# Introduction to Robotics for cognitive science

**Dr. Andrej Lúčny**

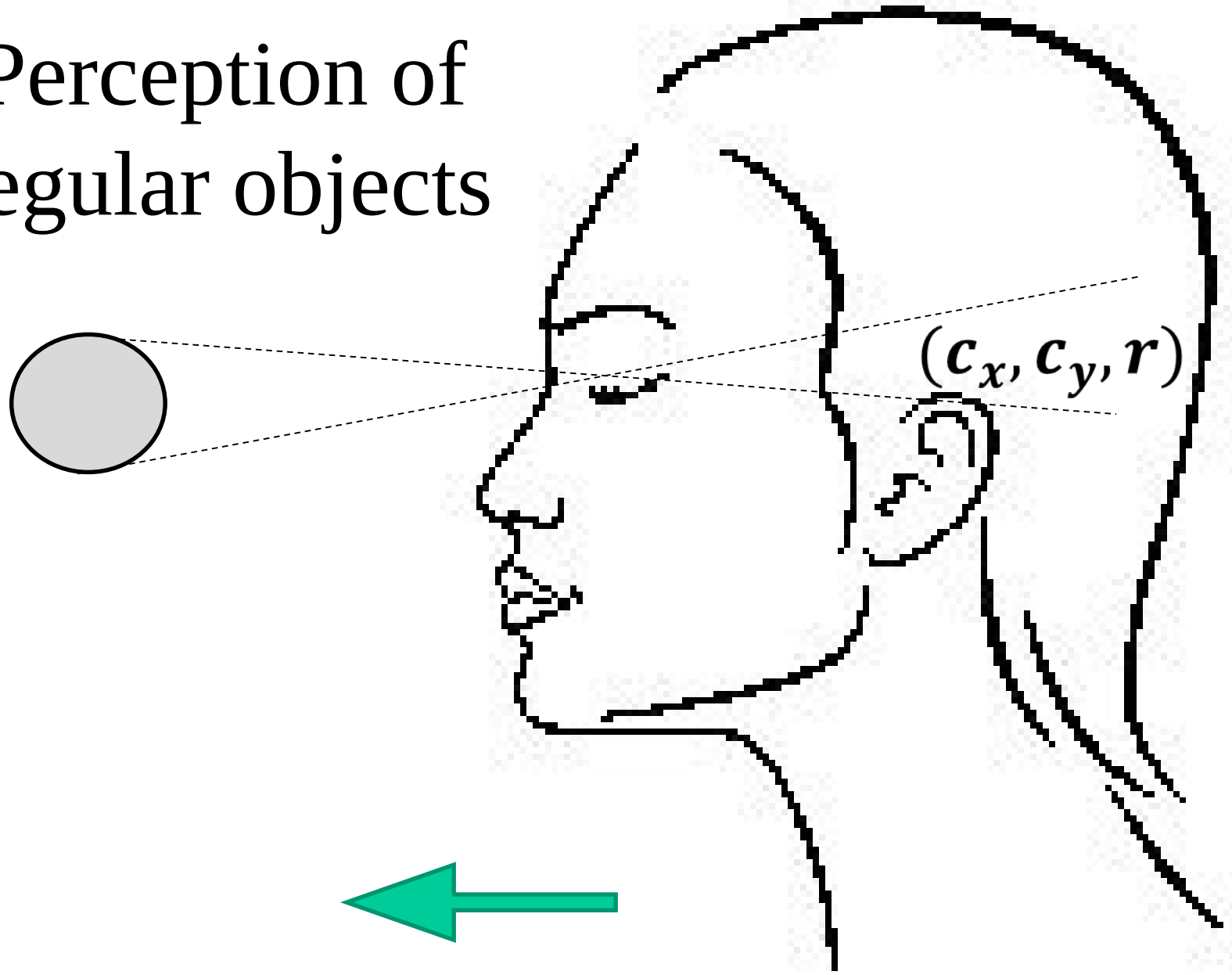
**KAI FMFI UK**

**lucny@fmph.uniba.sk**

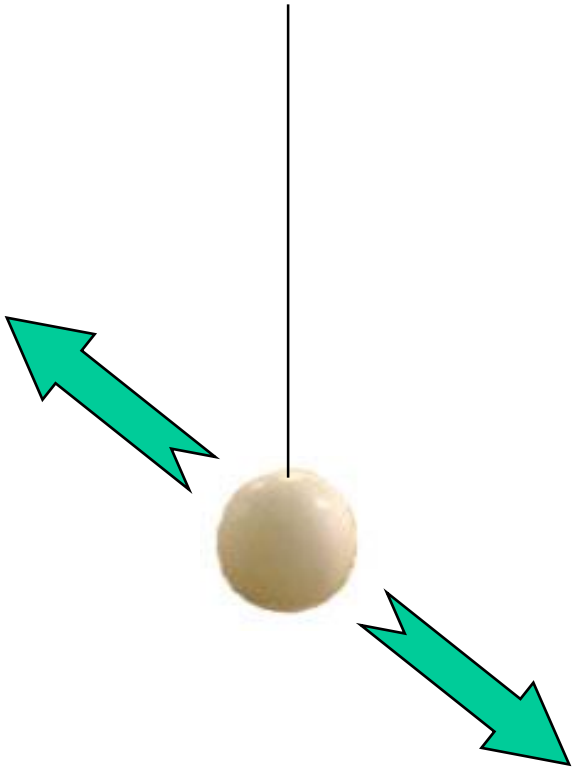
# Web page of the subject

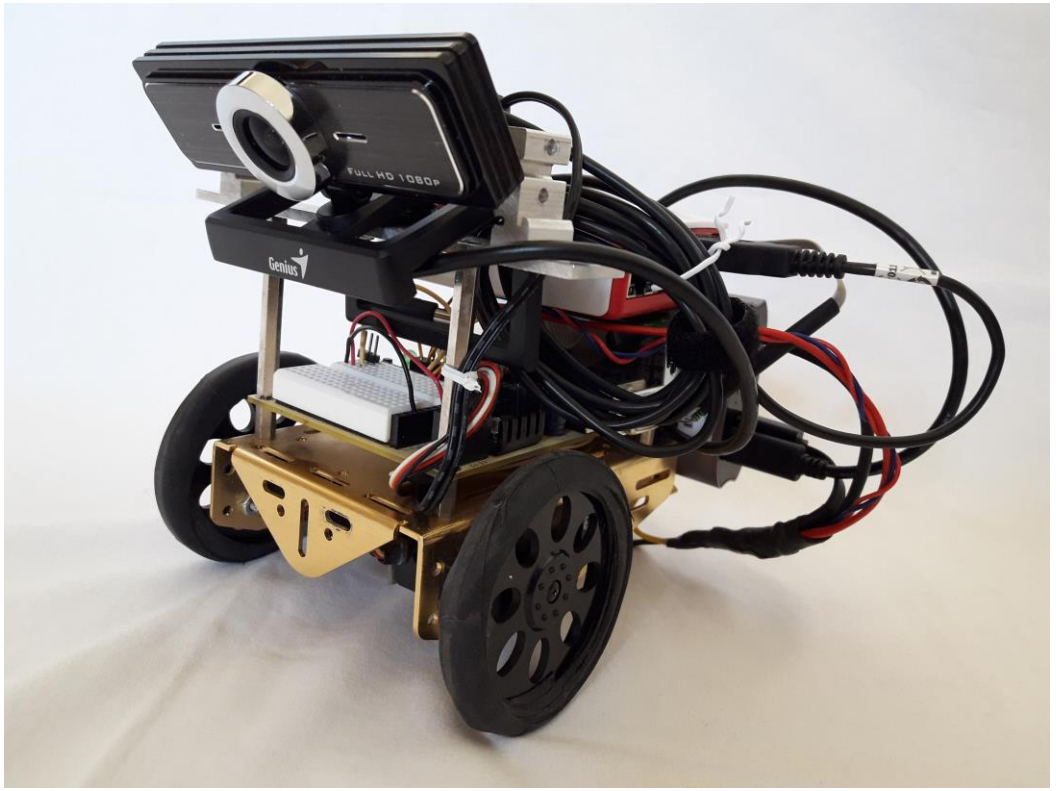
www.agentspace.org/kv

# Perception of regular objects

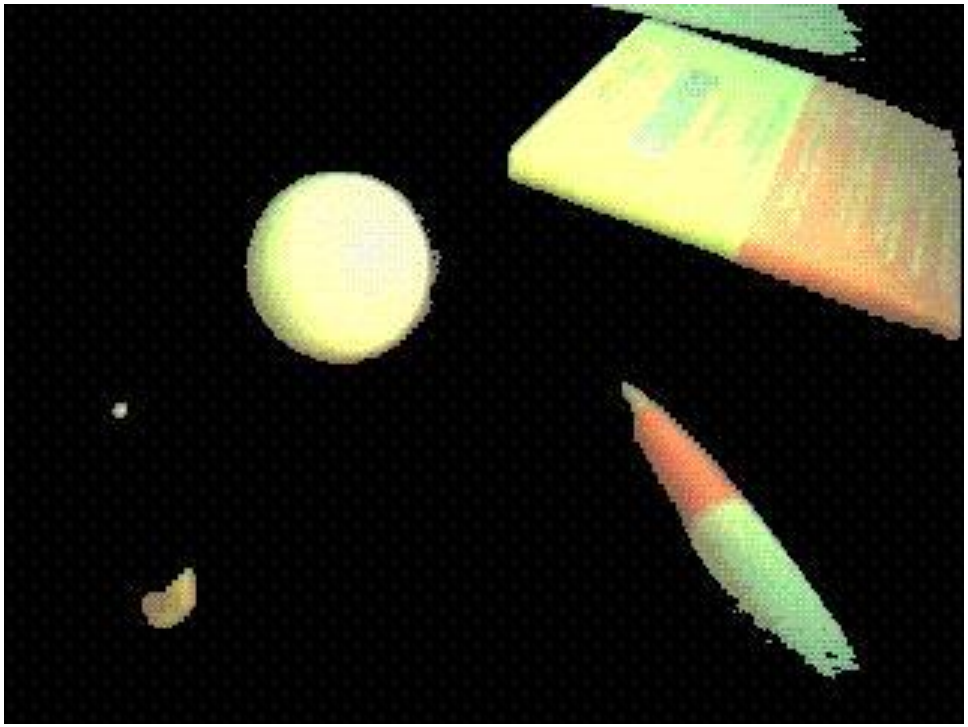


# Robot following ball





# Camera provides color image



BGR `np.array((height,width,3),np.uint8)`

# Ball following

1. Edge detection

2. Circle shape detection provides *(x,y,r)*
   where *(x,y)* is center of the ball circle and *r* is its radius

3. If *x* is too much on the left of image, turn left.
   If *x* is too much on the right of image, turn right.
   If *r* is too big, go backward.
   If *r* is too small, go forward.

