Exercise 4

Fundamentals of machine vision algorithms

dr.sc. Filip Šuligoj
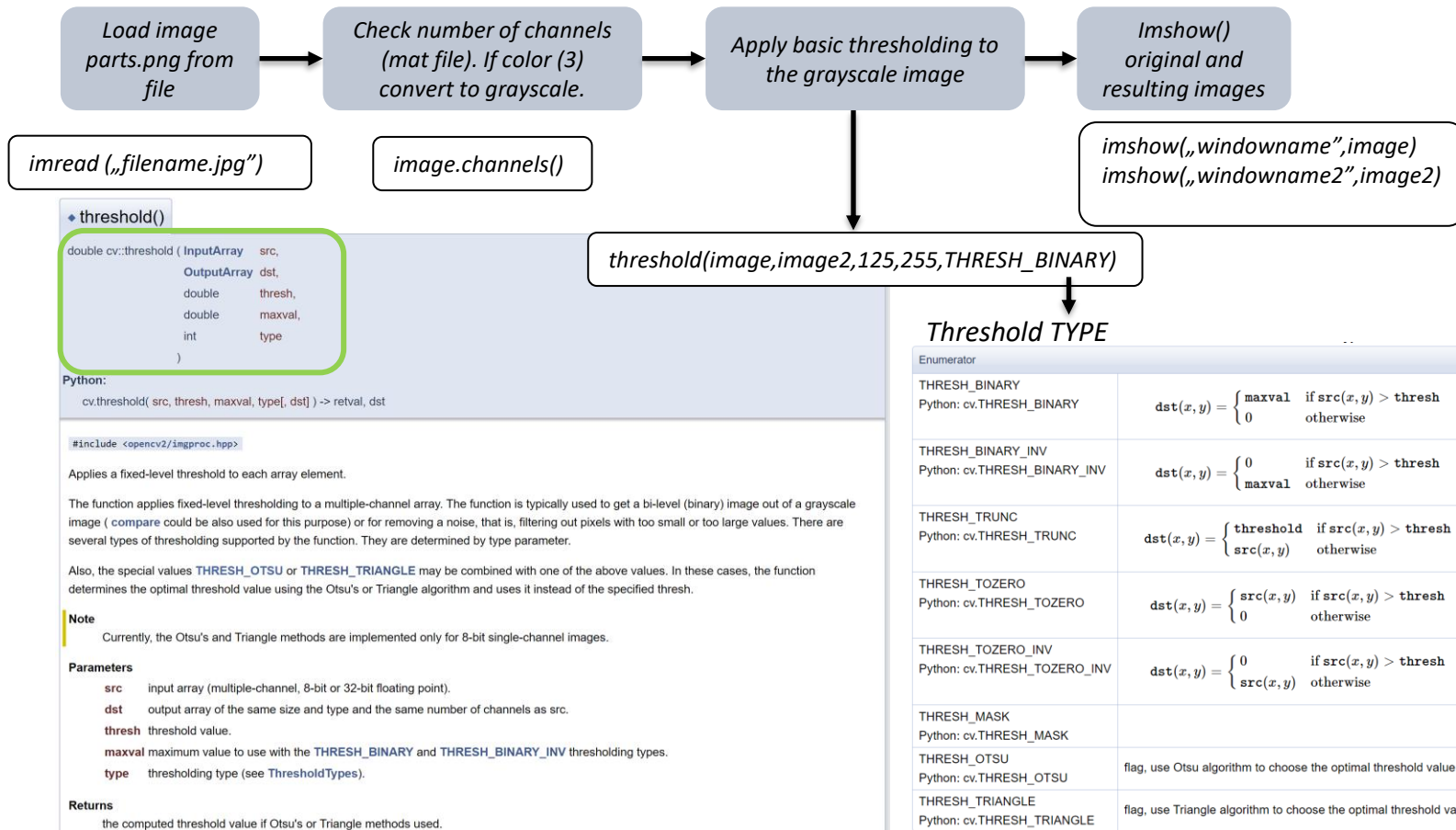*fsuligoj@fsb.hr*

University of Zagreb

FSB — Faculty of mechanical engineering and naval arhitecture

Practical task 1

*Load image and perform **BASIC** thresholding:*

- *Test different threshold values na threshold types (binary, trunc, tozero)*

Version 1

| Load image parts.png from file | → | Check number of channels (mat file). If color (3) convert to grayscale. | → | Apply basic thresholding to the grayscale image | → | Imshow() original and resulting images |

imread („filename.jpg")

image.channels()

imshow(„windowname",image)
imshow(„windowname2",image2)

