# Introduction to Robotics for cognitive science

**Dr. Andrej Lúčny**

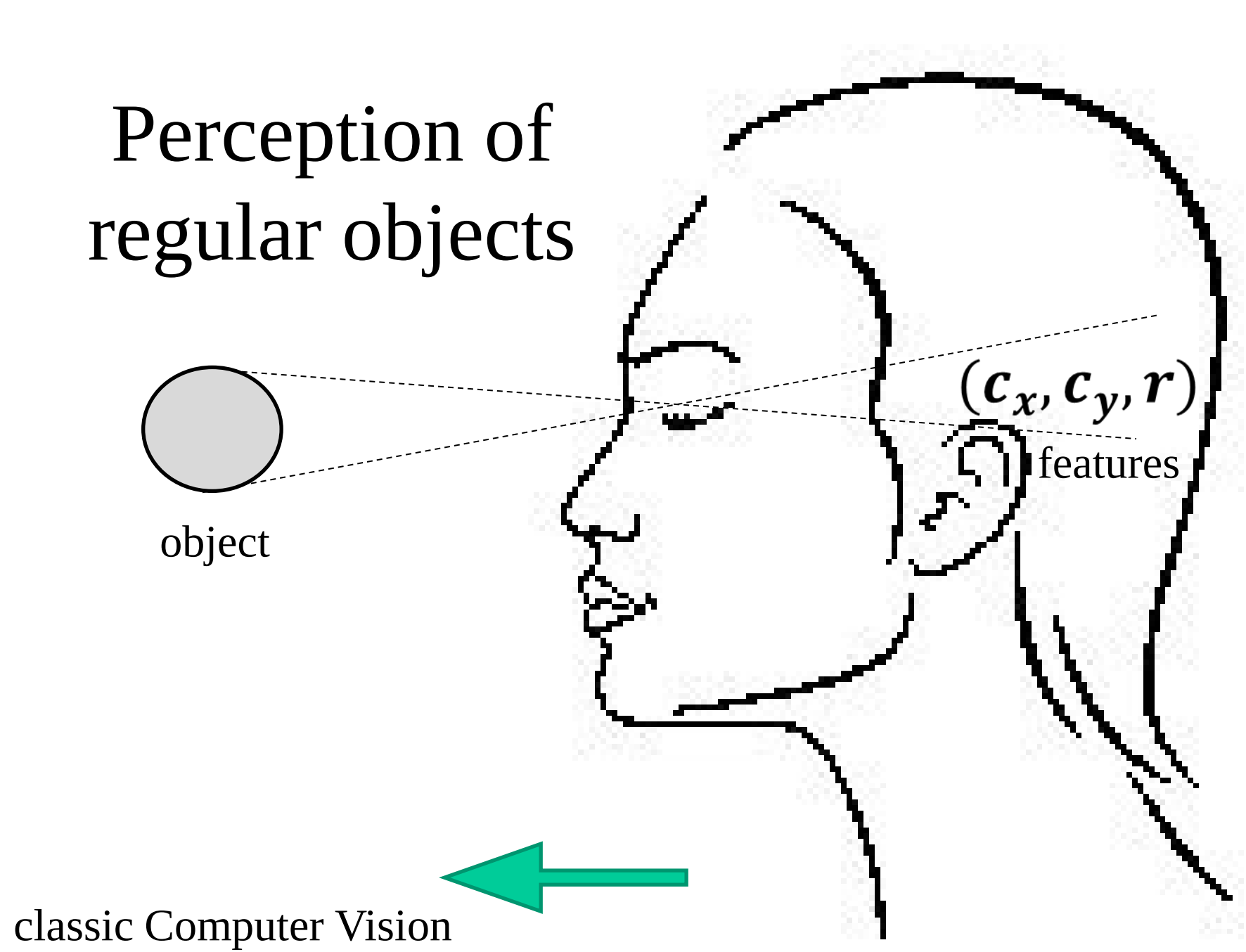
**KAI FMFI UK**

**lucny@fmph.uniba.sk**

# Web page of the subject

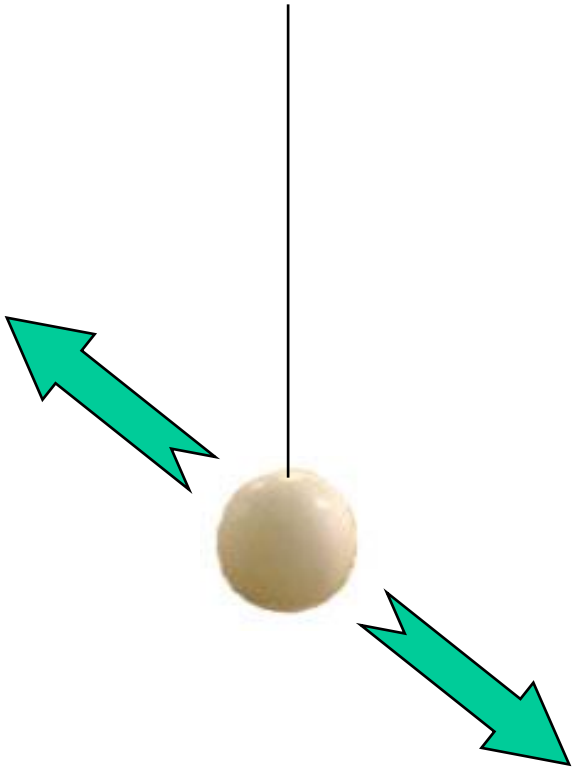
www.agentspace.org/kv

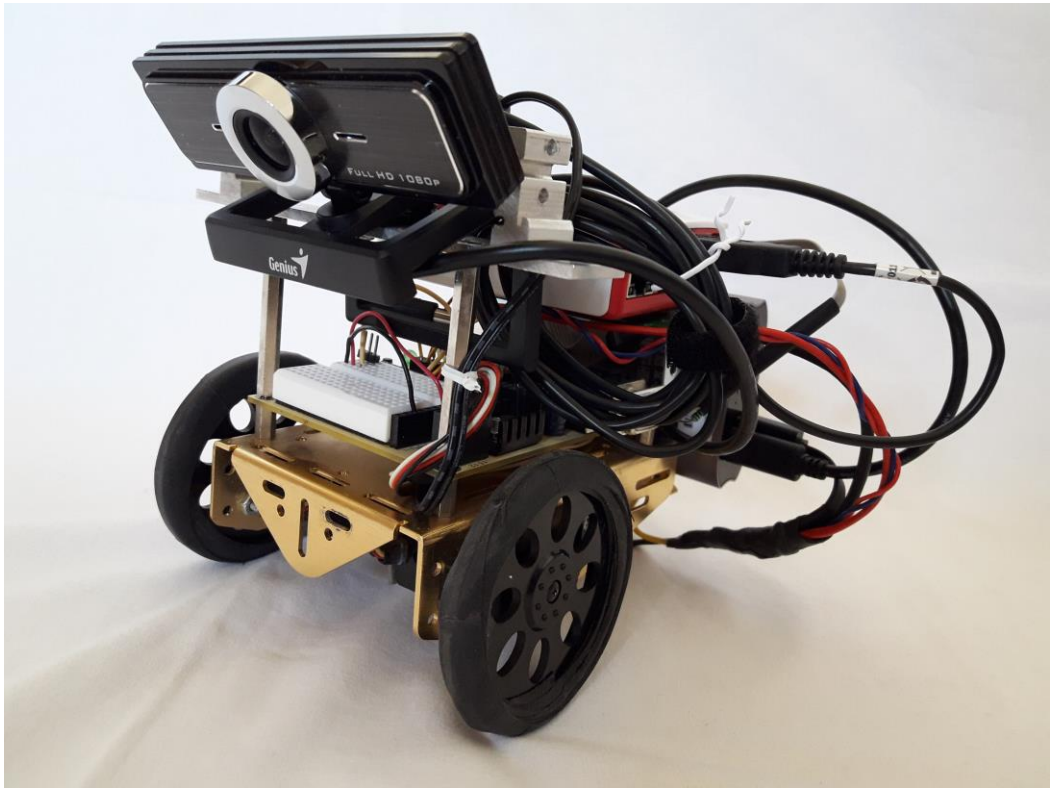
# Perception of regular objects



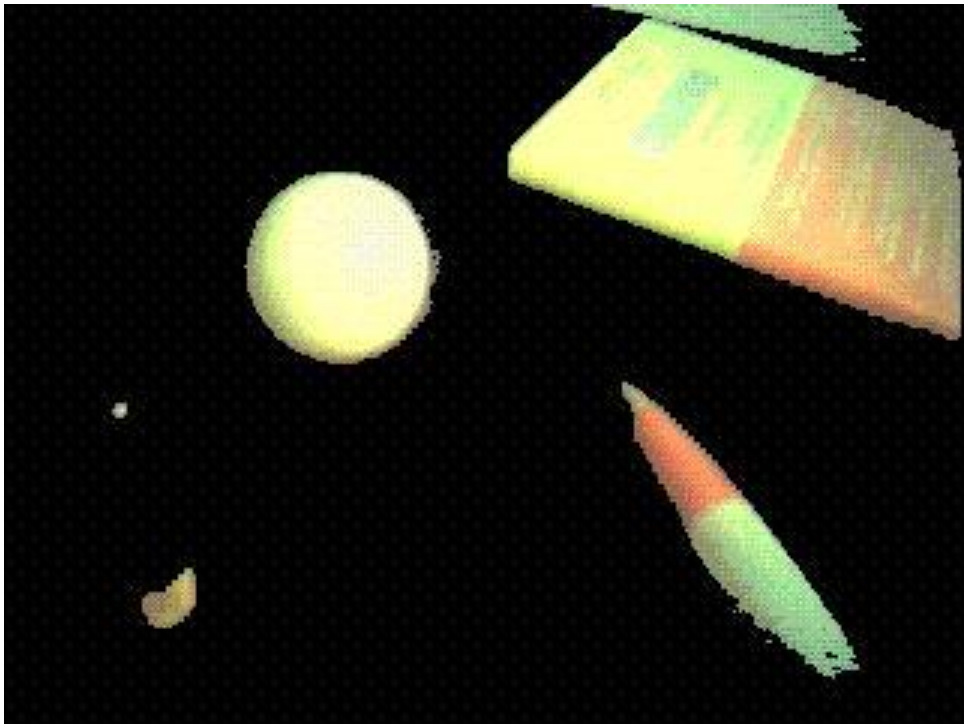

classic Computer Vision
no machine learning

# Robot following ball





# The camera provides color images



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 the center of the ball circle and *r* is its radius

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

# Grayscale image

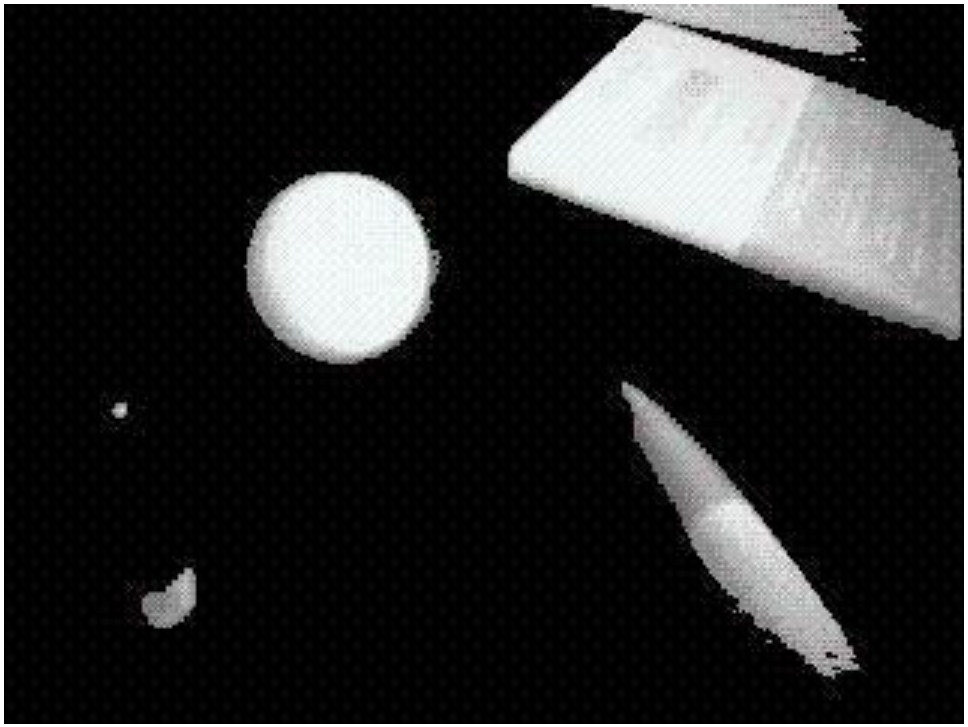
- Turning color images (height, width, 3) to grayscale (height, width)
- by numpy:

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

by openCV:

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

# Grayscale image

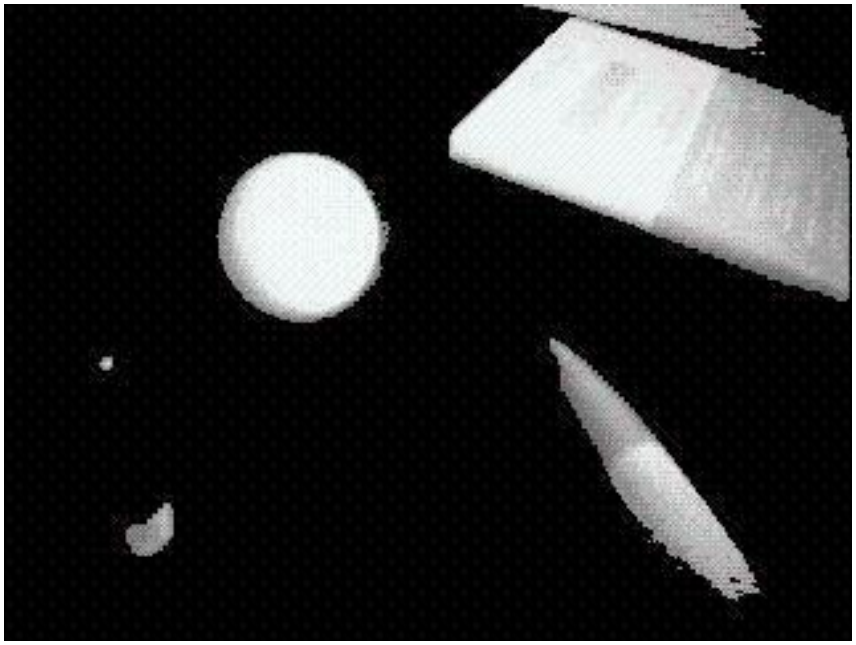


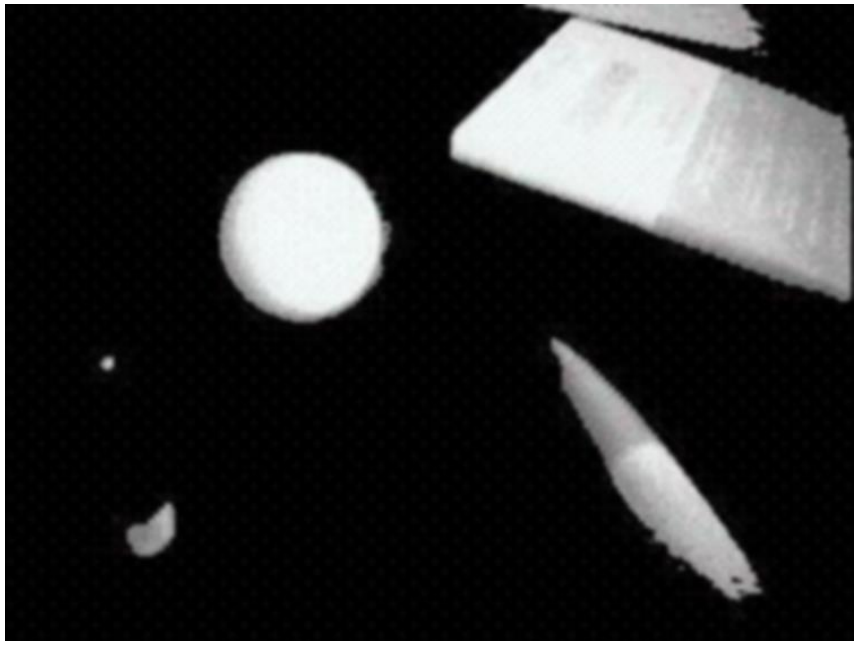
2D array of intensities 0..255

# Blur (noise reduction)

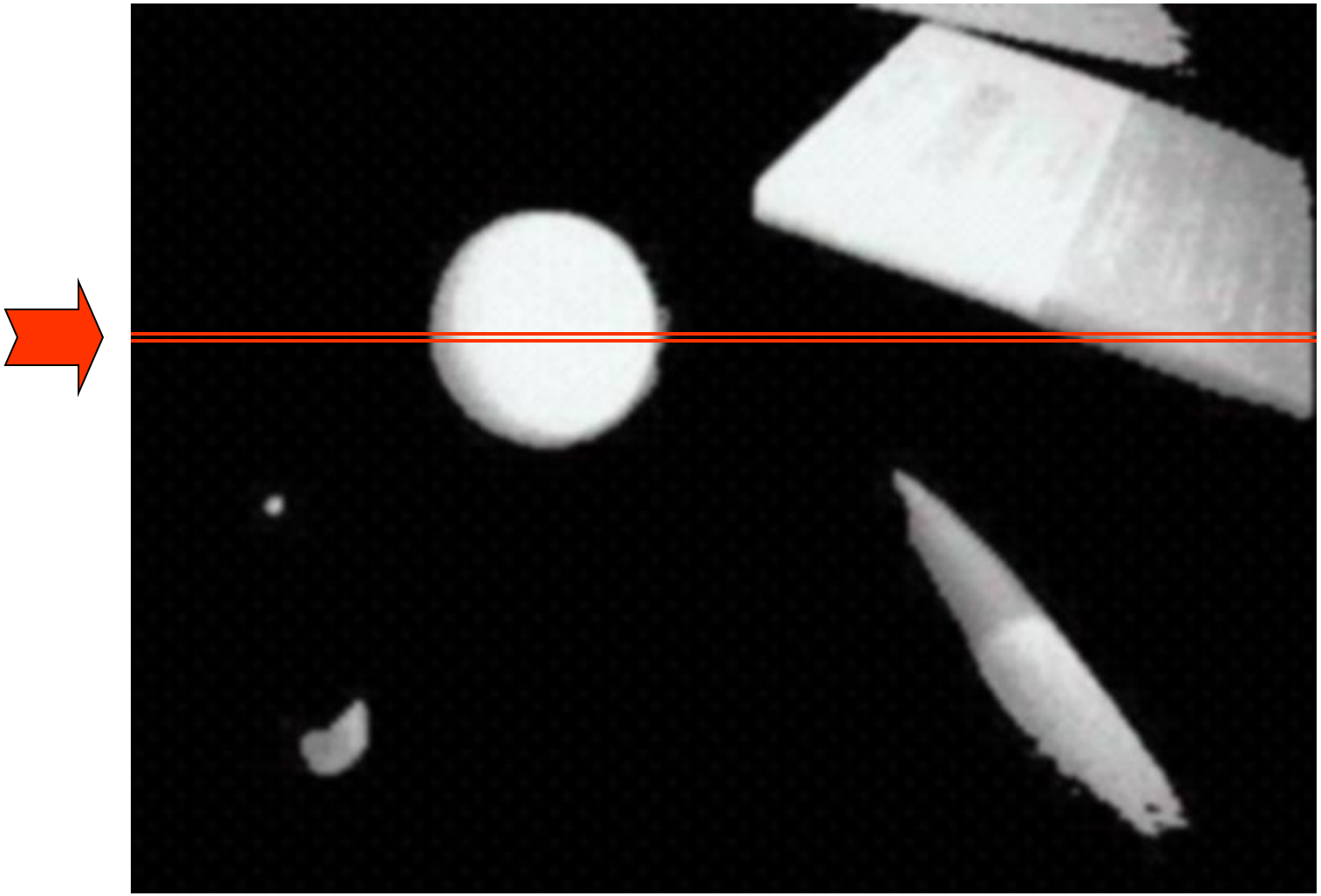
blur kernel 3x3

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