# Grayscale image

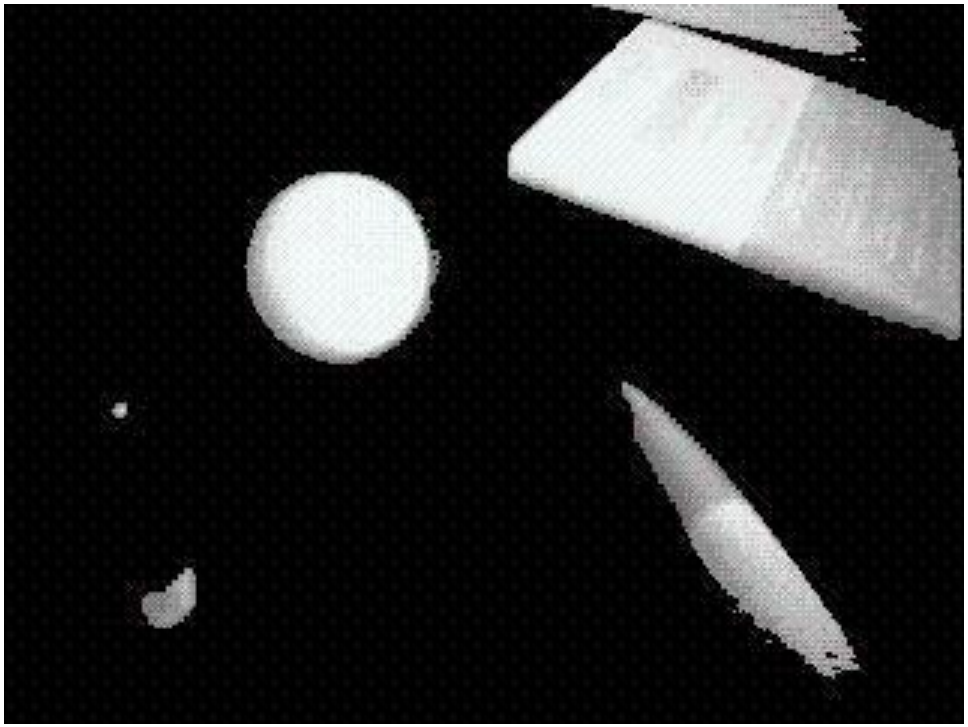
- Turn color image (height, width, 3) to grayscale (height, width)
- By numpy:

```
gray=np.asarray(np.average(rgb,axis=2),np.uint8)
```

or by openCV:

```
gray = cv2.cvtColor(bgr,cv.COLOR_BGR2GRAY)
```

# Grayscale image

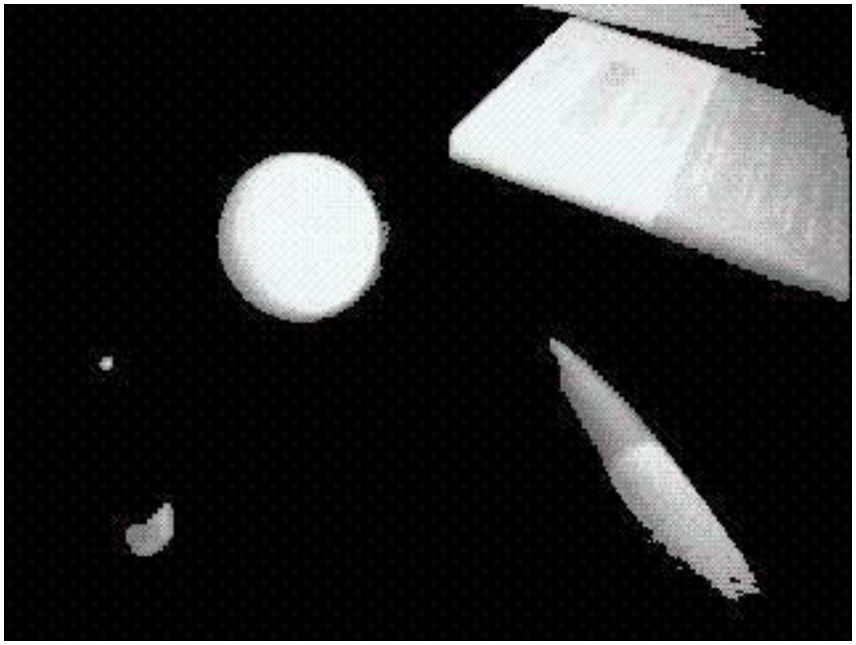


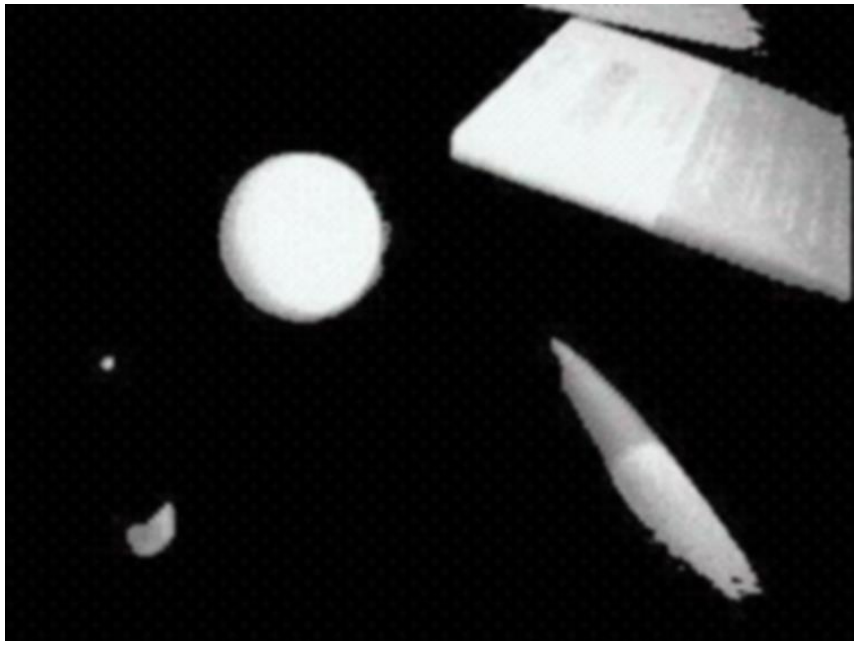
2D array of intensities 0..255

# Blur (noise reduction)

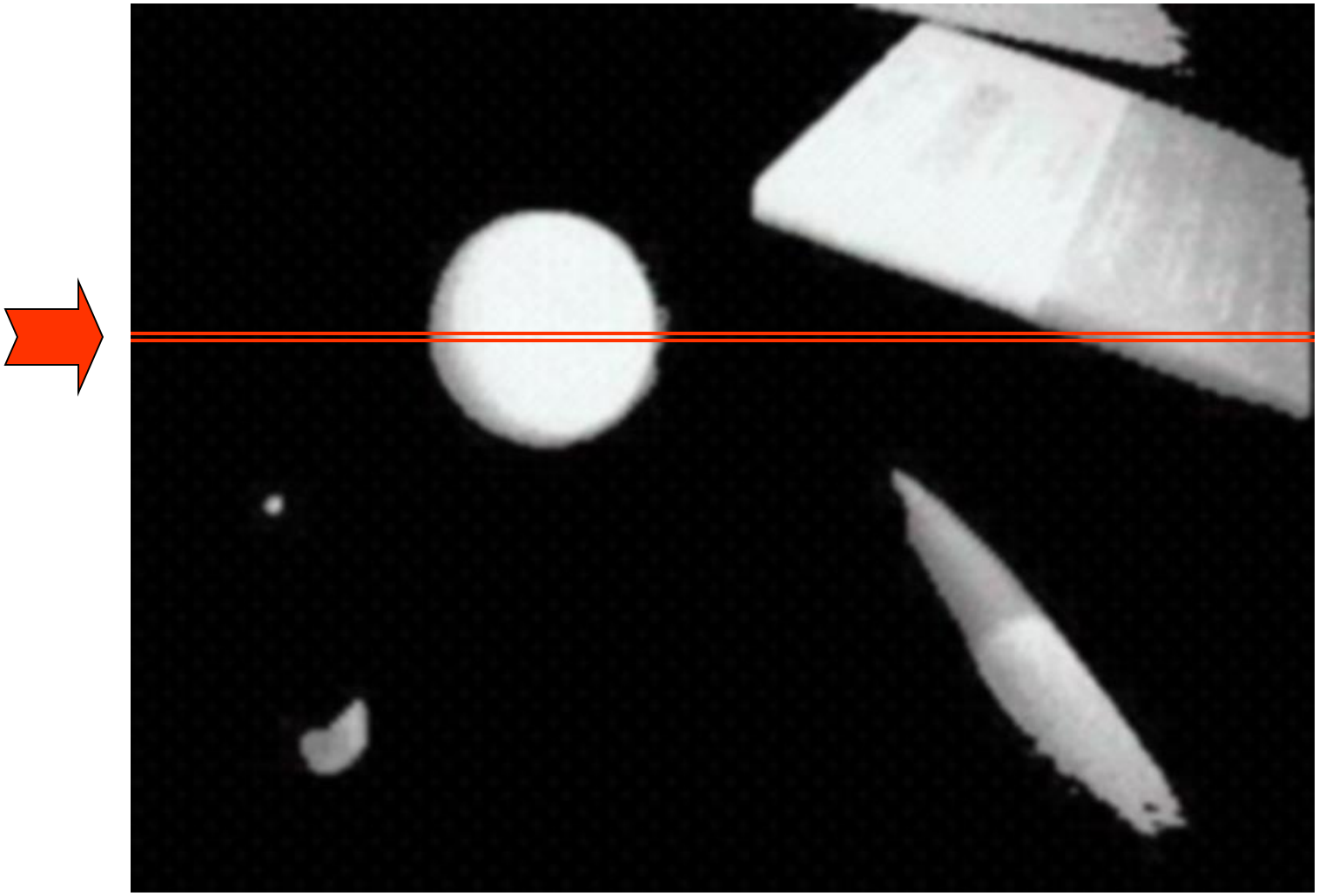
blur kernel 3x3

| | | |
|---|---|---|
| 1/9 | 1/9 | 1/9 |
| 1/9 | 1/9 | 1/9 |
| 1/9 | 1/9 | 1/9 |