♦ threshold()

```
double cv::threshold ( InputArray    src,
                       OutputArray   dst,
                       double        thresh,
                       double        maxval,
                       int           type
                     )
```

Python:
```
cv.threshold( src, thresh, maxval, type[, dst] ) -> retval, dst
```

`#include <opencv2/imgproc.hpp>`

Applies a fixed-level threshold to each array element.

The function applies fixed-level thresholding to a multiple-channel array. The function is typically used to get a bi-level (binary) image out of a grayscale image (compare could be also used for this purpose) or for removing a noise, that is, filtering out pixels with too small or too large values. There are several types of thresholding supported by the function. They are determined by type parameter.

Also, the special values THRESH_OTSU or THRESH_TRIANGLE may be combined with one of the above values. In these cases, the function determines the optimal threshold value using the Otsu's or Triangle algorithm and uses it instead of the specified thresh.

Note
Currently, the Otsu's and Triangle methods are implemented only for 8-bit single-channel images.

Parameters
- **src**   input array (multiple-channel, 8-bit or 32-bit floating point).
- **dst**   output array of the same size and type and the same number of channels as src.
- **thresh** threshold value.
- **maxval** maximum value to use with the THRESH_BINARY and THRESH_BINARY_INV thresholding types.
- **type**   thresholding type (see ThresholdTypes).

Returns
the computed threshold value if Otsu's or Triangle methods used.

threshold(image,image2,125,255,THRESH_BINARY)

Threshold TYPE

Enumerator	
THRESH_BINARY Python: cv.THRESH_BINARY	$\mathrm{dst}(x,y) = \begin{cases} \mathtt{maxval} & \text{if } \mathrm{src}(x,y) > \mathtt{thresh} \\ 0 & \text{otherwise} \end{cases}$
THRESH_BINARY_INV Python: cv.THRESH_BINARY_INV	$\mathrm{dst}(x,y) = \begin{cases} 0 & \text{if } \mathrm{src}(x,y) > \mathtt{thresh} \\ \mathtt{maxval} & \text{otherwise} \end{cases}$
THRESH_TRUNC Python: cv.THRESH_TRUNC	$\mathrm{dst}(x,y) = \begin{cases} \mathtt{threshold} & \text{if } \mathrm{src}(x,y) > \mathtt{thresh} \\ \mathrm{src}(x,y) & \text{otherwise} \end{cases}$
THRESH_TOZERO Python: cv.THRESH_TOZERO	$\mathrm{dst}(x,y) = \begin{cases} \mathrm{src}(x,y) & \text{if } \mathrm{src}(x,y) > \mathtt{thresh} \\ 0 & \text{otherwise} \end{cases}$
THRESH_TOZERO_INV Python: cv.THRESH_TOZERO_INV	$\mathrm{dst}(x,y) = \begin{cases} 0 & \text{if } \mathrm{src}(x,y) > \mathtt{thresh} \\ \mathrm{src}(x,y) & \text{otherwise} \end{cases}$
THRESH_MASK Python: cv.THRESH_MASK	
THRESH_OTSU Python: cv.THRESH_OTSU	flag, use Otsu algorithm to choose the optimal threshold value
THRESH_TRIANGLE Python: cv.THRESH_TRIANGLE	flag, use Triangle algorithm to choose the optimal threshold value