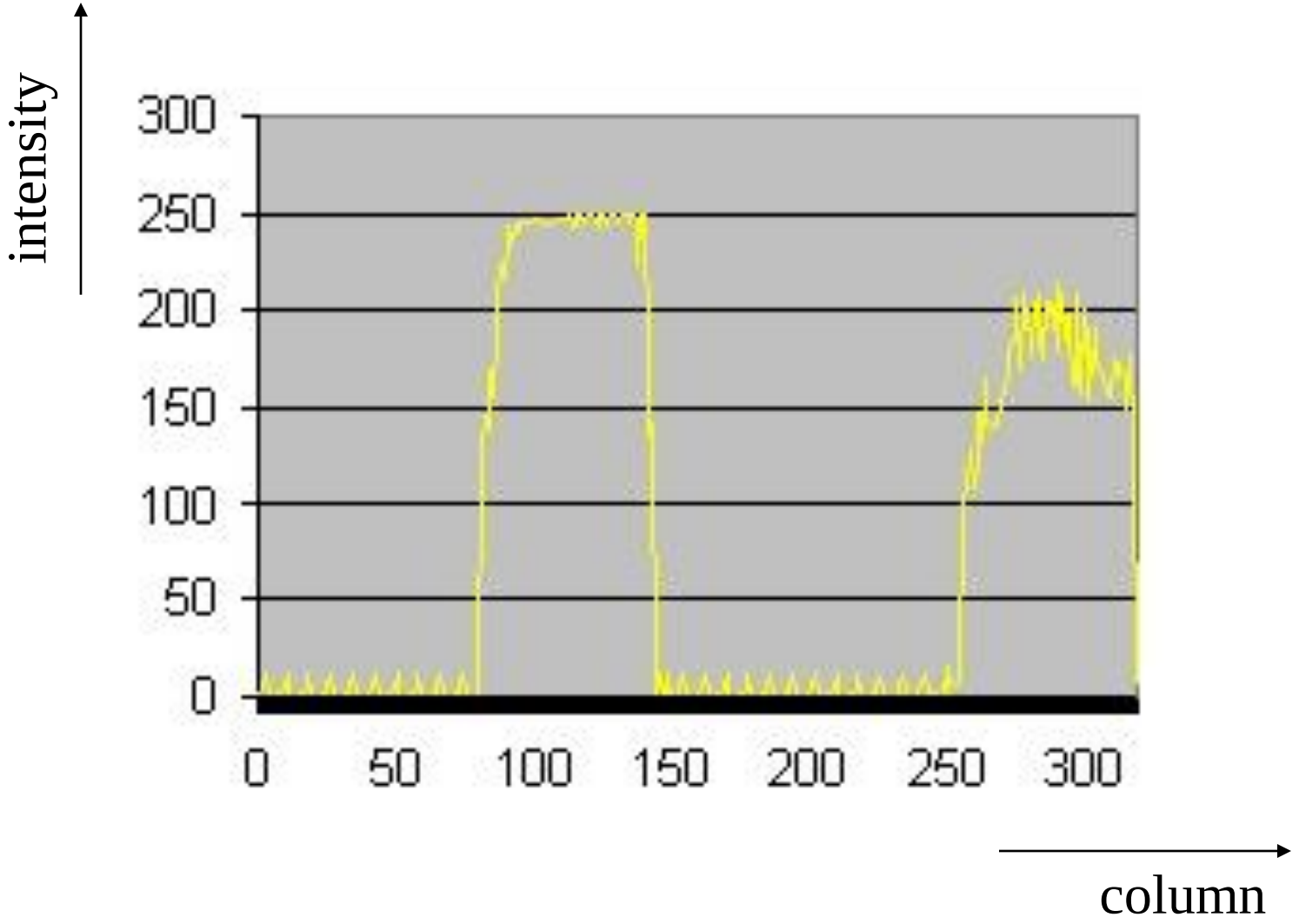




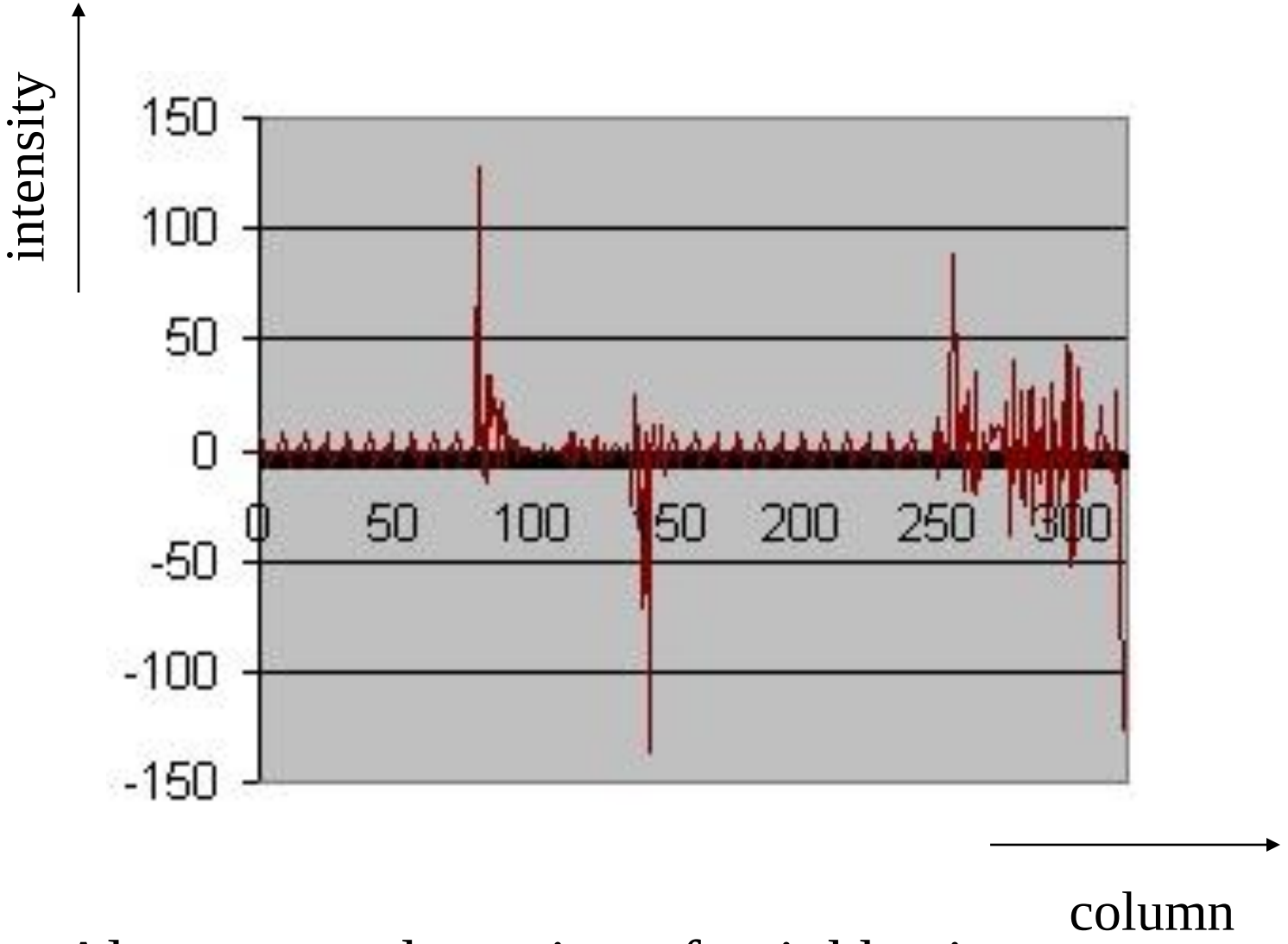
```
frame = cv2.blur(frame,(3,3))
```
or
```
frame = cv2.GaussianBlur(frame,(5,5),0)
```



- Let us consider one line as a function of the column




- Edges correspond to signitficant changes. How do we filter them from the rest?