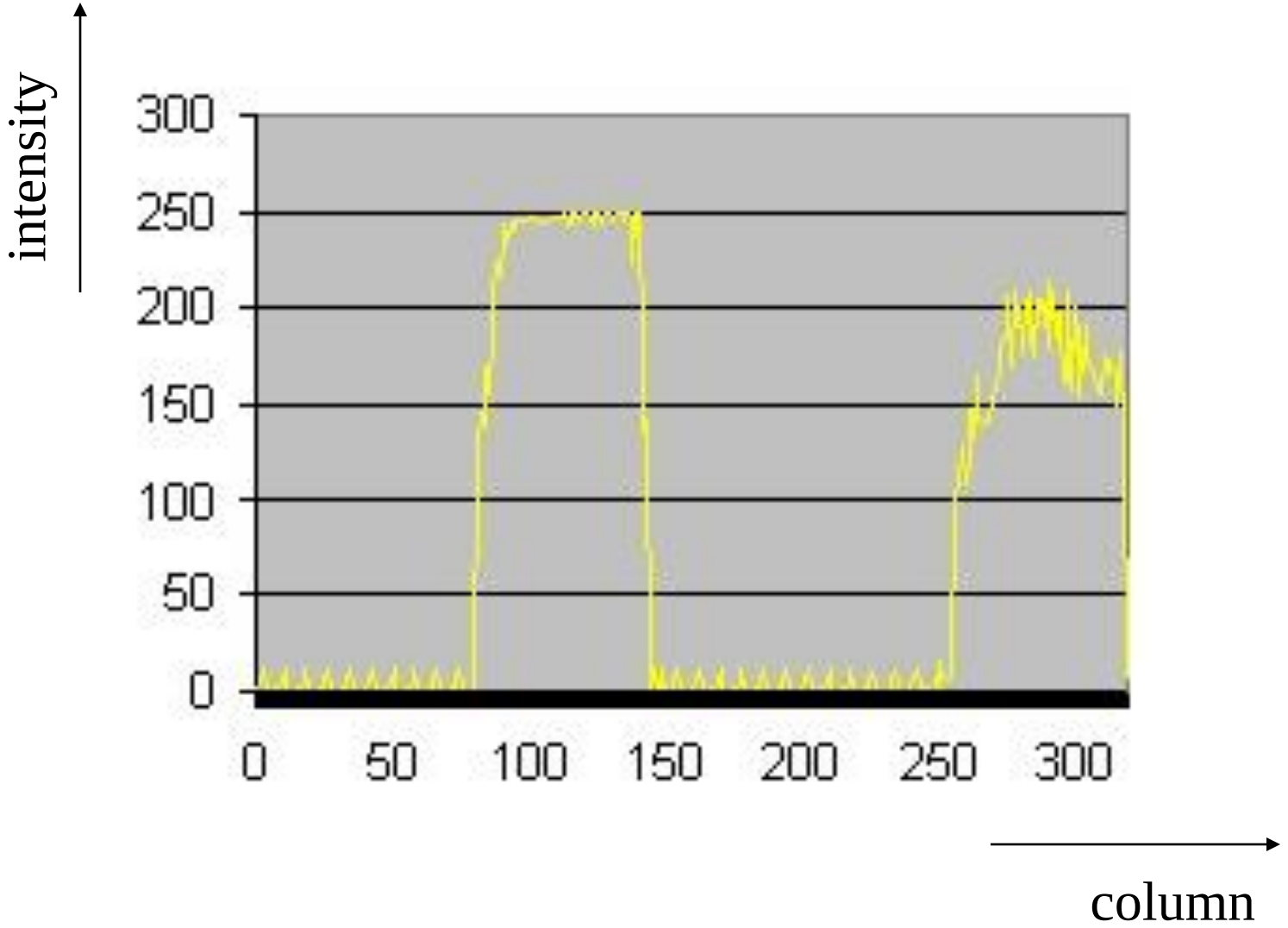




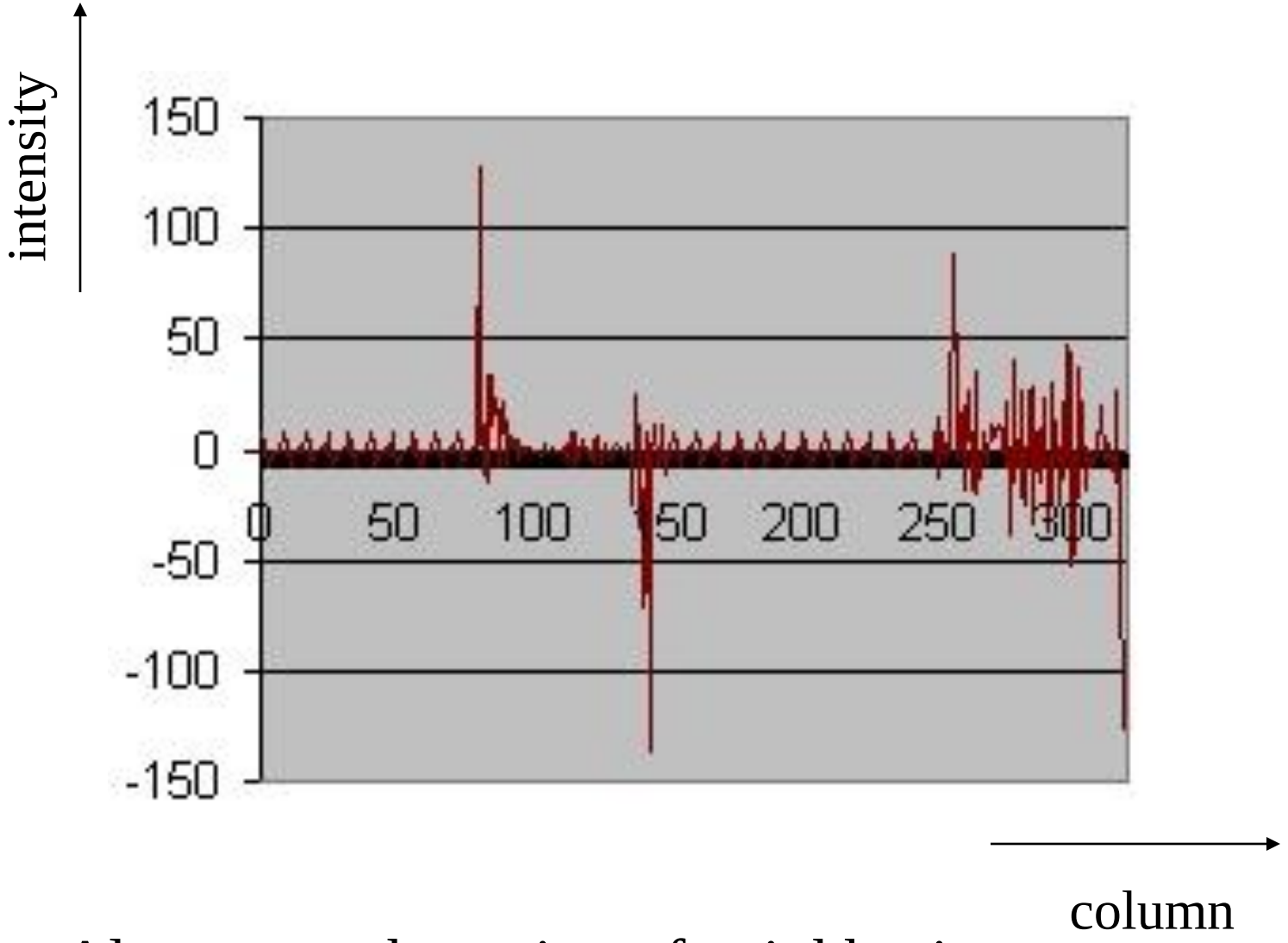
```
frame = cv2.blur(frame,(3,3))
```
or
```
frame = cv2.GaussianBlur(frame,(5,5),0)
```



- Let us consider one line as function of column




- Edges corresponds to big changes.
  How to filter them from the rest?




