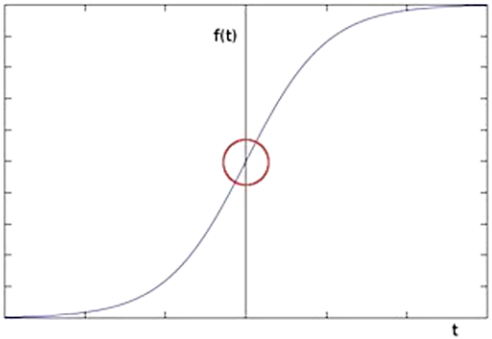
Edge Detection

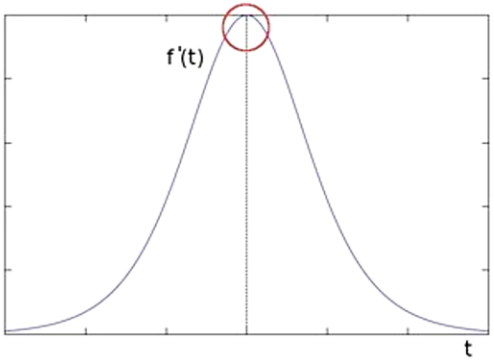
- What if we want to detect different objects regardless of colour?
- Edge detection is the process for finding structure and properties of the object (i.e. edges in an image)
- Edges are one of the most important features
- Edges can be described by sudden changes in pixel intensity
- Locations with extreme differences in brightness of pixels indicate an edge
- We need to examined changes in the neighbouring pixels
- In OpenCV, we have several options for edge detection
- We will primary experimented with: **Sobel Edge Detection** and **Canny Edge Detection**
- **You can learn more about edge detection in the follow-up courses (Digital Image Processing and Image Analysis)**

Edge Detection

To be more graphical, let's assume we have a 1D-image. An edge is shown by the "jump" in intensity in the following plot.



The edge "jump" can be seen more easily if we take the first derivative (actually, here appears as a maximum).



https://docs.opencv.org/3.4/d2/d2c/tutorial_sobel_derivatives.html

Edge Detection

Assuming that the image to be operated is I:

1. We calculate two derivatives:

   a. **Horizontal changes**: This is computed by convolving I with a kernel G_x with odd size. For example for a kernel size of 3, G_x would be computed as:

$$G_x = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix} * I$$

   a. **Vertical changes**: This is computed by convolving I with a kernel G_y with odd size. For example for a kernel size of 3, G_y would be computed as:

$$G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ +1 & +2 & +1 \end{bmatrix} * I$$

2. At each point of the image we calculate an approximation of the *gradient* in that point by combining both results above:

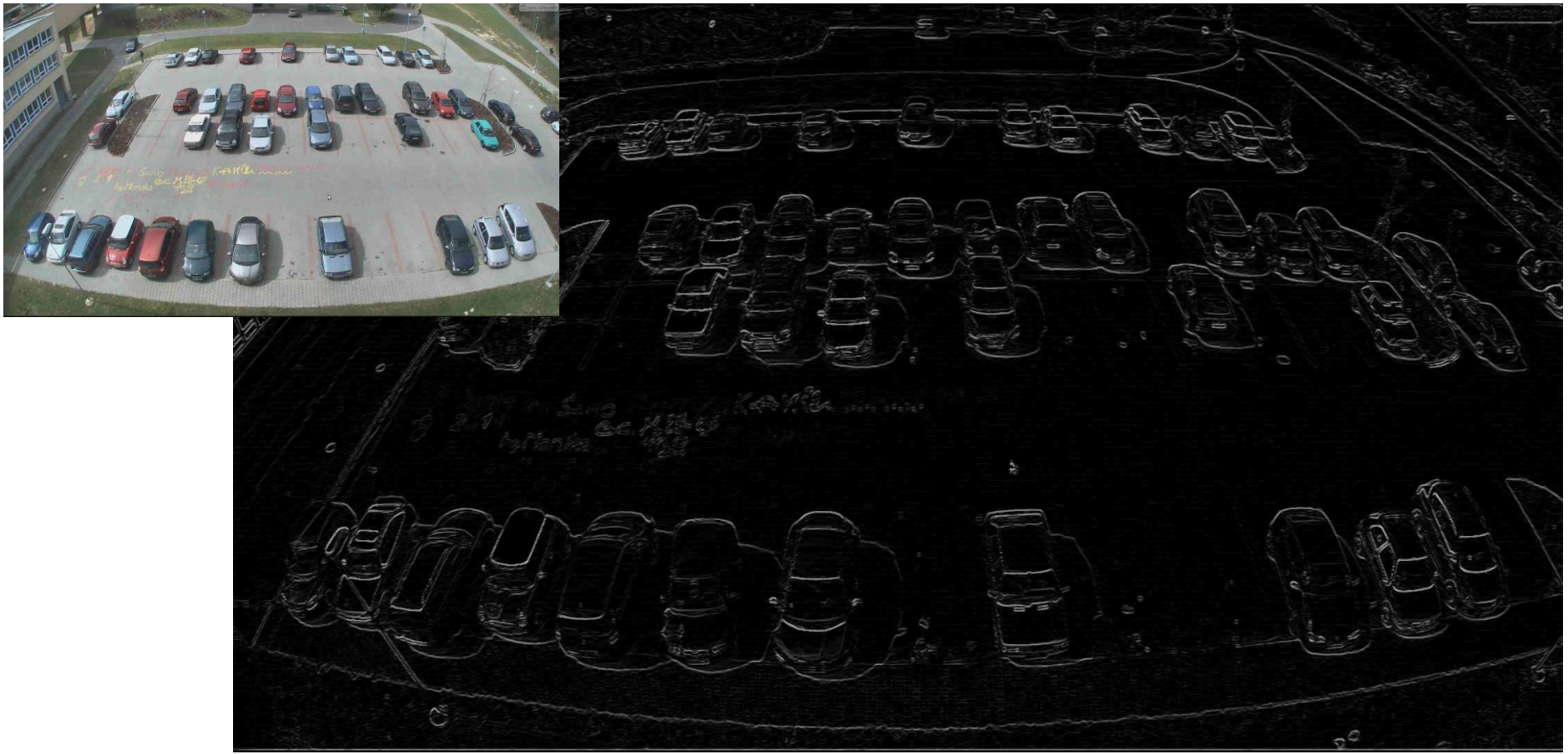
$$G = \sqrt{G_x^2 + G_y^2}$$

   Although sometimes the following simpler equation is used:

$$G = |G_x| + |G_y|$$

https://docs.opencv.org/3.4/d2/d2c/tutorial_sobel_derivatives.html

Edge Detection



Edge Detection

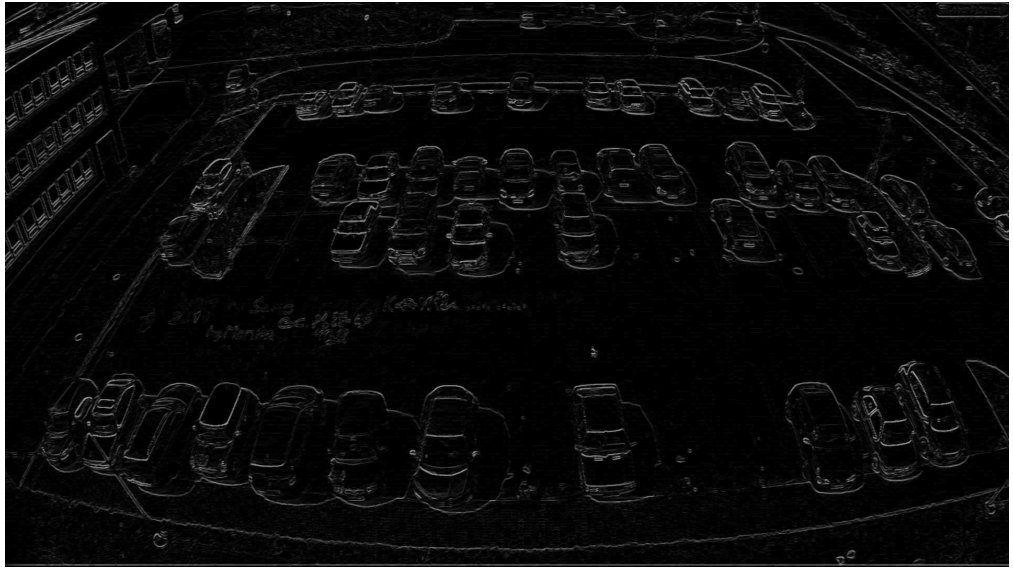
- **Sobel edge detection**
- We can use operation that is called convolution
- We need input image and kernel
- Multiply the image pixels by pixels of the filter, then sum the results
- In Sobel, we have two kernels

X – Direction Kernel

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

Y – Direction Kernel

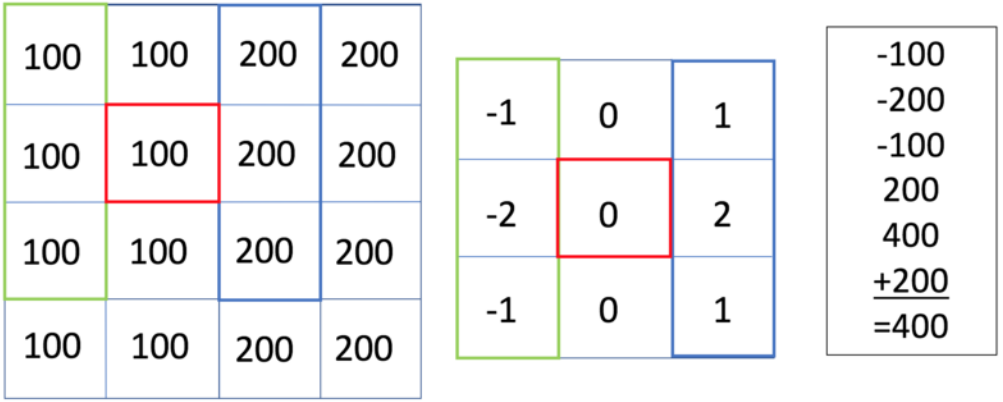
-1	-2	-1
0	0	0
1	2	1



Edge Detection

Simple (naive) explanation of convolution steps:

1. Center of the kernel is positioned over a specific pixel in an input image.
2. Each element in the kernel is multiplied with the corresponding pixel element in the input image.
3. Sum the result of multiplications
4. This result can be stored in our new image (edge map)



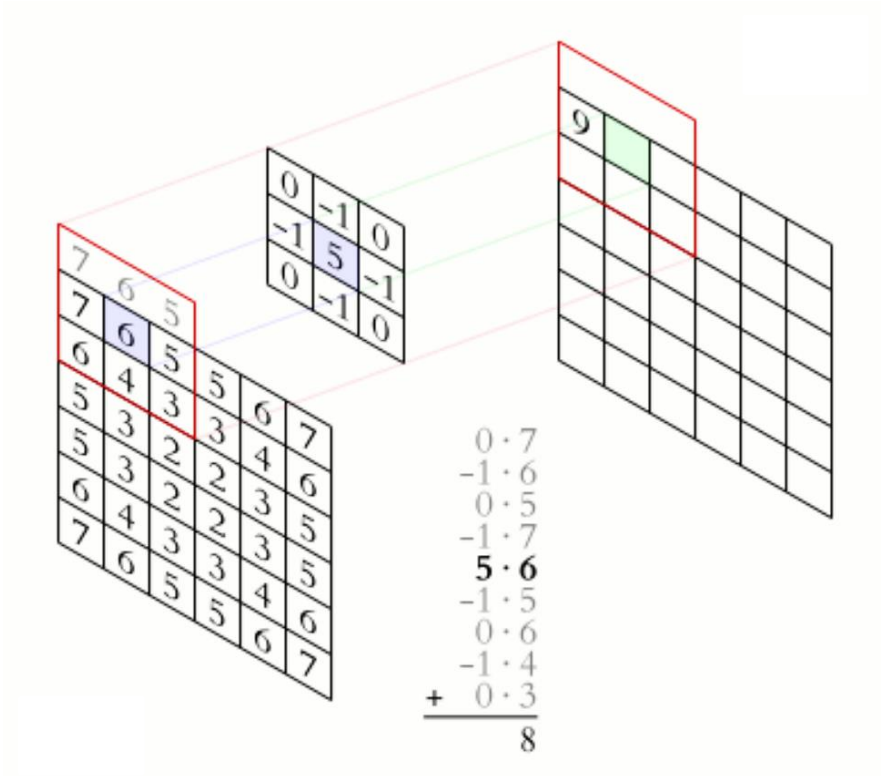
Kernel Convolution: The bigger the value at the end, the more noticeable the edge will be.

https://www.projectrhea.org/rhea/index.php/An_Implementation_of_Sobel_Edge_Detection
https://medium.com/@bdhuma/6-basic-things-to-know-about-convolution-daef5e1bc411

Edge Detection

Simple (naive) explanation of convolution steps:

1. Center of the kernel is positioned over a specific pixel in an input image.
2. Each element in the kernel is multiplied with the corresponding pixel element in the input image.
3. Sum the result of multiplications
4. This result can be stored in our new image (edge map)