- Also, pure subtraction of neighboring pixels gives relatively good edges

# Sobel kernel (vertical)

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

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

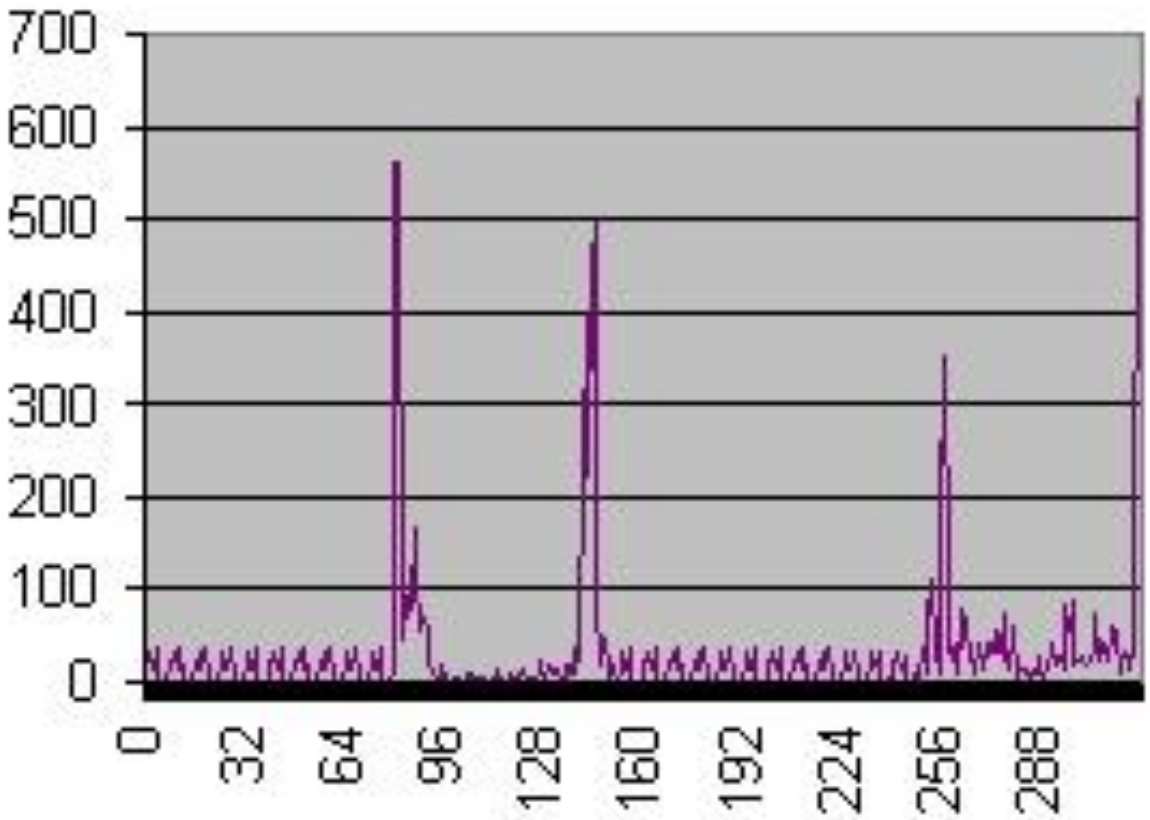
o

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

=

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

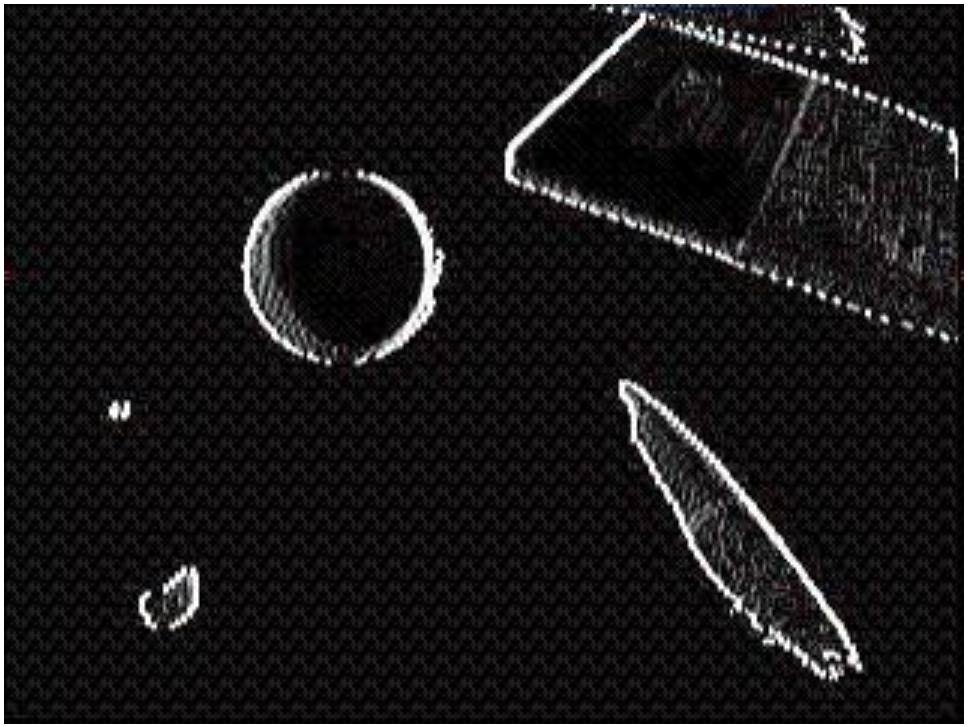
$$b_{i,j} = | a_{i-1,j+1} + 2a_{i,j+1} + a_{i+1,j+1} - a_{i-1,j-1} - 2a_{i,j-1} - a_{i+1,j-1} |$$




Yes, it works better than the pure subtraction

Let us look at 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

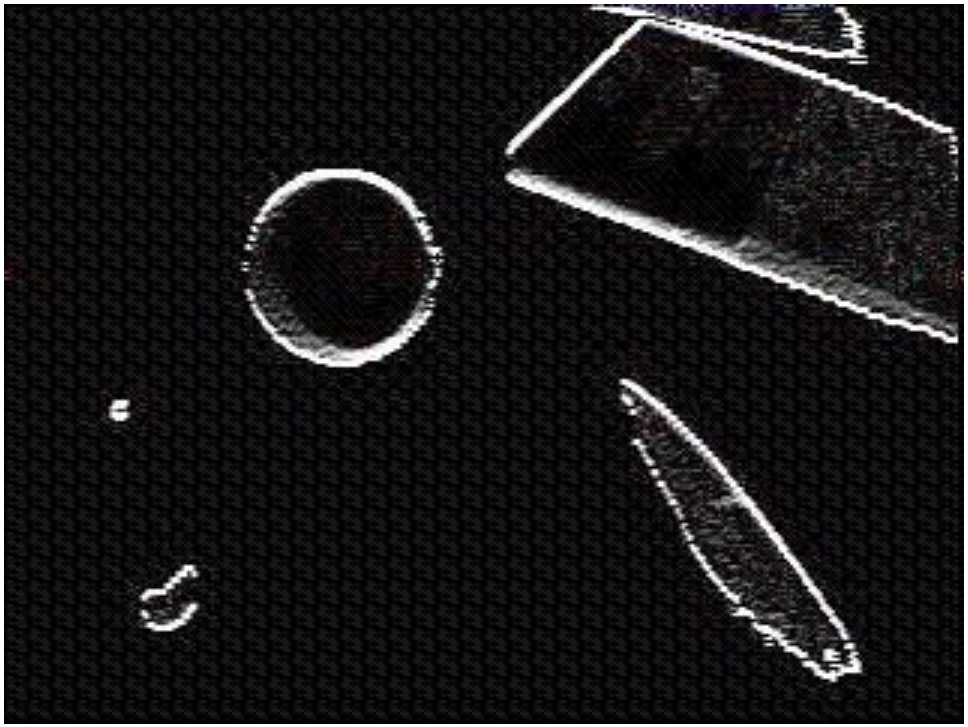
| | | |
|---|---|---|
| -1 | -2 | -1 |
| 0 | 0 | 0 |
| 1 | 2 | 1 |