- Also pure subtraction of neighboring pixels give relatively good edges

# Sobel kernel (vertical)

- just little bit more sophisticated method
- concerns also other close pixels and aim to get more compact edges

| | | |
|---|---|---|
| $a_{i-1,j-1}$ | $a_{i-1,j}$ | $a_{i-1,j+1}$ |
| $a_{i,j-1}$ | $a_{i,j}$ | $a_{i,j+1}$ |
| $a_{i+1,j-1}$ | $a_{i+1,j}$ | $a_{i+1,j+1}$ |

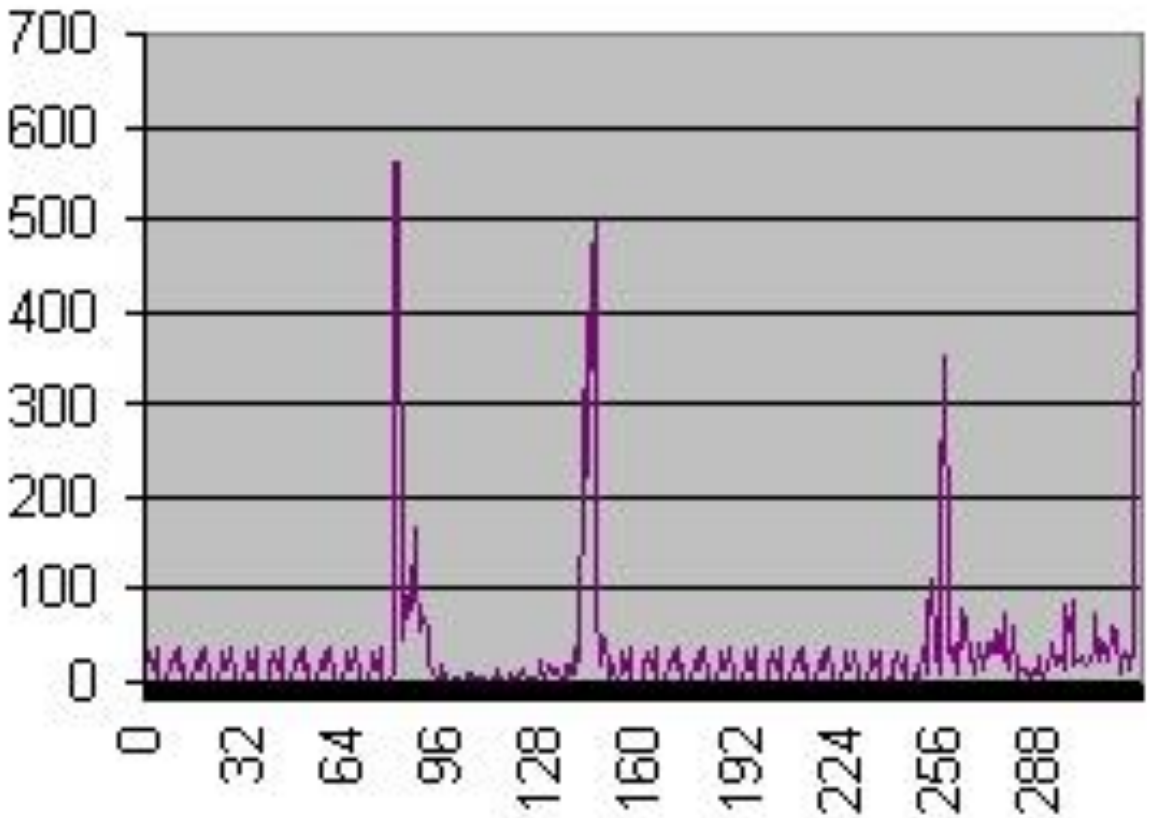
o

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

=

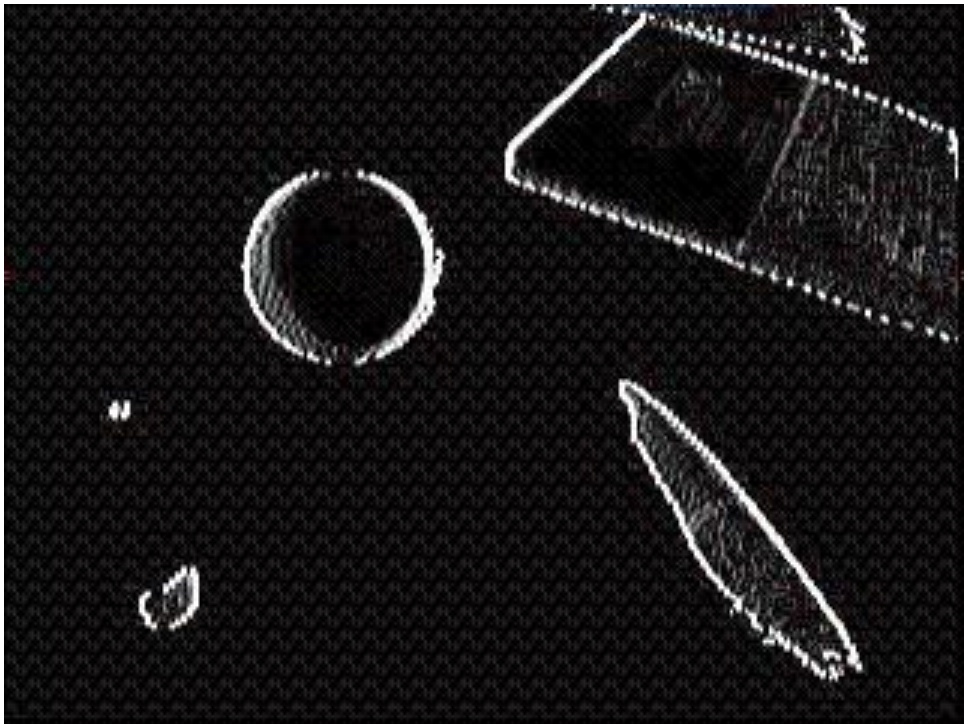
| | | |
|---|---|---|
| | | |
| | $b_{i,j}$ | |
| | | |

$$b_{i,j} = | a_{i-1,j+1} + 2a_{i,j+1} + a_{i+1,j+1} - a_{i-1,j-1} - 2a_{i,j-1} - a_{i+1,j-1} |$$




Yes, it really works better than the pure subtraction
Let us look on the resulting image:

# Sobel operator (vertical)



dx

We have got nice vertical edges and almost no horizontal edges

# Sobel kernel (horizontal)

- therefore we perform the same rotated by 90º
- and we combine results

| | | |
|---|---|---|
| $a_{i-1,j-1}$ | $a_{i-1,j}$ | $a_{i-1,j+1}$ |
| $a_{i,j-1}$ | $a_{i,j}$ | $a_{i,j+1}$ |
| $a_{i+1,j-1}$ | $a_{i+1,j}$ | $a_{i+1,j+1}$ |

o

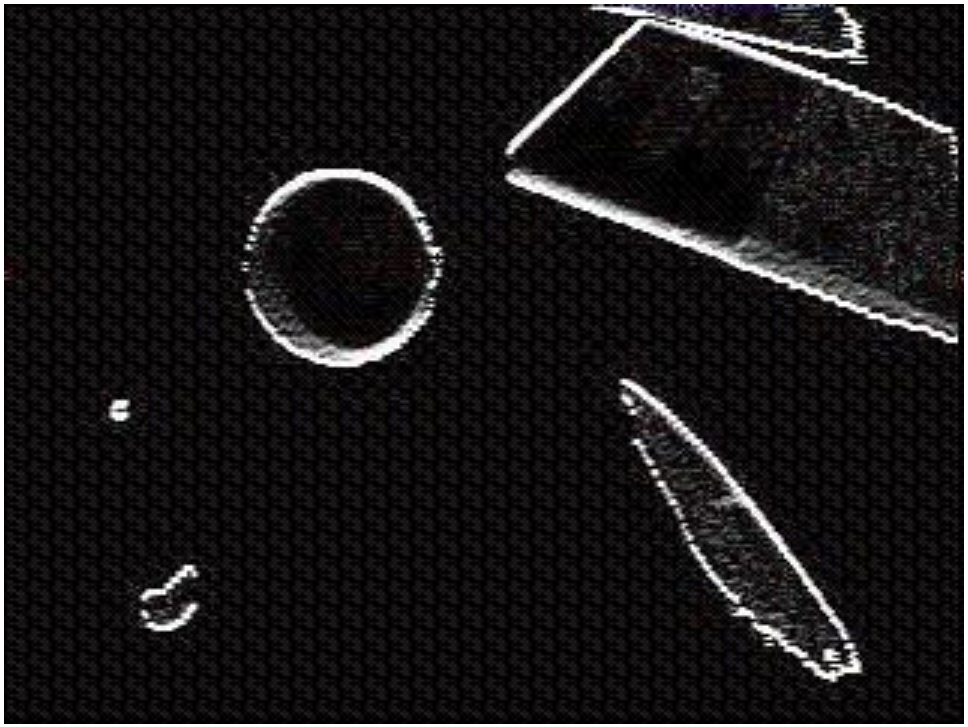
| | | |
|---|---|---|
| -1 | -2 | -1 |
| 0 | 0 | 0 |
| 1 | 2 | 1 |

=

| | | |
|---|---|---|
| | | |
| | $b_{i,j}$ | |
| | | |

$$b_{i,j} = | a_{i+1,j-1} + 2a_{i+1,j} + a_{i+1,j+1} - a_{i-1,j-1} - 2a_{i-1,j} - a_{i-1,j+1} |$$

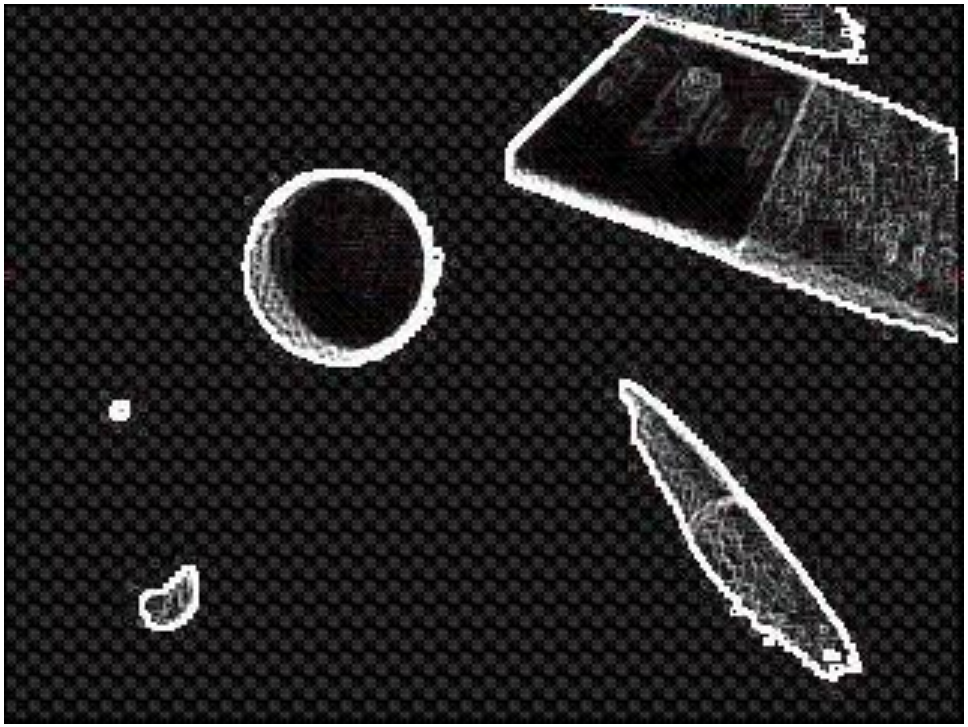
# Sobel operator (horizontal)



