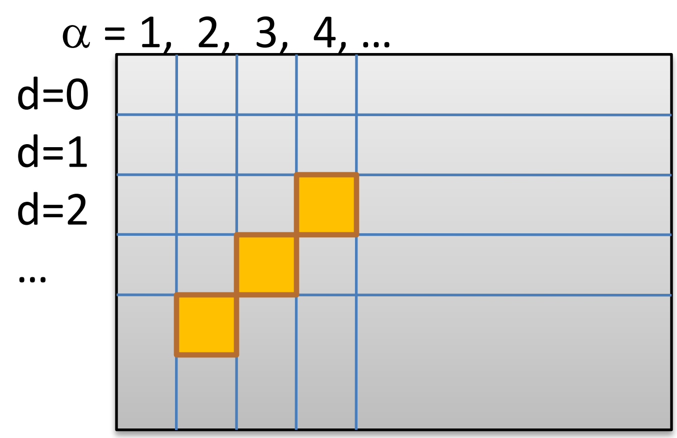


Which lines belong to each pixel?

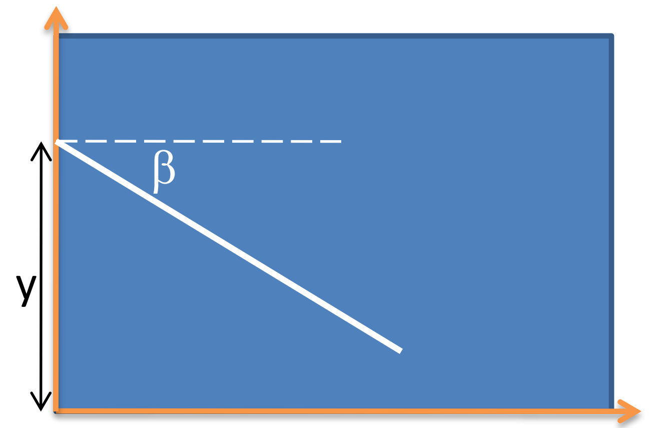
- Traverse through the pixels of the image. For each pixel of the foreground:
  - This pixel can belong the several lines. Increment the counters of ALL of these lines
- This can be implemented the following way:

For each pixel (x,y) of the image
    if (x,y) is a foreground-pixel
        For $\alpha = 0 \ldots 90°$
            $\boldsymbol{d = x \cos \alpha + y \sin \alpha}$
            increment(d,α)





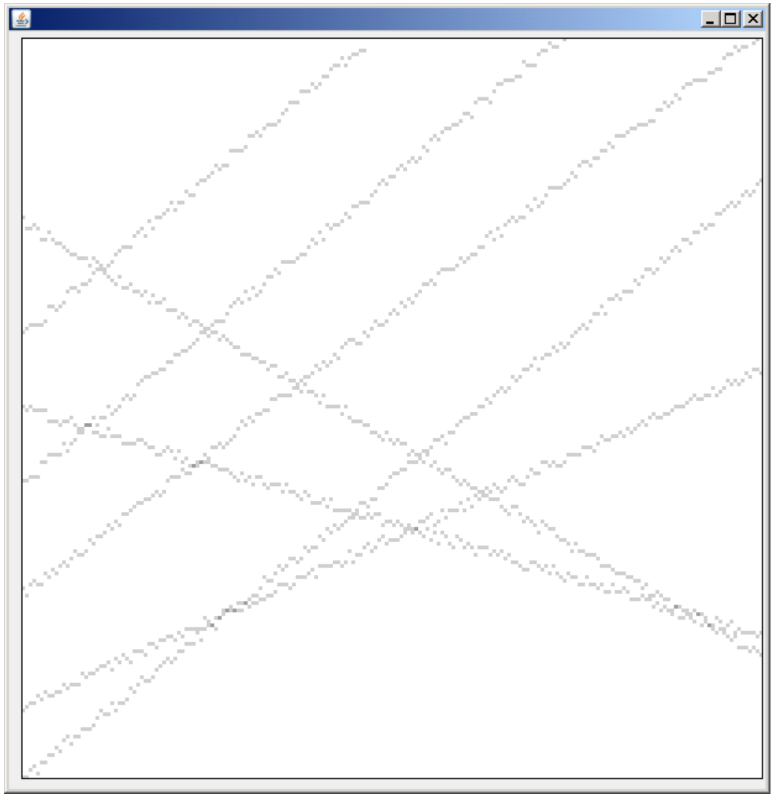
Hough Transformation

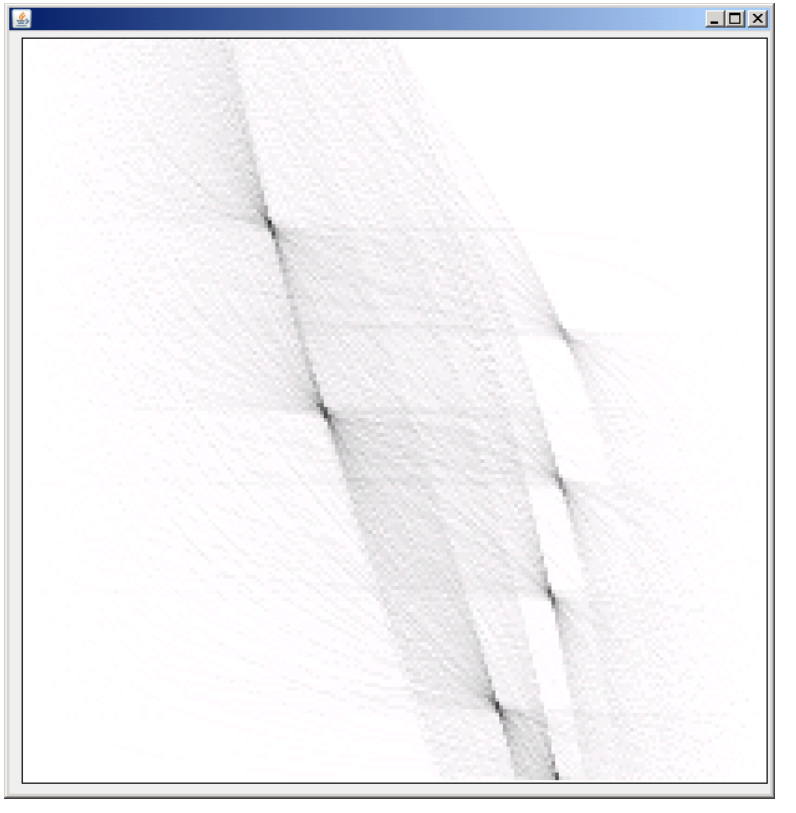


- There are other ways to describe a line, for example y-offset and b
- Using some parameters, one can describe other objects like circles, ellipses…
- The method seen before can easily be adapted for these cases
  - For example, one can describe a circle with its radius r and center x_0,y_0
  - For pixels (x,y) belonging to the circle: $(x-x_0)^2+(y-y_0)^2=r^2$
  - To which circles can a foreground pixel (x,y) belong?

for each foreground-pixel (x,y)
  for $x_0=0\ldots x_{max}$
    for $y_0=0\ldots y_{max}$
      $r=\sqrt{(x-x_0)^2+(y-y_0)^2}$
      increment(x_0, y_0, r)

Hough Transformation





β horizontal axle, -90° ...+90°
y-offset vertical axle

Outlook

- “Low level” edge detection can be though as a preprocessing step for “high level” edge detection (Hough transformation)
- Suppose one wants to determine the size of the wheels of the car in this image automatically
- First one can apply some low-level edge detection
- And then Hough transformation to detect circles