=

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

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

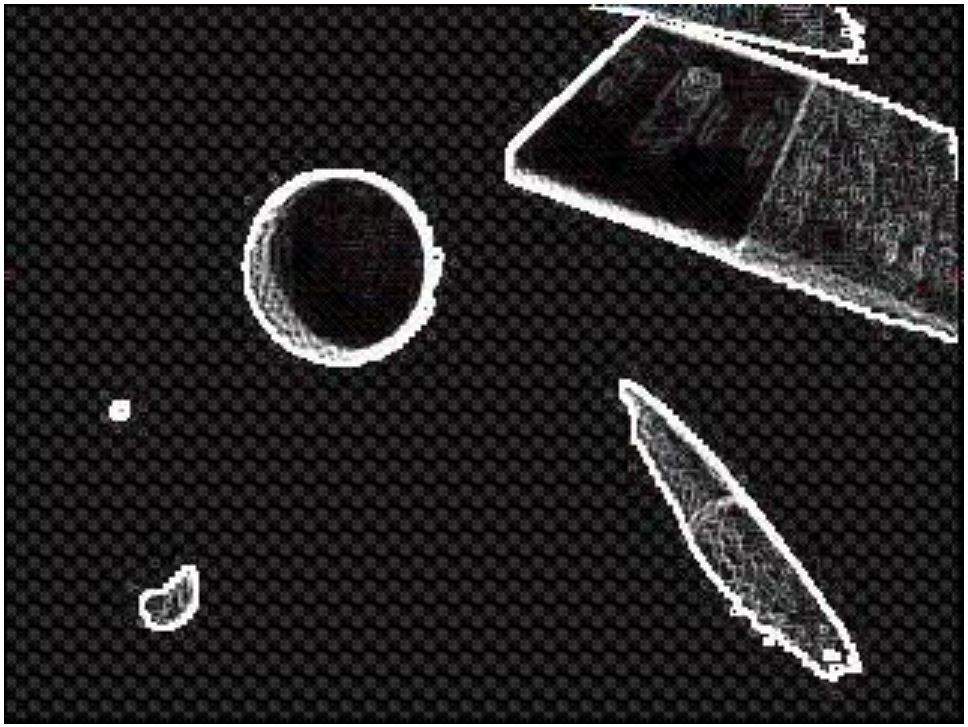
# Sobel operator (horizontal)



dy

We have got nice horizontal edges,

# Sobel operator (magnitude)



$|dx+dy|$

or

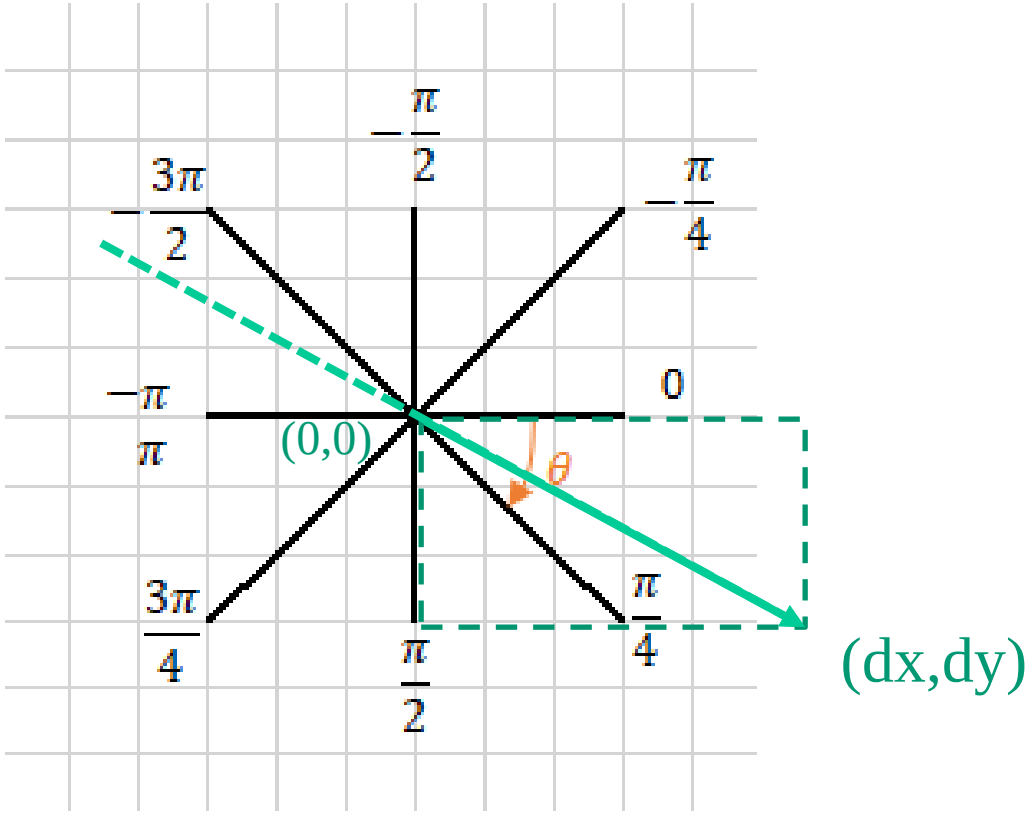
$(dx^2+dy^2)^{½}$

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

# Sobel operator (slope)

- dx and dy - two ingredients provided by the Sobel operator - also indicate the edge orientation

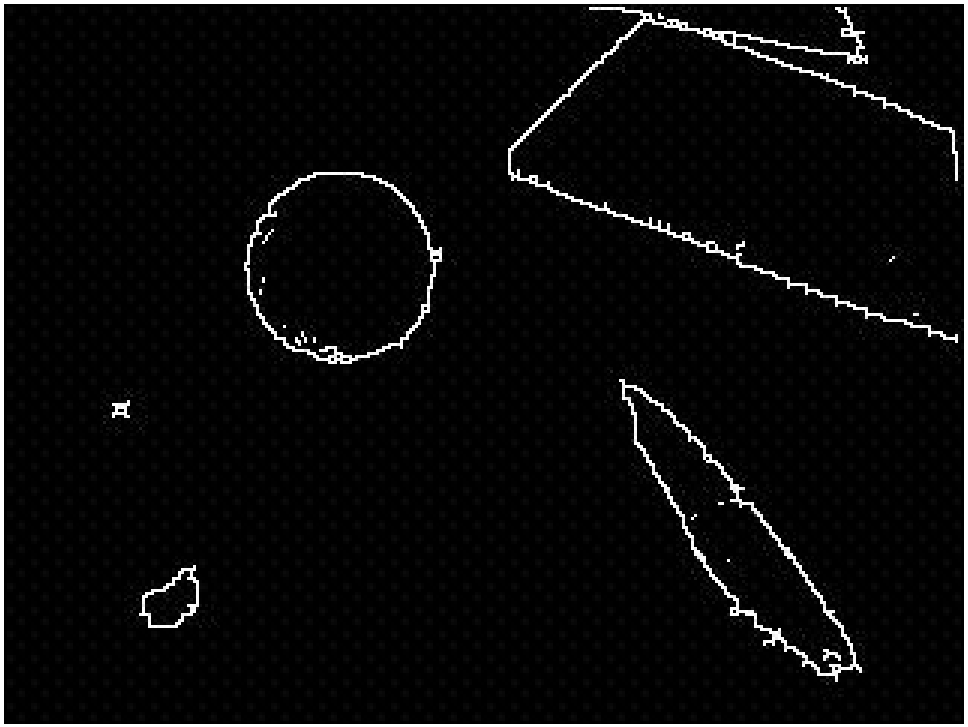
$\theta = [\ \arctan(dy/dx)\ ]$



# Canny operator

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

# Canny operator



- we have a binary image corresponding to the edges

# Shape recognition

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

# Ad Hoc method

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

# RANSAC

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

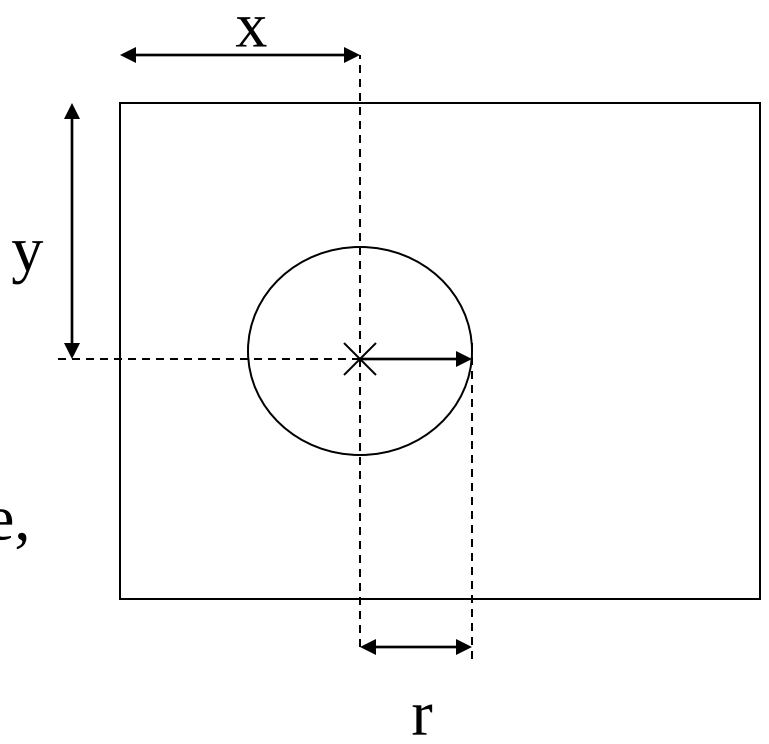
# Hough Transform

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

# Hough transform

We can represent a circle by three parameters:

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




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

# Hough transform

Each parameter has a range and resolution

- E.g., with a camera resolution of 320 x 240 pixels
- The x-coordinate of the center has a range of 0..319 and a resolution of 1
- The y-coordinate of the center has a range of 0..239 and a resolution of 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 the computer

# Hough transform

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

  *“There is a 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] votes 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
- ….