dy

We have got nice horizontal edges

# Sobel operator (magnitude)



$|dx+dy|$

or

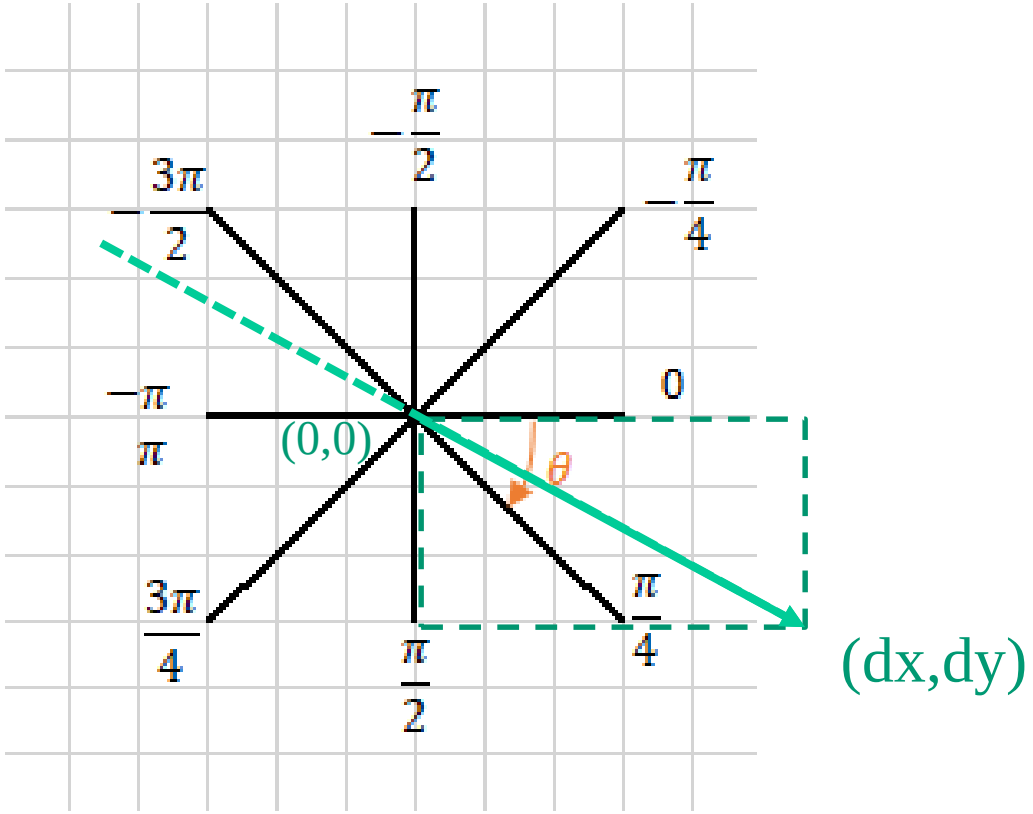
$(dx^2+dy^2)^{½}$

and this is the combined output. This is still grayscale image (0-255), we would like to get binary image (255=edge, 0=other)

# Sobel operator (slope)

- Two ingredients dx and dy provided by Sobel operator indicates also edge orientation

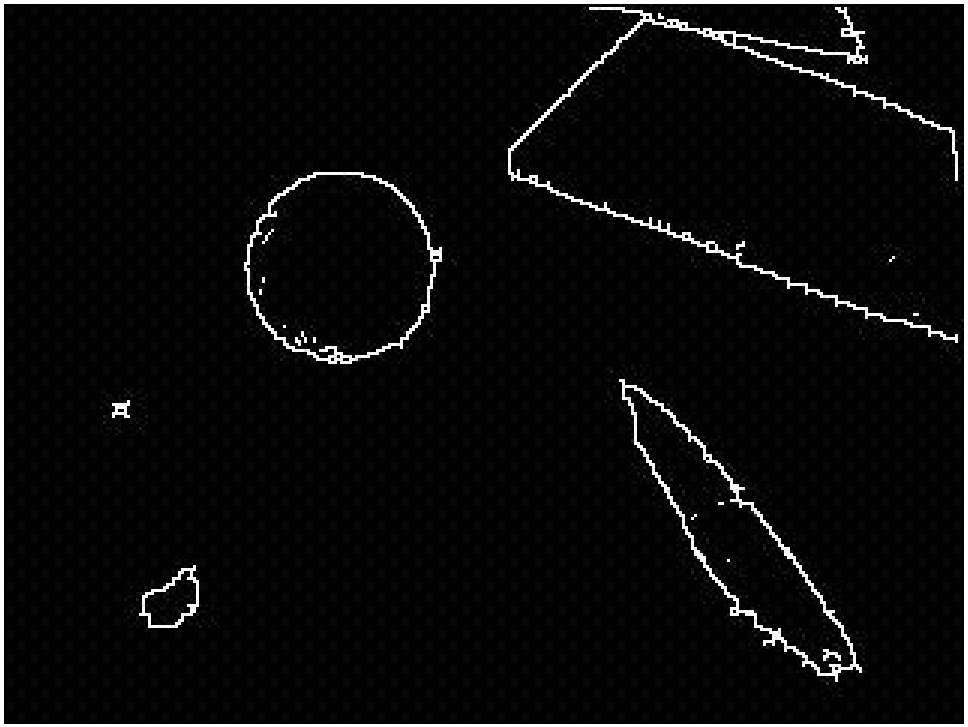
$\theta = [\ arctan(dy/dx)\ ]$



# Canny operator

- binarizes output of Sobel operator
- reduces edges (thinning) by non-maximum suppression (selects pixels with higher intensity than neighborough pixels in direction of the edge slope, magnitude of all others is put to zero)
- applies two thresholds – lower and higher:
  pixel with magnitude over the higher threshold is always edge, pixel with magnitude over the lower threshold is edge if it is in vicinity of pixel with magnitude over the higher threshold (hysteresis)

# Canny operator



- we have got binary image corresponding to edges

# Shape recognition

- How we select edge pixels which form e.g. circle? We can employ:
- Ad-hoc methods
- RANSAC
- *Hough transform*

# Ad Hoc method

- Component analysis of edge
- Test if component (connected set of edge pixels) is circle:
- Find the most left and the most right pixels
- Potential center of circle is their average
- Potential radius is half of their distance
- Test potential center and radius: 80% of rendered potential circle should be covered by actual edge pixels

# RANSAC

- Randomly select three edge pixels
- Calculate potential center radius from them
- Test the potential center and radius:  80% of rendered potential circle should be covered by actual edge pixels
- Repeat that many times
- Return the prevailing circle or no circle found

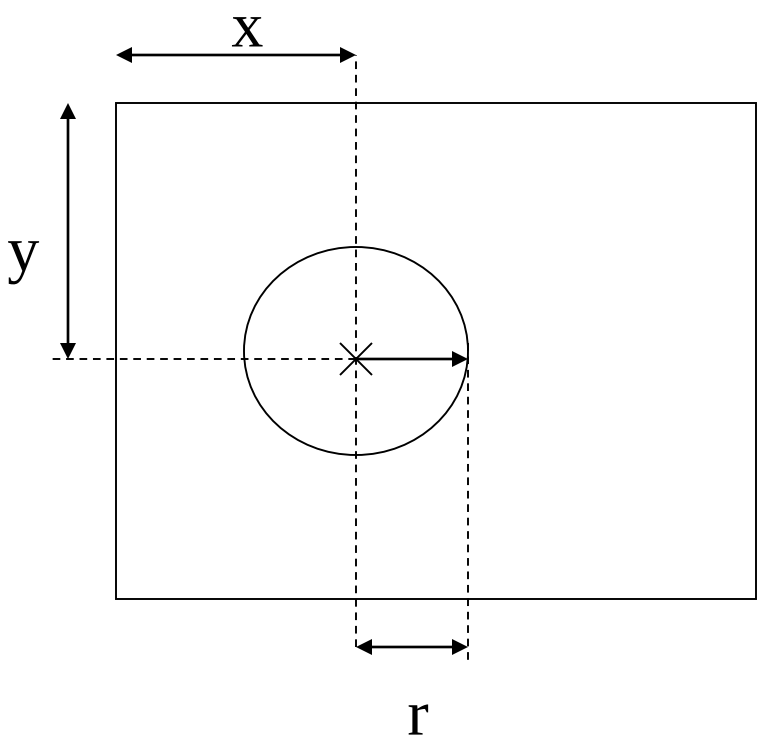
# Hough Transform

- Hough transform is opposite process to drawing a regular object from its parameters
- It projects edge pixels to parameters space

# Hough transform

Circle can be represented by three parameters:

- x-coordinate of center
- y-coordinate of center
- r-radius




having the parameters values, we can draw the corresponding circle, i.e. we can calculate x,y coordinates of its edge pixels

# Hough transform

Each parameter has certain range and resolution

E.g. with camera resolution 320 x 240 pixels

- x-coordinate of center has range 0..319 and resolution 1
- y-coordinate od center has range 0..239 and resolution 1
- radius has range e.g. 10..200, resolution 1

Thus we look for one choice from 320 x 240 x 191

Too much for us, but not for computer

# Hough transform

- For each such choice [x,y,r] we evaluate number of voters P[x,y,r] who vote that:

  *"There is circle with center [x,y] and radius r on the image!"*

- Who are the voters ?