https://commons.wikimedia.org/wiki/File:2D_Convolution_Animation.gif

Edge Detection

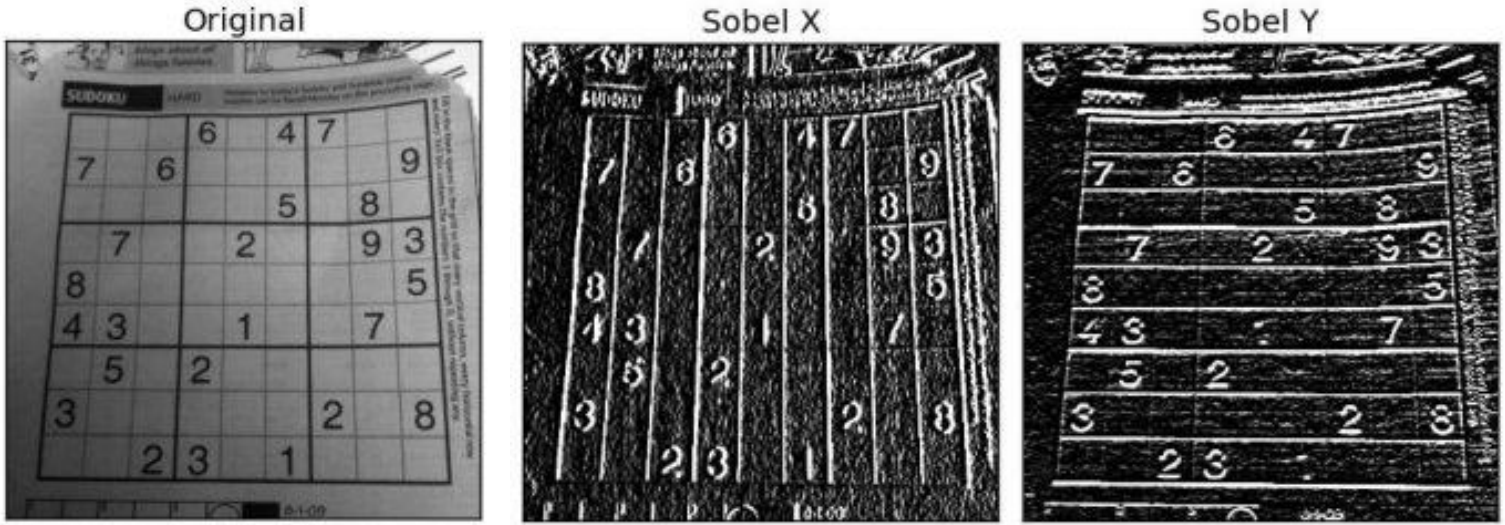
In Sobel, we have two kernels

X – Direction Kernel

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

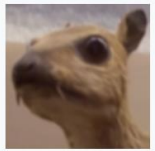
Y – Direction Kernel

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



https://docs.opencv.org/4.x/d5/d0f/tutorial_py_gradients.html

Types of Kernels

Operation	Kernel ω	Image result g(x,y)
Identity	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	
Gaussian blur 3 × 3 (approximation)	$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$	
Gaussian blur 5 × 5 (approximation)	$\frac{1}{256}\begin{bmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & 36 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix}$	
Sharpen	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
Box blur (normalized)	$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	

https://en.wikipedia.org/wiki/Kernel_(image_processing)

Types of Kernels

```
source_image = cv2.imread("test1.jpg")
image_gray = cv2.cvtColor(source_image, cv2.COLOR_BGR2GRAY)

identity_result = cv2.filter2D(image_gray, -1, identity_kernel)
sobel_y_result = cv2.filter2D(image_gray, -1, sobel_y_kernel)
sobel_x_result = cv2.filter2D(image_gray, -1, sobel_x_kernel)

#alpha * image1 + beta * image2 + y
sobel_final = cv2.addWeighted(sobel_x_result, 0.5, sobel_y_result, 0.5, 0)
```

```
identity_kernel = np.array([
                    [0, 0, 0],
                    [0, 1, 0],
                    [0, 0, 0]])

sobel_y_kernel = np.array([
                    [-1, -2, -1],
                    [0, 0, 0],
                    [1, 2, 1]])

sobel_x_kernel = np.array([
                    [-1, 0, 1],
                    [-2, 0, 2],
                    [-1, 0, 1]])
```

Types of Kernels

```
cv2.imshow("sobel_y_result", sobel_y_result)
```

Types of Kernels

```
cv2.imshow("sobel_x_result", sobel_x_result)
```

Types of Kernels

```
cv2.imshow("sobel_final", sobel_final)
```

Edge Detection

Canny edge detection (main steps , simple naive explanation):

- Noise reduction
- Sobel filters
- Edge thinning process (non-maximum suppression). In ideal case, we need thin edges. It means that the pixels with the maximum value are used.
- **Thresholding - we need to set appropriate thresholds**

- **You can learn more about Canny edge detection in the follow-up courses (Digital Image Processing)**

Edge Detection

Canny Edge Detection in OpenCV

```
43  def edge_detection_canny():
44      in_mat = cv2.imread("test2.jpg")
45      in_mat_gray = cv2.cvtColor(in_mat, cv2.COLOR_BGR2GRAY)
46
47      # img, minVal, maxVal
48      canny_edges = cv2.Canny(in_mat_gray, 100, 200)
49
50      cv2.imshow('canny_edges', canny_edges)
51      cv2.imwrite("canny_edges_100_200.png", canny_edges)
52      cv2.waitKey(0)
```

- **You can learn more about Canny edge detection in the follow-up course (Digital Image Processing)**