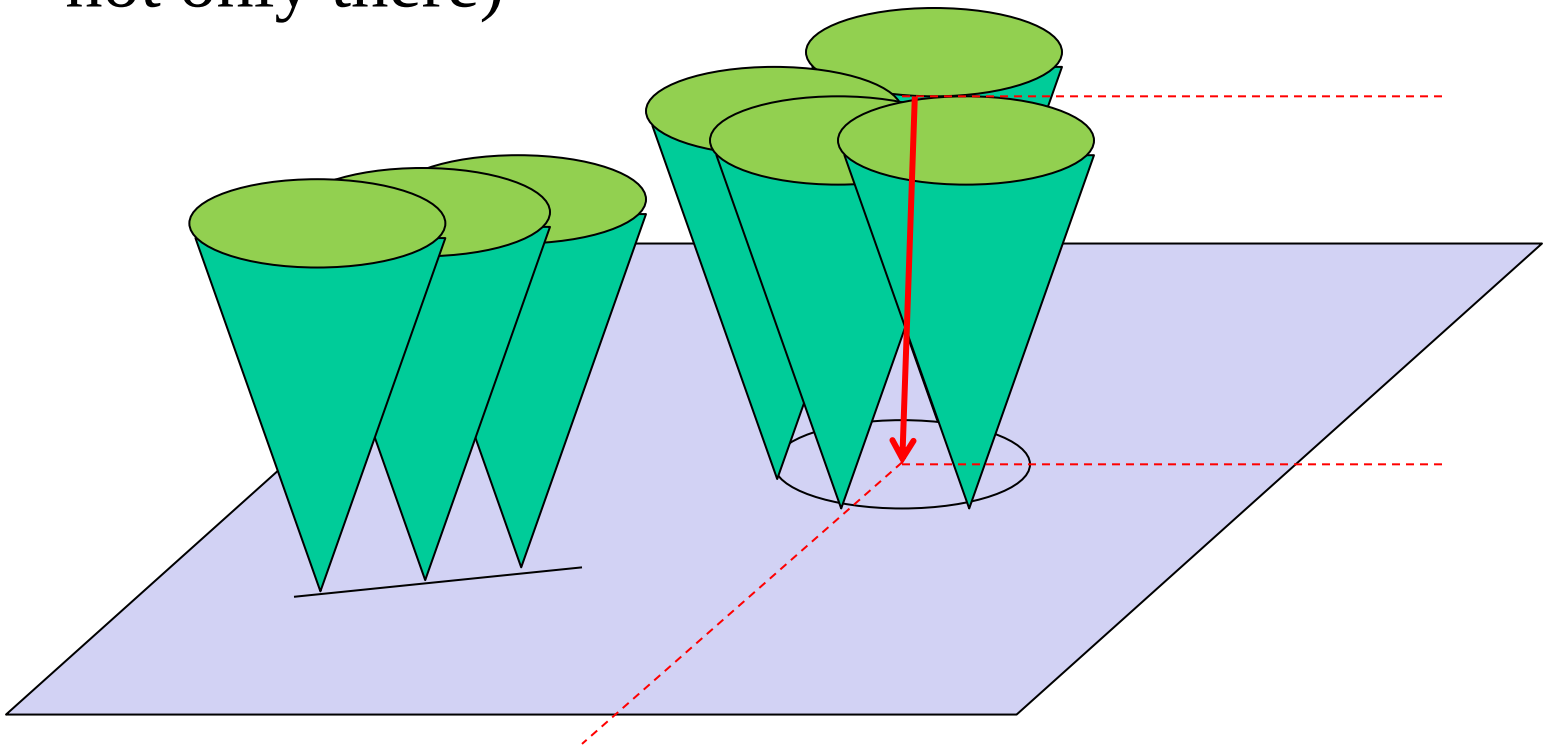
# Hough transform

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

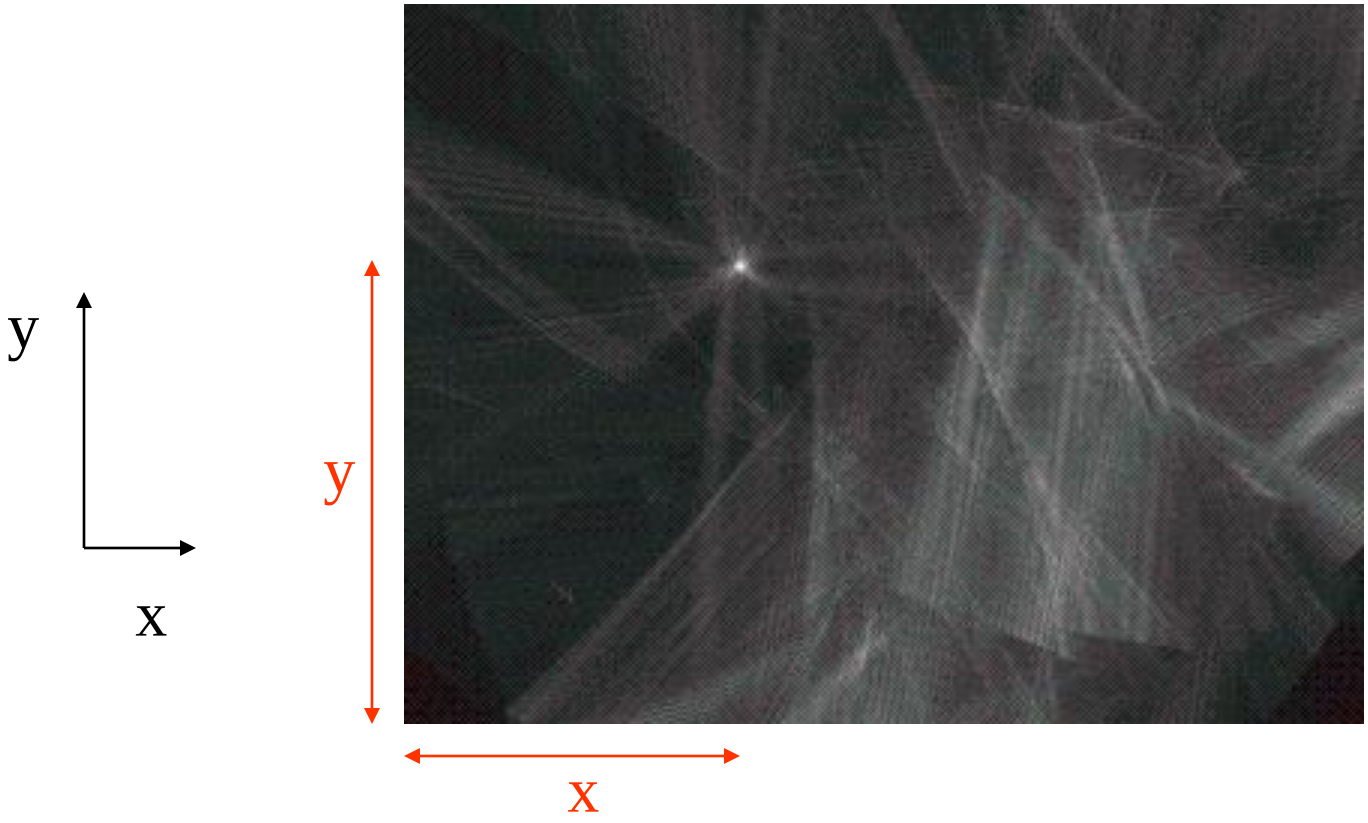
- There is a circle with center [x,y] and radius 0
- There is a circle with center[x-1,y] and radius 1
- There is a circle with center[x+1,y] and radius 1
- There is a circle with center[x,y-1] and radius 1
- There is a circle with center[x,y+1] and radius 1
- There is a circle with center [x-2,y] and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

# Hough transform

The votes of each pixel form a cone in the three-dimensional space [x,y,r]. These cones intersect at the point corresponding to the actual parameters (but not only there)