# Hough transform

Voters are the edge pixels. Each such pixel [x,y] vote that on the image:

- There is a circle with center [x,y] and radius 0
- There is a circle with center[x-1,y] and radius 1
- There is a circle with center[x+1,y] and radius 1
- There is a circle with center[x,y-1] and radius 1
- There is a circle with center[x,y+1] and radius 1
- There is a circle with center [x-2,y] and radius 2
- ….

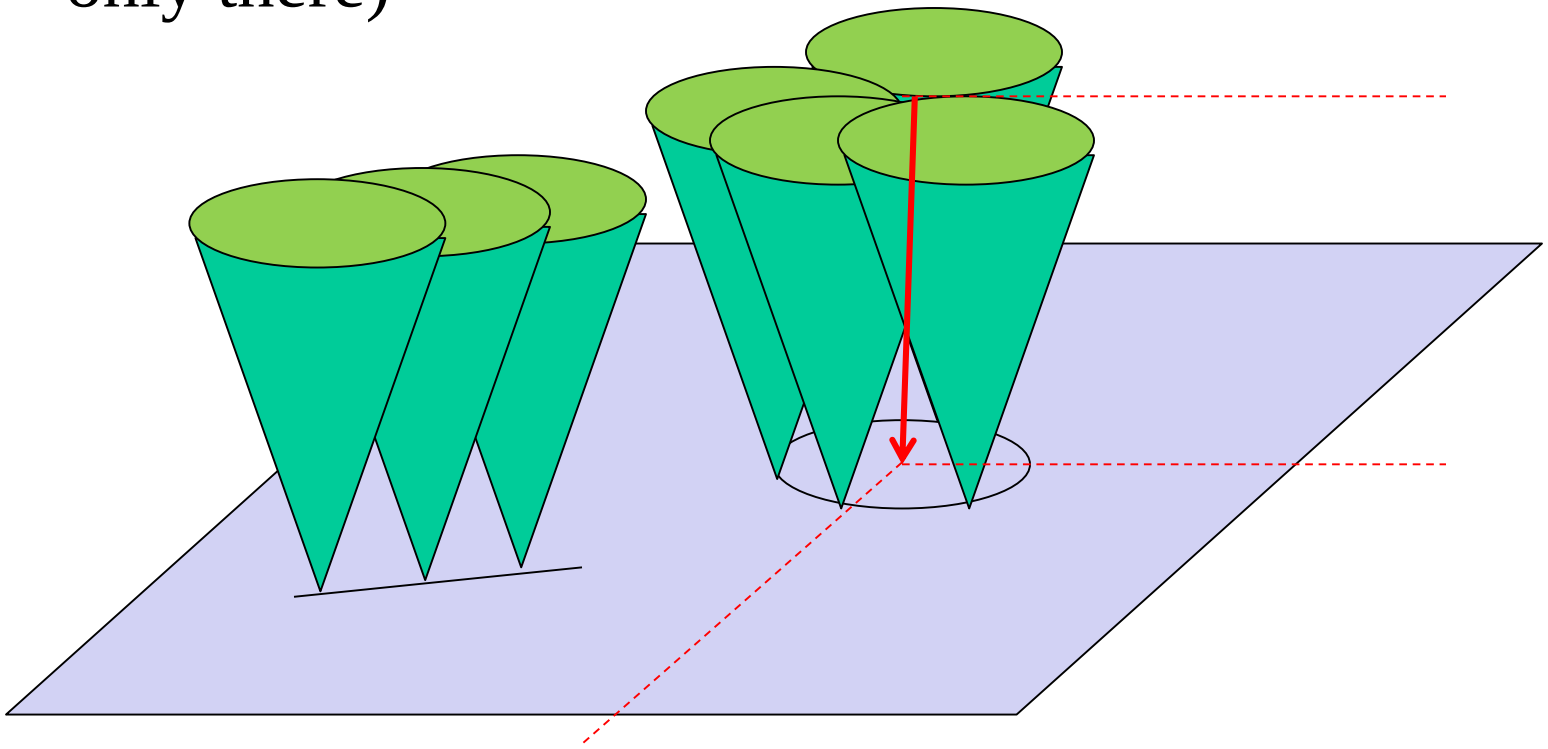
# Hough transform

Voters are the edge pixels. Each such pixel [x,y] vote that on the image:

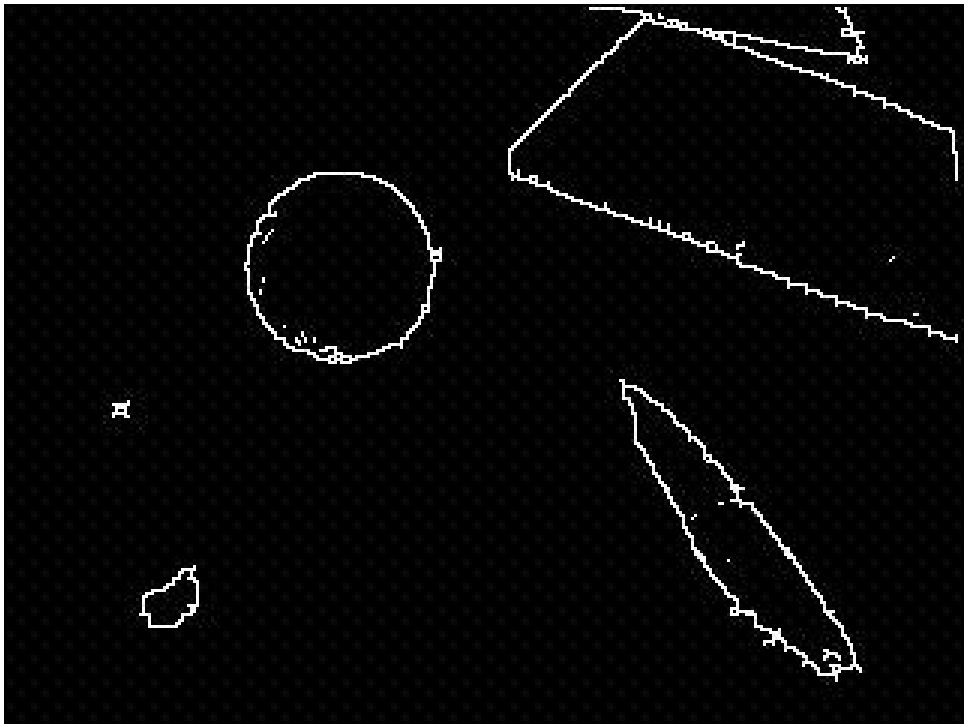
- There is a circle with center [x,y] and radius 0
- There is a circle with center[x-1,y] and radius 1
- There is a circle with center[x+1,y] and radius 1
- There is a circle with center[x,y-1] and radius 1
- There is a circle with center[x,y+1] and radius 1
- There is a circle with center [x-2,y] and radius 2
- …. i.e. for all possible circles which have pixel [x,y]

# Hough transform

The votes of each pixels forms a cone in the three dimensional space [x,y,r] which intersects in the point corresponding to the real parameters (but not only there)