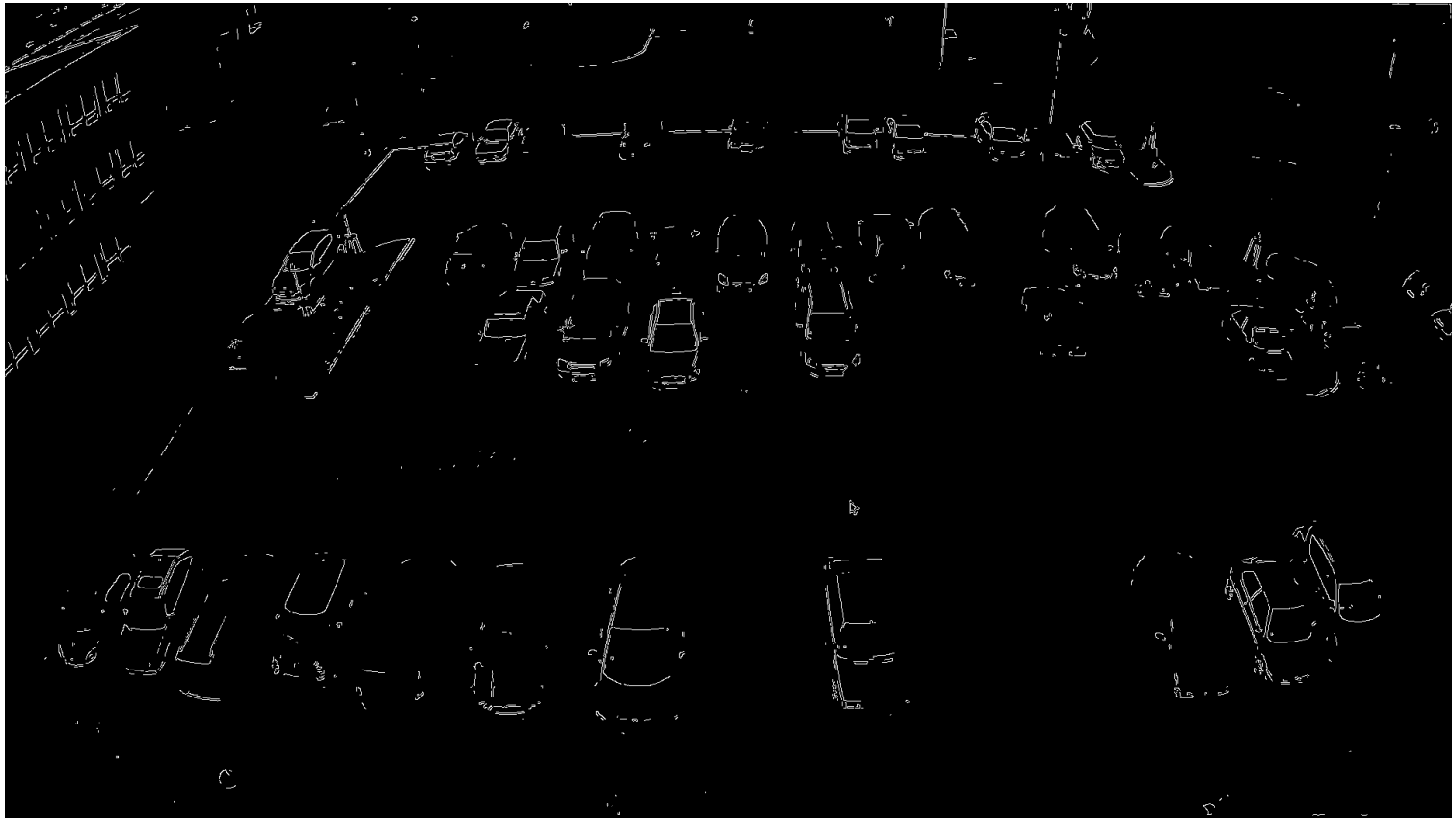
Edge Detection

10, 100

- **You can learn more about Canny edge detection in the follow-up course (Digital Image Processing)**