https://docs.opencv.org/master/d7/d1b/group__imgproc__misc.html#gae8a4a146d1ca78c626a53577199e9c57
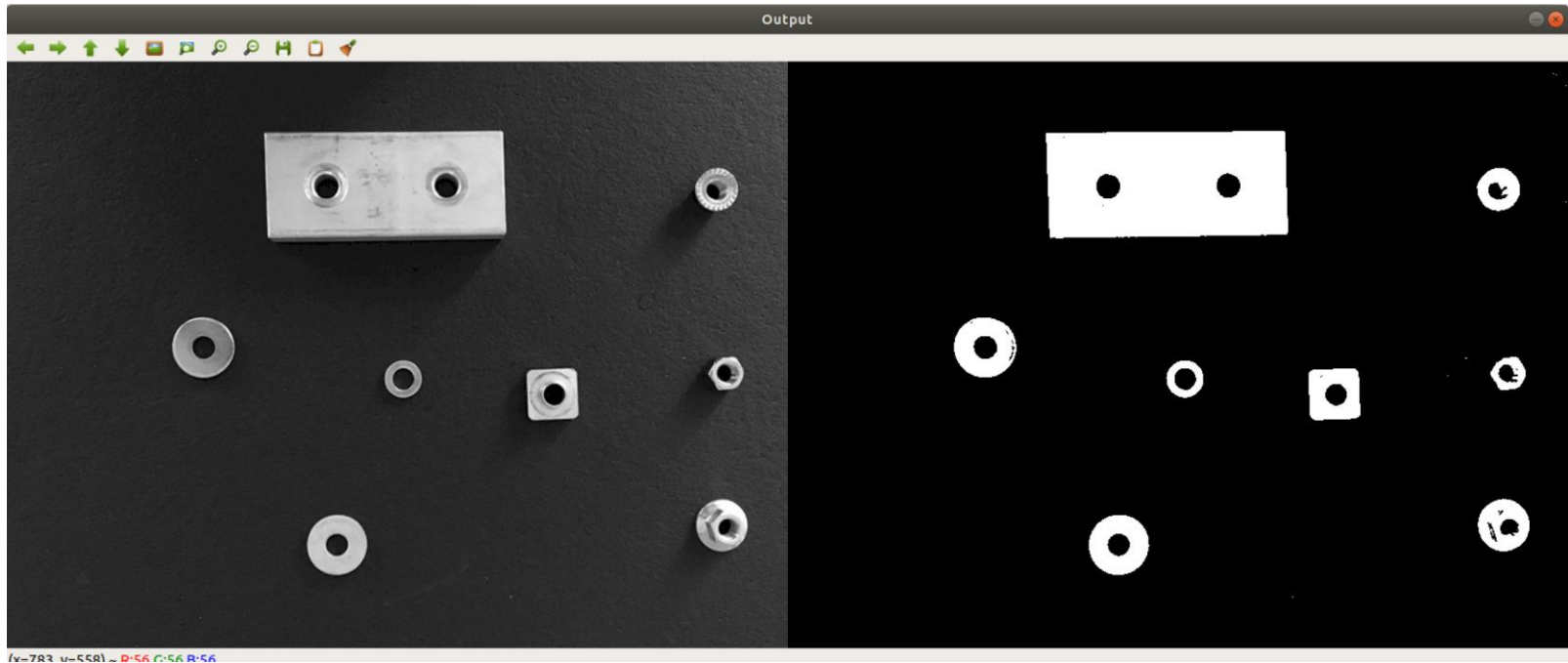
University of Zagreb