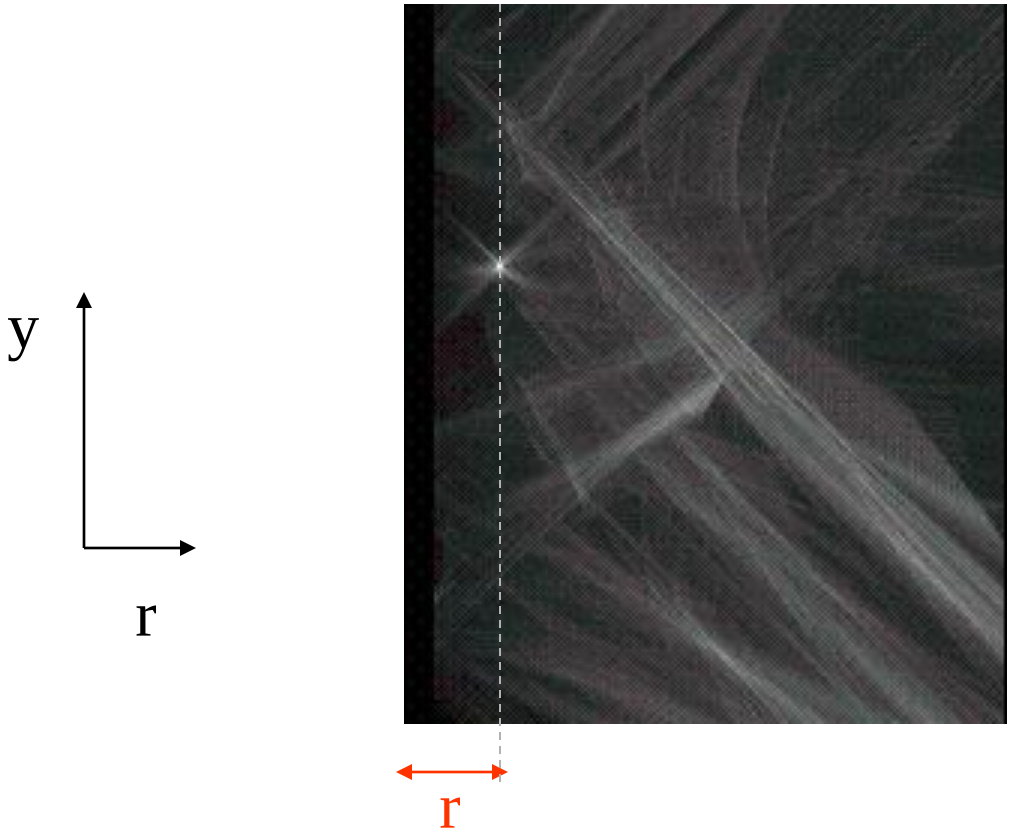


Though most votes are false, the highest number is still assigned to the actual parameters!



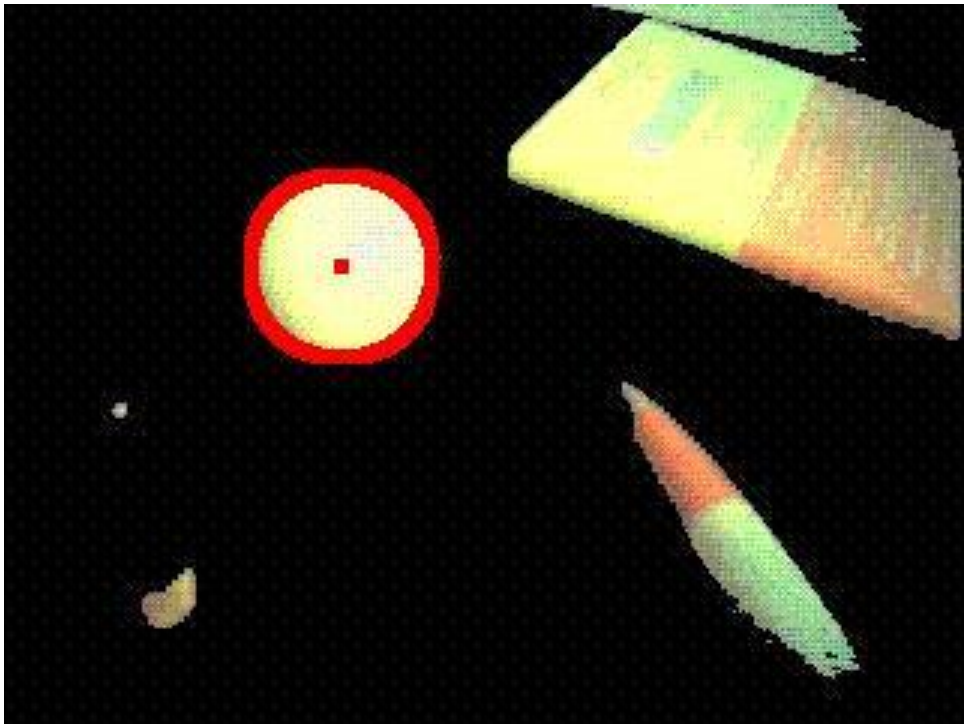

So, when we look at the 3D space, we can detect the correct center,

# Hough transform



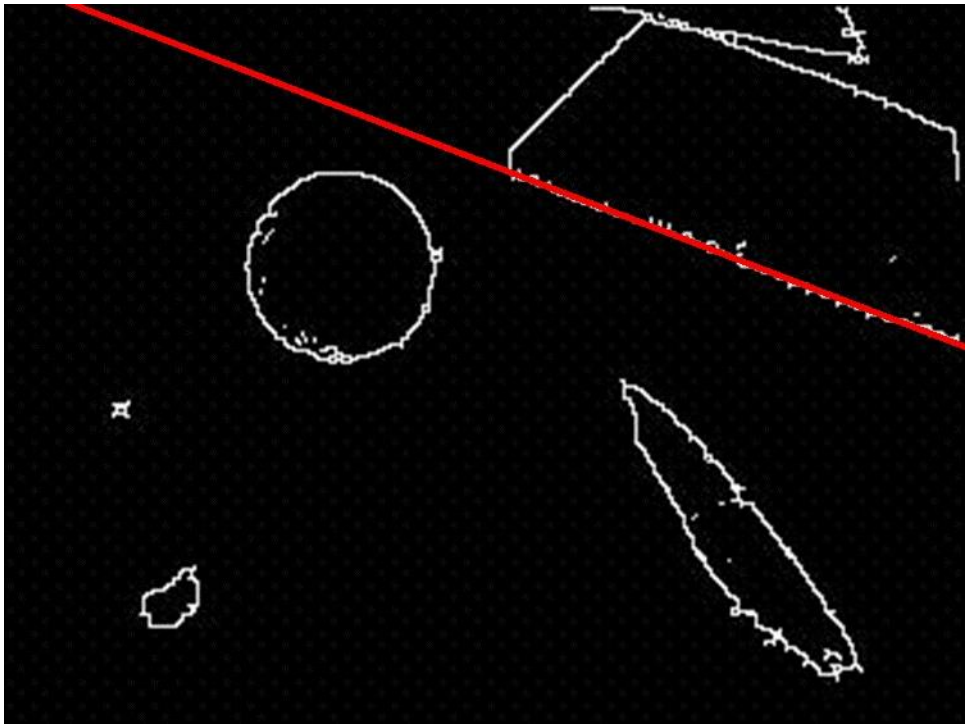

and from another view, we can see the proper radius.

# Hough transform



... Thus, we have identified the circle by shape

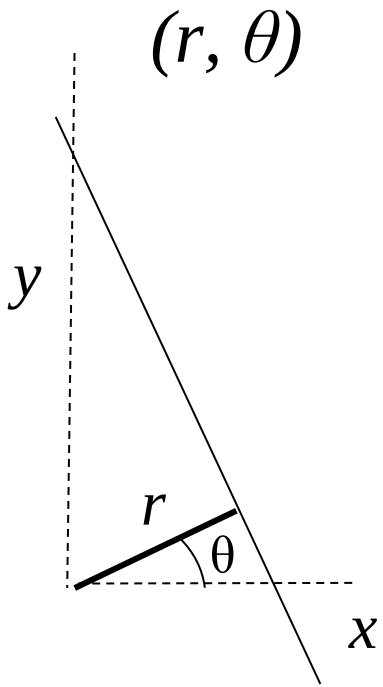
# Hough transform (lines)