Edge Detection

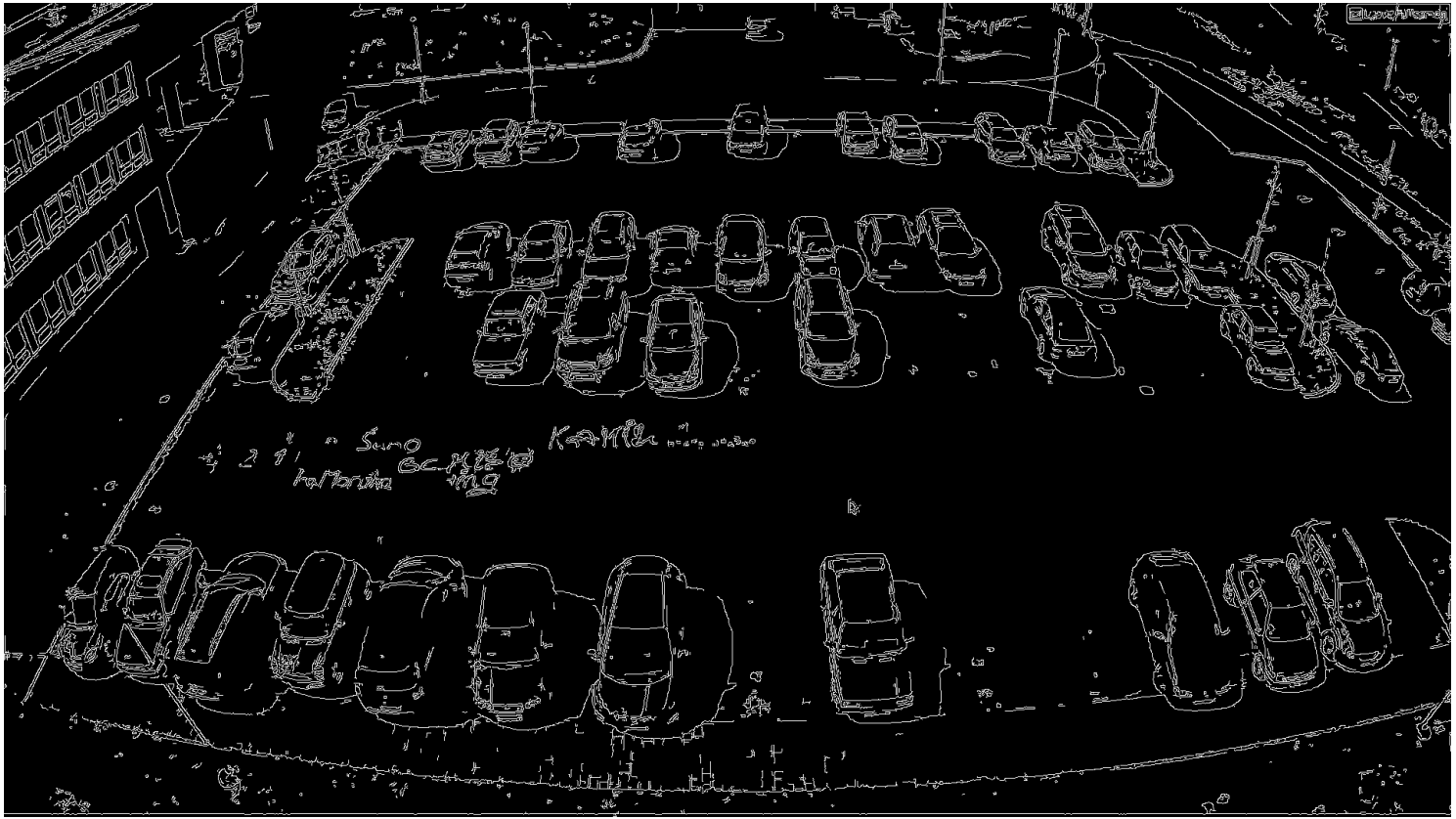
400, 500



- **You can learn more about Canny edge detection in the follow-up course (Digital Image Processing)**

Edge Detection

100, 200

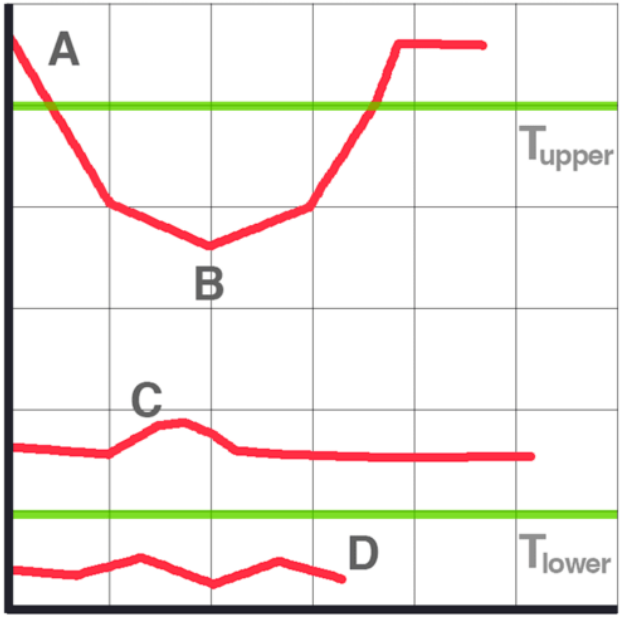


- **You can learn more about Canny edge detection in the follow-up course (Digital Image Processing)**

Edge Detection

- **Thresholding - we need to set appropriate thresholds**

- At the top of the graph, we can see that **A** is a sure edge, since $A > T_{\text{upper}}$.
- **B** is also an edge, even though $B < T_{\text{upper}}$ since it is connected to a sure edge, **A**.
- **C** is not an edge since $C < T_{\text{upper}}$ and is not connected to a strong edge.
- Finally, **D** is not an edge since $D < T_{\text{lower}}$ and is automatically discarded.



Setting these threshold ranges is not always a trivial process.

If the threshold range is *too wide,* then we'll get many false edges instead of being about to find *just* the structure and outline of an object in an image.

Similarly, if the threshold range is *too tight*, we won't find many edges at all and could be at risk of missing the structure/outline of the object entirely!