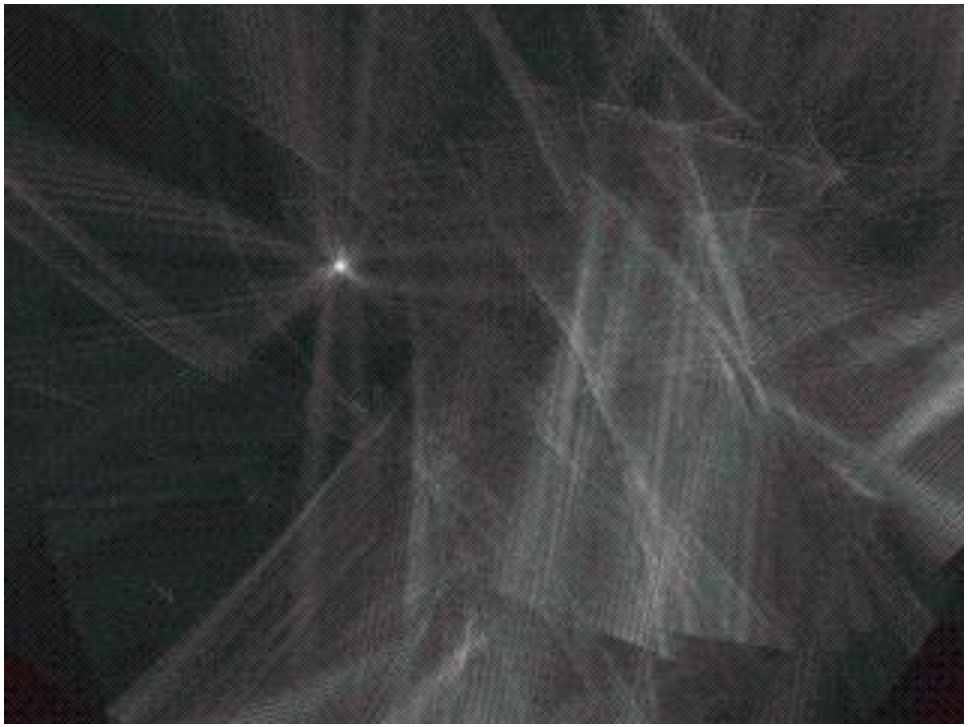


# Hough transform



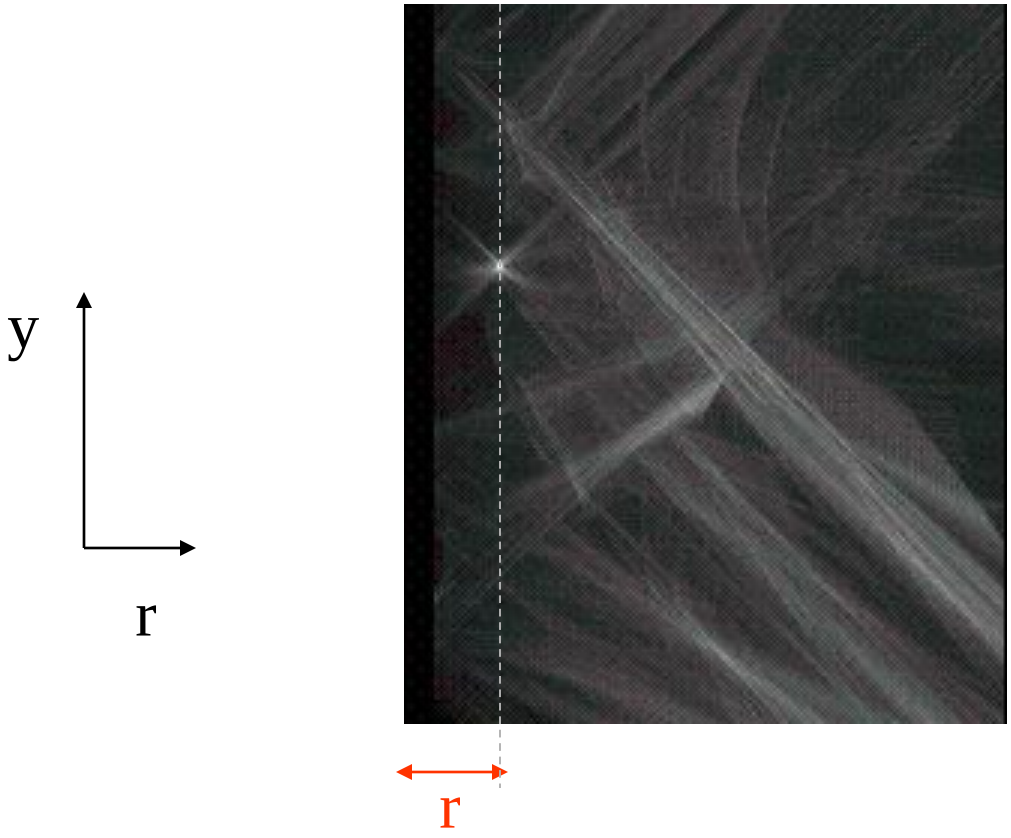
Though majority of votes is false, still the highest number of votes is assigned to the real parameters!