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

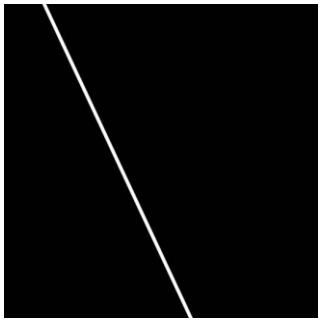
# Hough transform

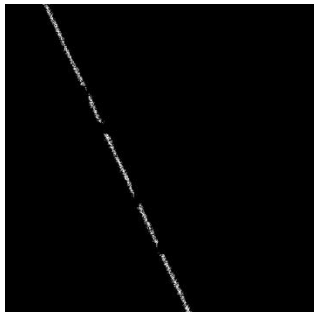
$$r = x \cos\theta + y \sin\theta$$




Rendering

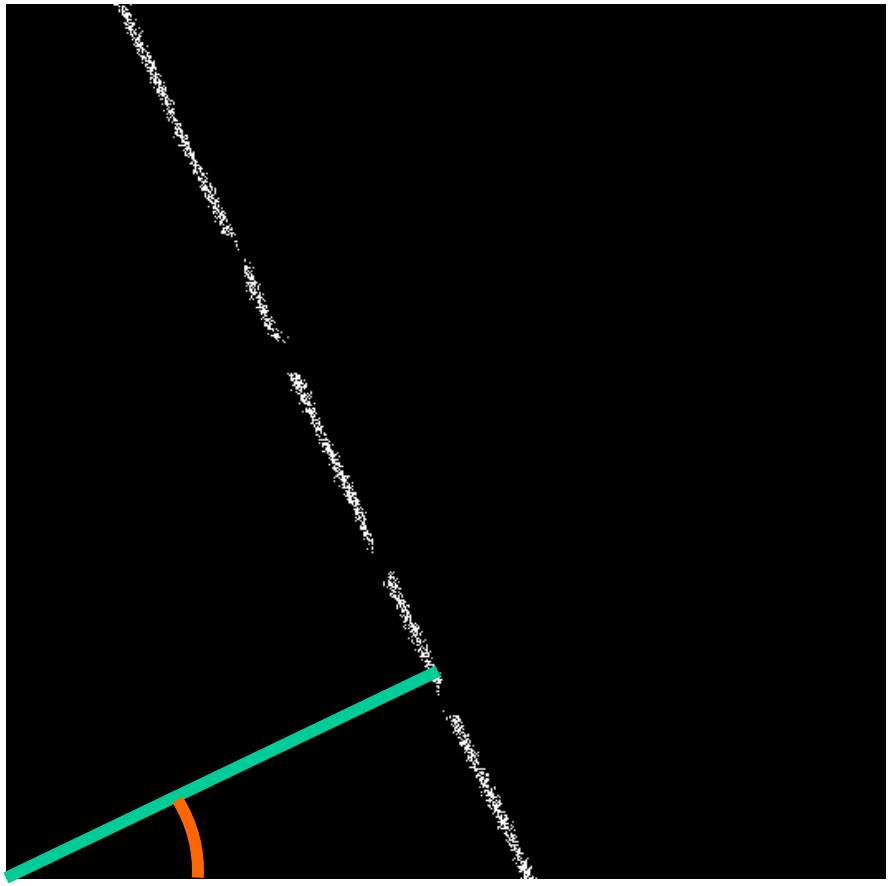
Hough transform



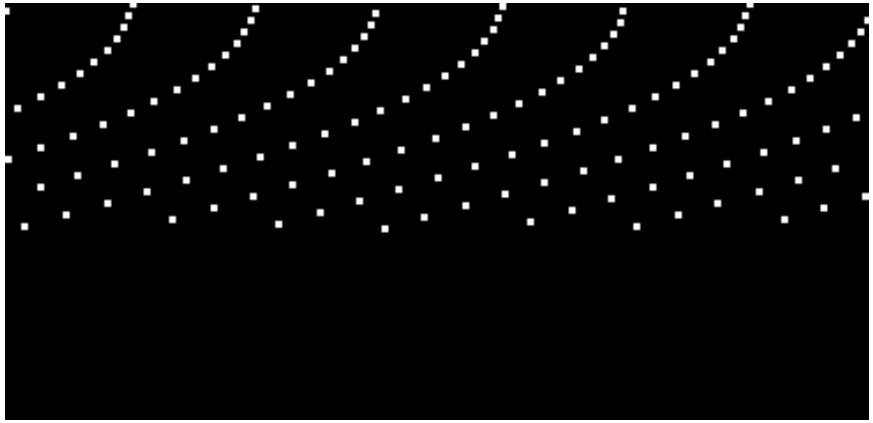


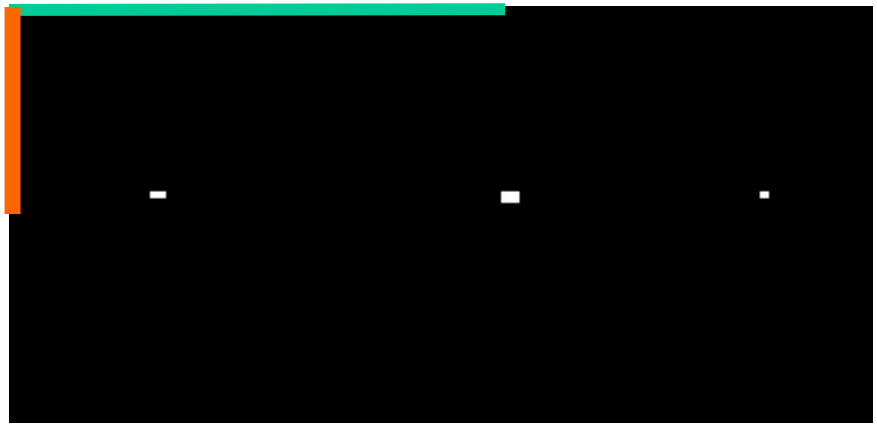
# Hough transform

voters



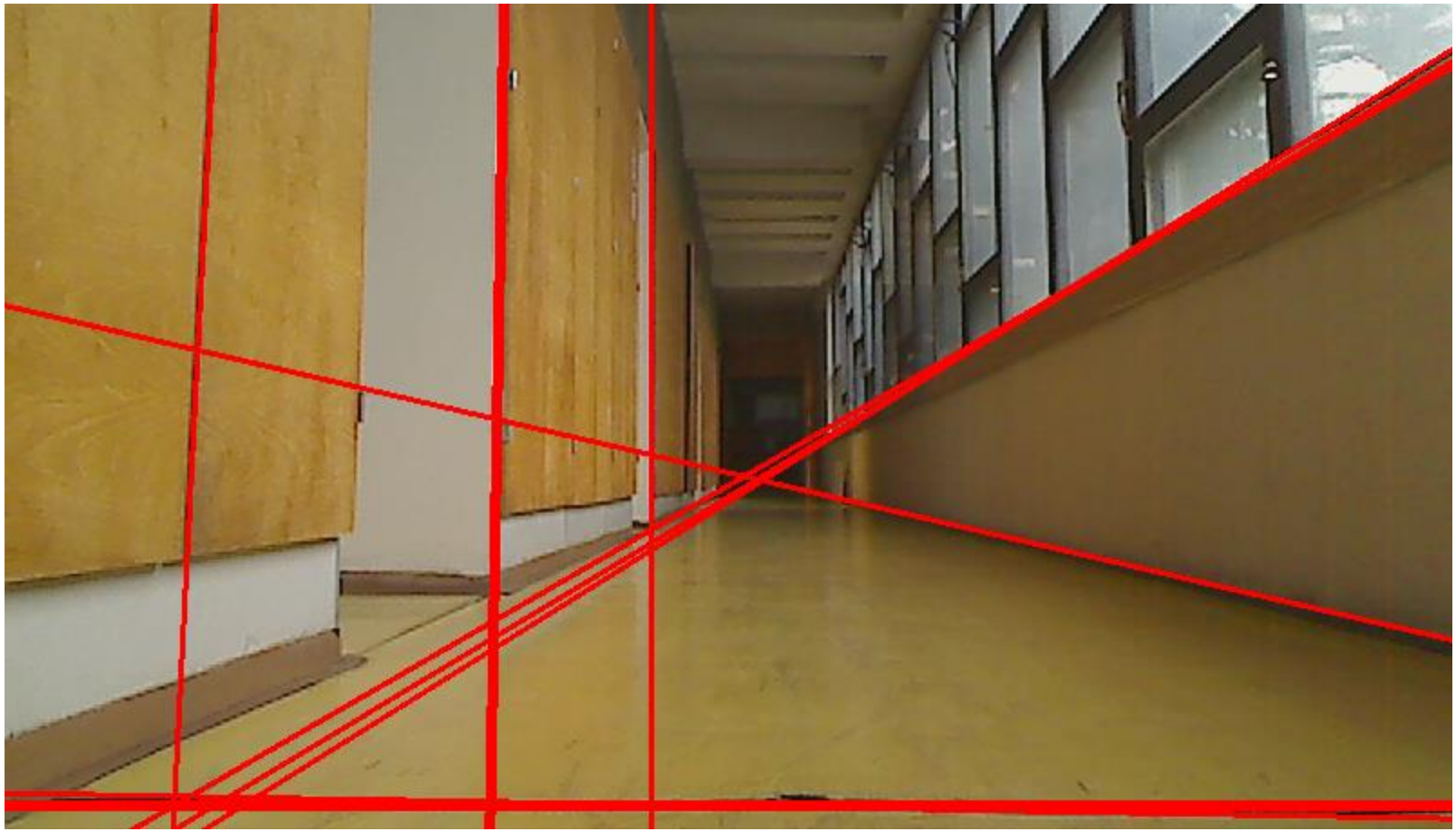
votes of one voter





votes summary

# Hough transform - lines



# Hough transform - segments

# Vanishing point detection