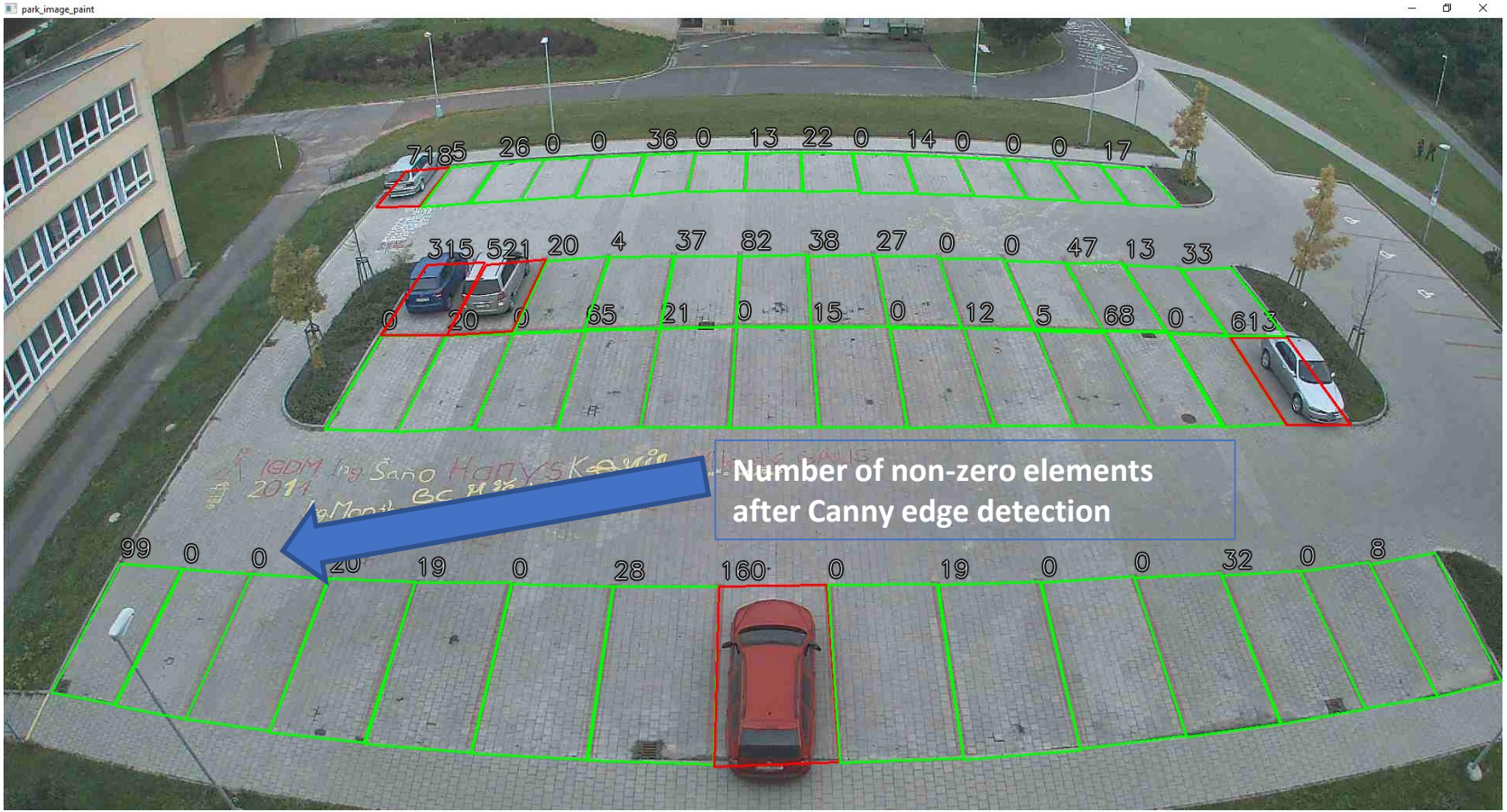
https://www.pyimagesearch.com/2021/05/12/opencv-edge-detection-cv2-canny/

```python
def main(argv):

    pkm_file = open('parking_map_python.txt', 'r')
    pkm_lines = pkm_file.readlines()
    pkm_coordinates = []

    for line in pkm_lines:
        st_line = line.strip()
        sp_line = list(st_line.split(" "))
        pkm_coordinates.append(sp_line)

    test_images = [img for img in glob.glob("test_images_zao/*.jpg")]
    test_images.sort()
    temp = cv.imread("template.png")
    size = (80, 80)
    temp = cv.resize(temp, size)
    cv.namedWindow("image_clone", 0)
    cv.namedWindow("one_place_img", 0)
    cv.namedWindow("temp", 0)
    cv.namedWindow("source", 0)
    n_park = 0
    font = cv.FONT_HERSHEY_PLAIN
    for img_name in test_images:
        image = cv.imread(img_name)
        image_clone = image.copy()
        cv.imshow("image", image)

        for coord in pkm_coordinates:
            n_park+=1
            print("coord", coord)
            pt_1 = (int(coord[0]), int(coord[1]))
            pt_3 = (int(coord[4]), int(coord[5]))
            center = ((pt_1[0]+pt_3[0])/2, (pt_1[1]+pt_3[1])/2)
            one_place_img = four_point_transform(image, coord)
            one_place_img = cv.resize(one_place_img, size)

        cv.imshow("image_clone", image_clone)
        n_park = 0
        cv.waitKey(0)
```