# Hough transform



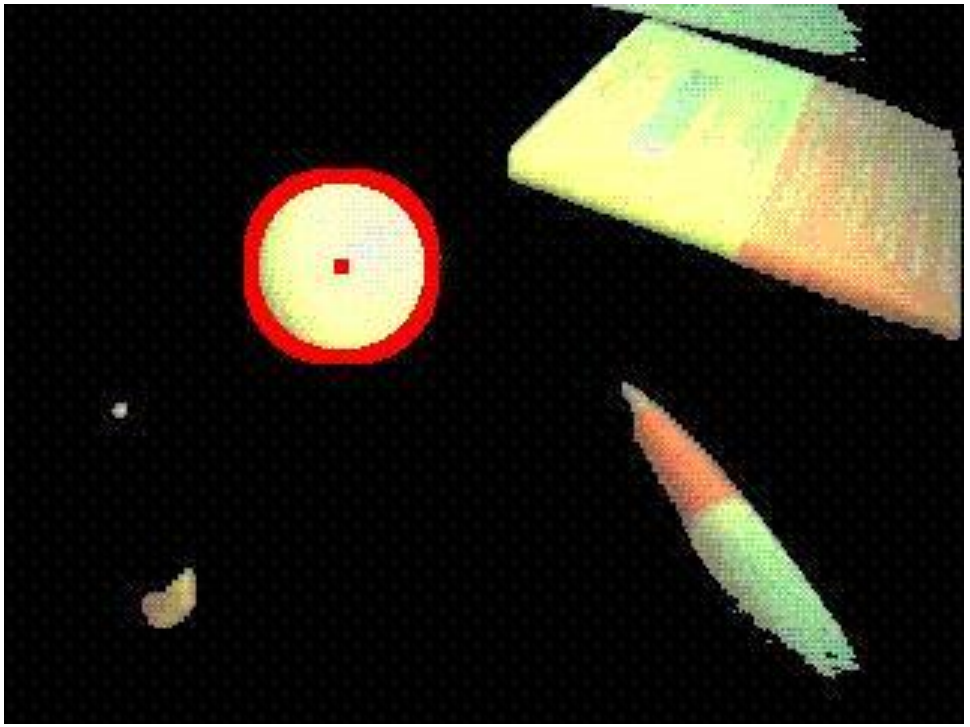

So when we look to the 3D space, we can detect the correct center

# Hough transform



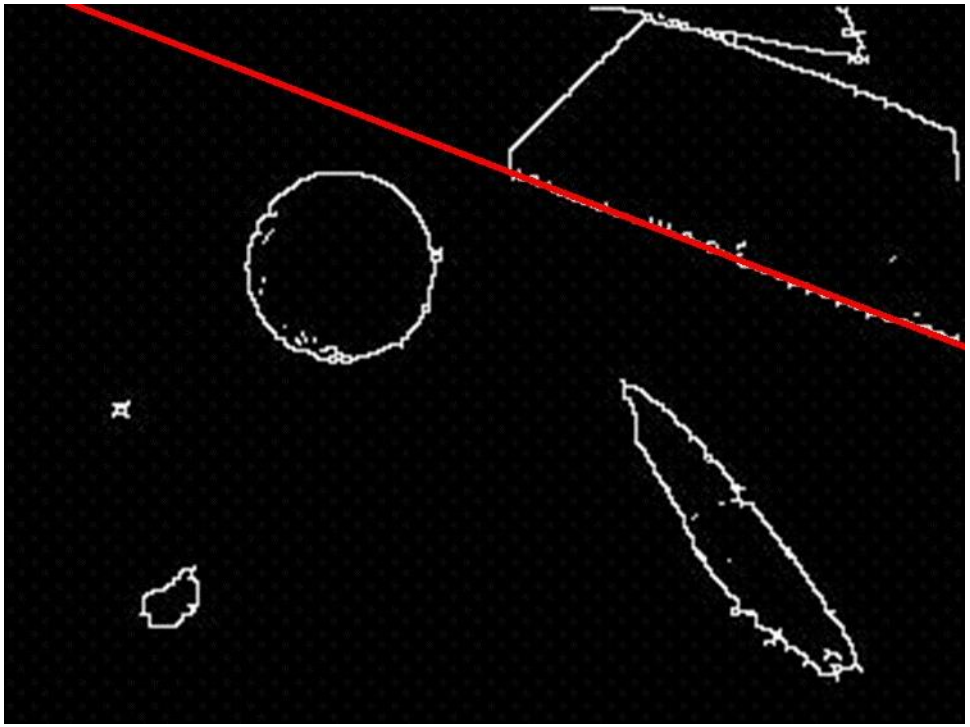

and from other view we can see the correct radius

# Hough transform



... Thus we have identified the circle by shape

# Hough transform (lines)

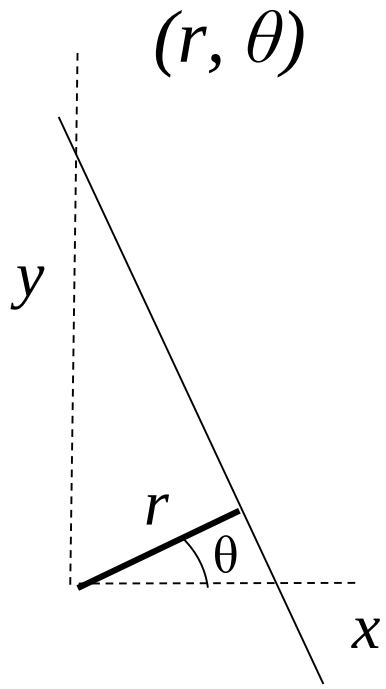


Analogically we can detect lines (2 parameters), ellipses (4 parameters) ,…

# Edge detection (Canny)

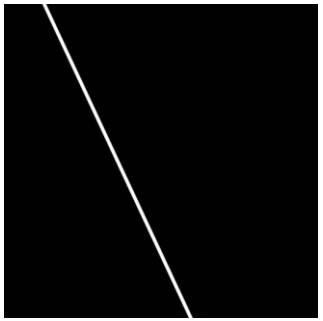
# Hough transform

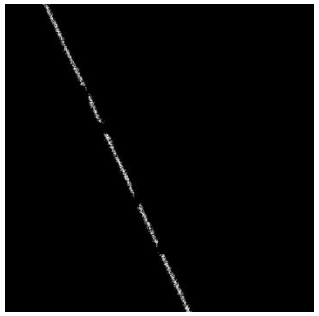
$$r = x \cos\theta + y \sin\theta$$




Rendering

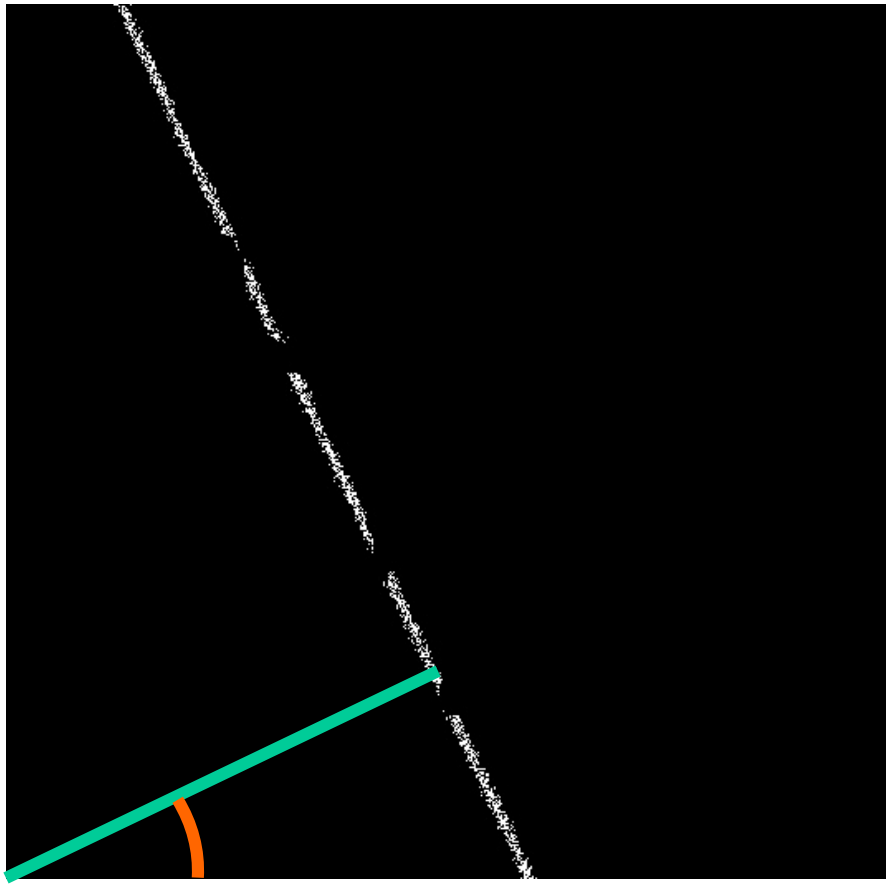
Hough transform



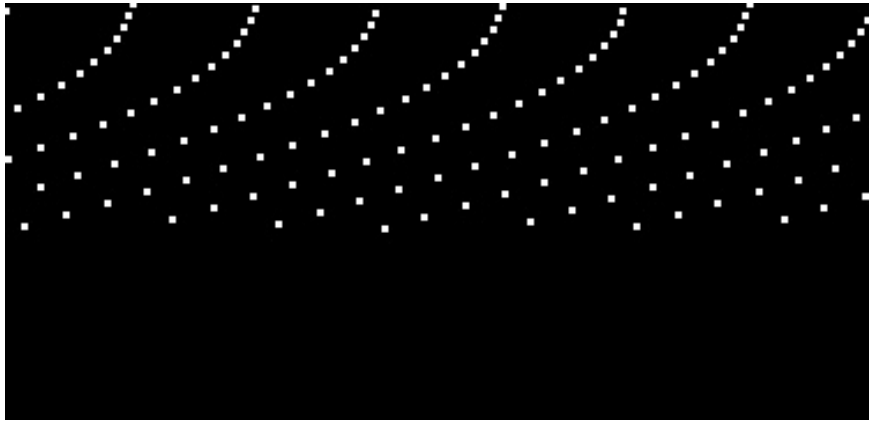


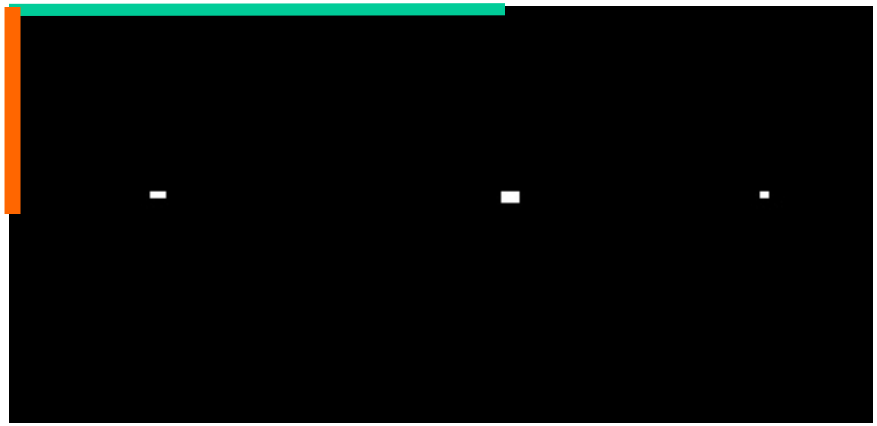
# Hough transform

voters



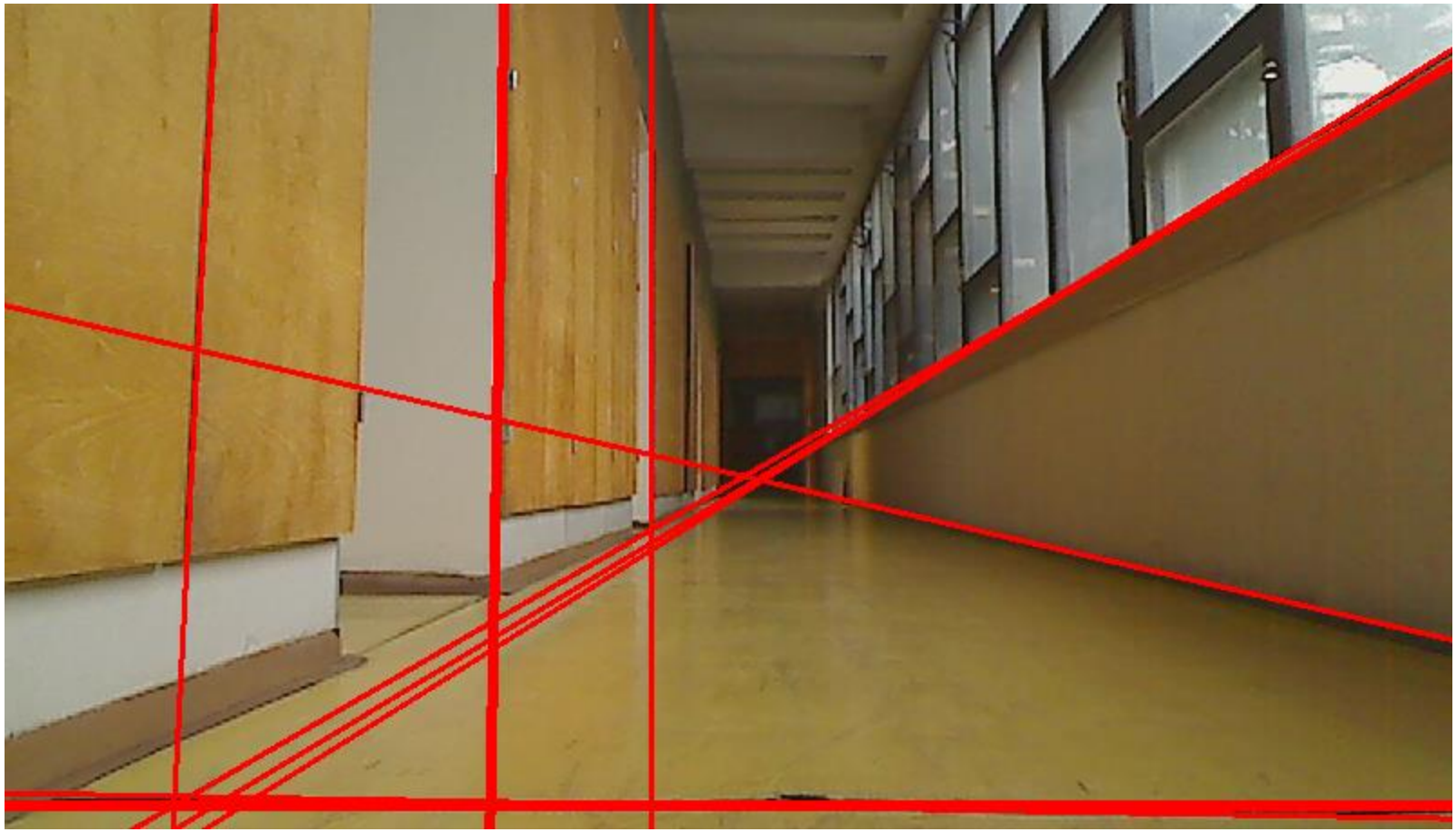
votes of one voter





votes summary

# Hough transform - lines

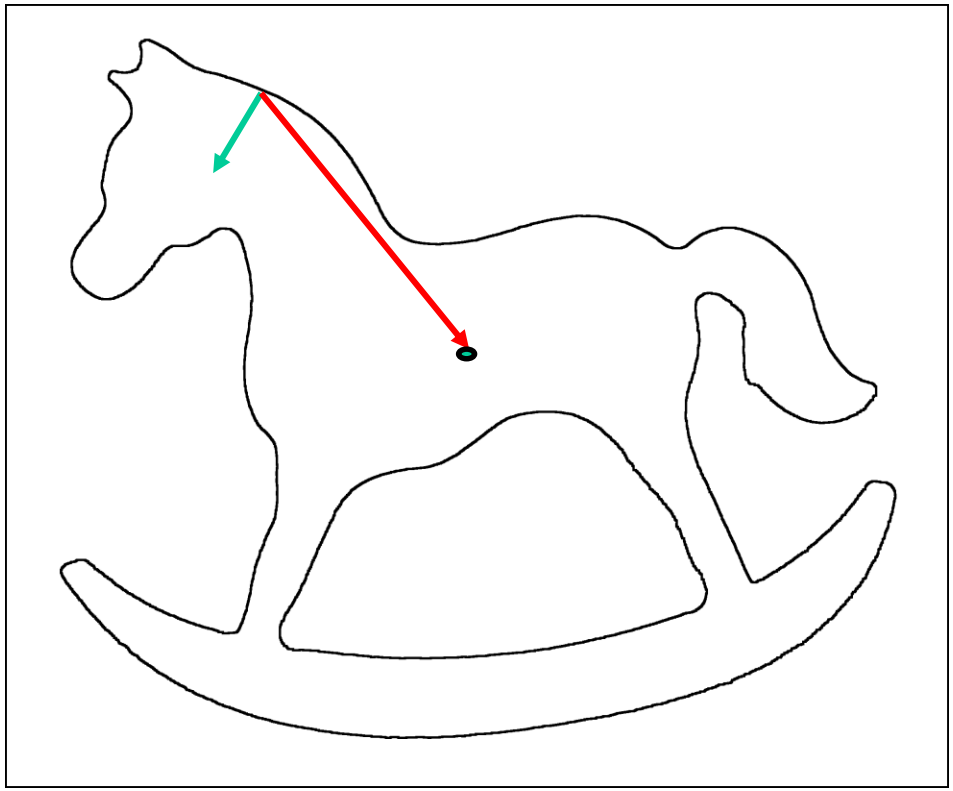


# Hough transform - segments

# Vanishing point detection

# Vanishing point detection

# Generalized Hough transform



Analogically we can detect an object with predefined shape (x, y and scale parameters)