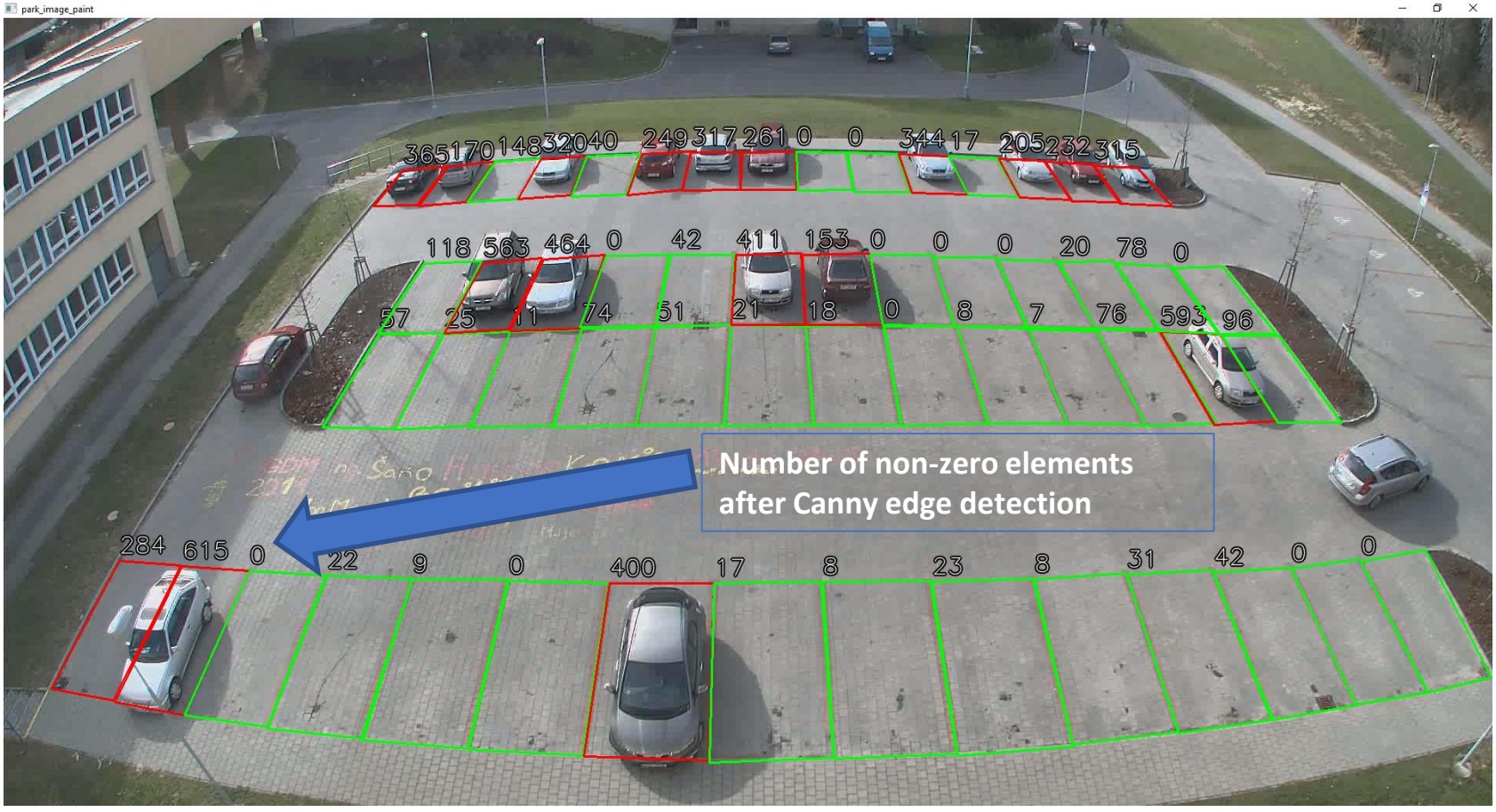
Edge Detection

- **TASK – parking lot occupation detection**
- Continue with the template
- Try new testing images:
  - http://mrl.cs.vsb.cz/data/vyuka/zao/parking/test_images_zao_24.zip
- Experiments with image filtering
- Experiments with the edge detectors:
  - Different thresholds
  - Sobel vs. Canny
  - Sobel vs. Canny vs. Template Match.
  - Hint: after edge detection, calculate non-zero elements of each place or create appropriate thresholds

Edge Detection



Edge Detection



Edge Detection