FSB — Faculty of mechanical engineering and naval architecture

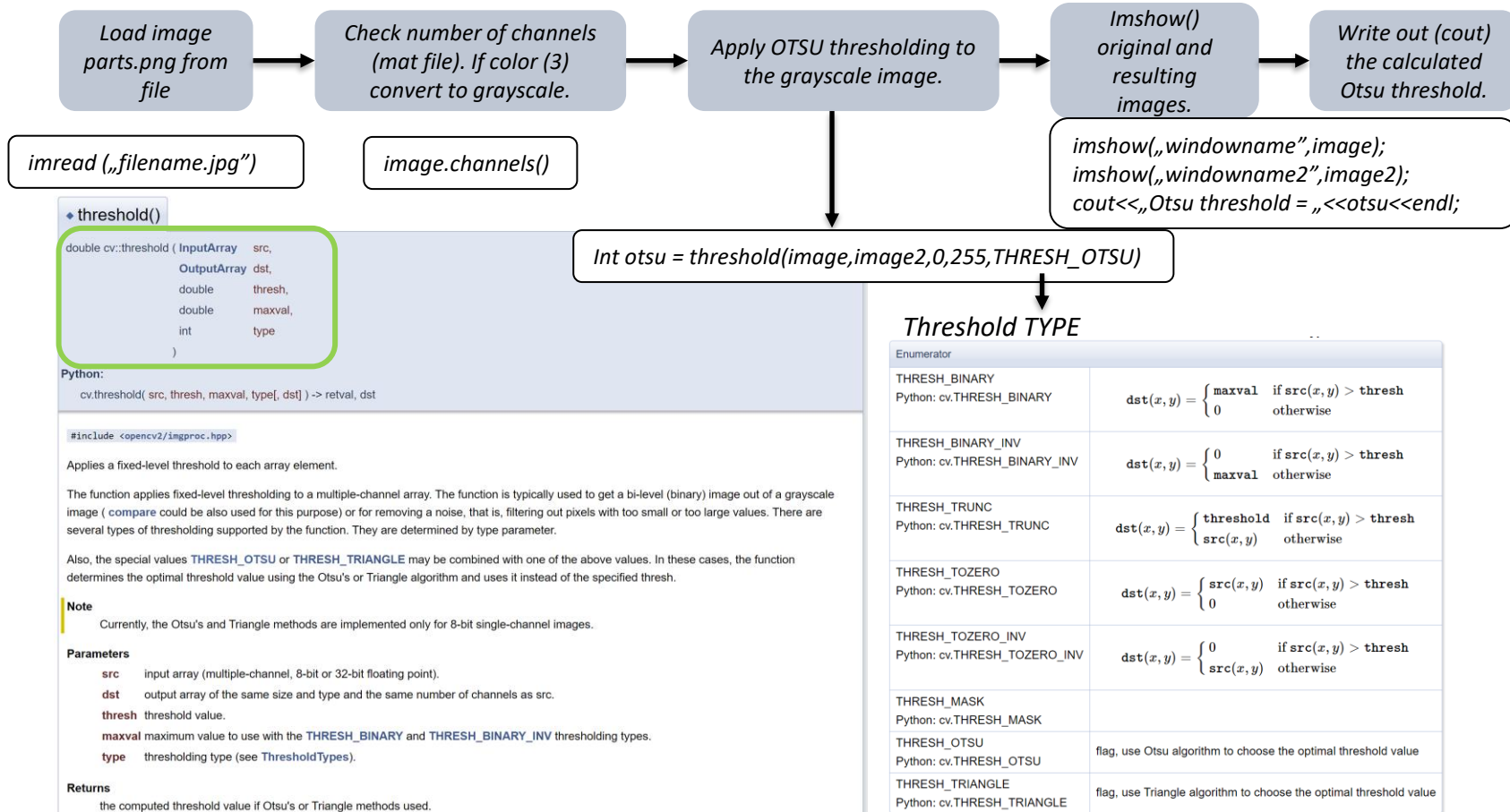# Practical task 1

*Load image and perform **BASIC** thresholding.*

RESULT:

Practical task 2

Load image and perform **OTSU** thresholding:
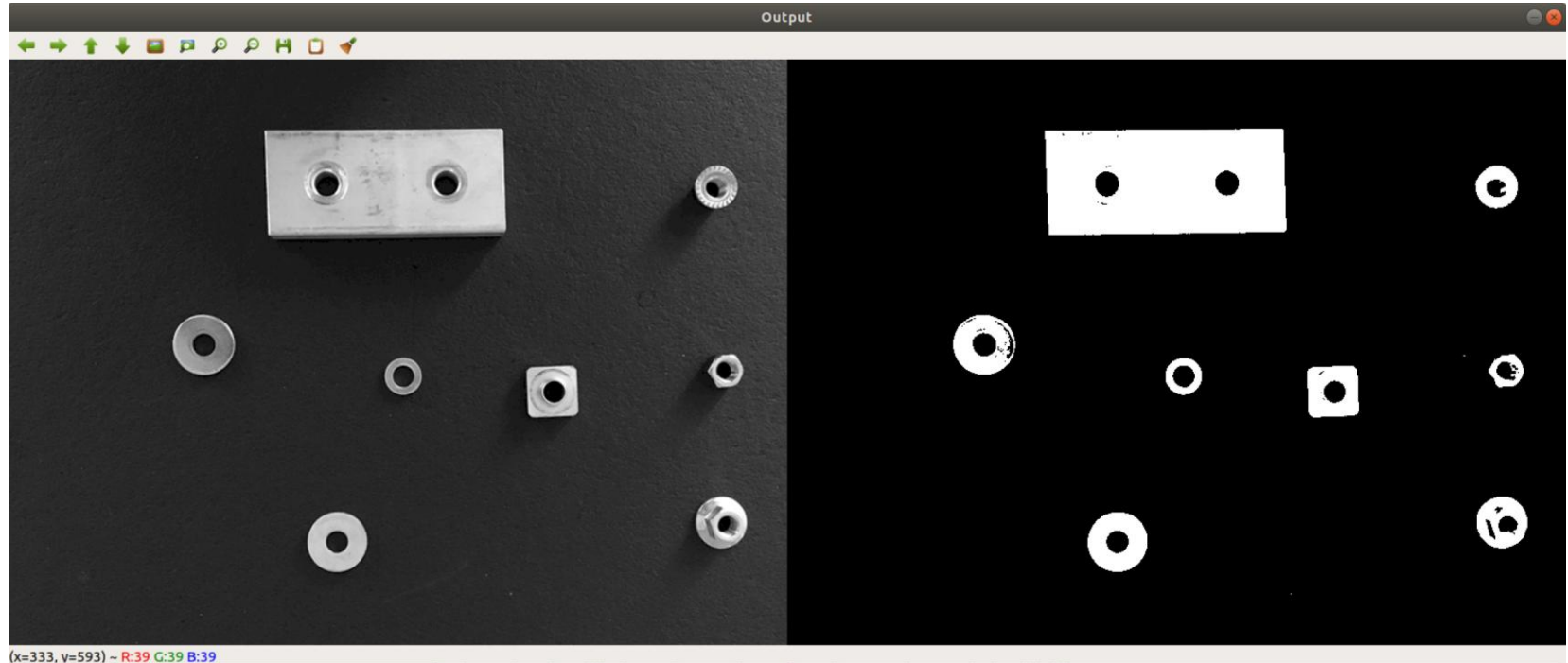
– *Write out the calcualted Otsu's threshold*

Version 1

| Load image parts.png from file | → | Check number of channels (mat file). If color (3) convert to grayscale. | → | Apply OTSU thresholding to the grayscale image. | → | Imshow() original and resulting images. | → | Write out (cout) the calculated Otsu threshold. |

imread („filename.jpg")

image.channels()

imshow(„windowname",image);
imshow(„windowname2",image2);
cout<<„Otsu threshold = „<<otsu<<endl;

Int otsu = threshold(image,image2,0,255,THRESH_OTSU)

◆ threshold()

```
double cv::threshold ( InputArray    src,
                       OutputArray   dst,
                       double        thresh,
                       double        maxval,
                       int           type
                      )
```

Python:
   cv.threshold(src, thresh, maxval, type[, dst]) -> retval, dst

```
#include <opencv2/imgproc.hpp>
```

Applies a fixed-level threshold to each array element.

The function applies fixed-level thresholding to a multiple-channel array. The function is typically used to get a bi-level (binary) image out of a grayscale image (compare could be also used for this purpose) or for removing a noise, that is, filtering out pixels with too small or too large values. There are several types of thresholding supported by the function. They are determined by type parameter.

Also, the special values THRESH_OTSU or THRESH_TRIANGLE may be combined with one of the above values. In these cases, the function determines the optimal threshold value using the Otsu's or Triangle algorithm and uses it instead of the specified thresh.

Note
   Currently, the Otsu's and Triangle methods are implemented only for 8-bit single-channel images.

Parameters
   src   input array (multiple-channel, 8-bit or 32-bit floating point).
   dst   output array of the same size and type and the same number of channels as src.
   thresh threshold value.
   maxval maximum value to use with the THRESH_BINARY and THRESH_BINARY_INV thresholding types.
   type   thresholding type (see ThresholdTypes).

Returns
   the computed threshold value if Otsu's or Triangle methods used.

Threshold TYPE

Enumerator	
THRESH_BINARY Python: cv.THRESH_BINARY	$dst(x,y) = \begin{cases} \text{maxval} & \text{if } src(x,y) > \text{thresh} \\ 0 & \text{otherwise} \end{cases}$
THRESH_BINARY_INV Python: cv.THRESH_BINARY_INV	$dst(x,y) = \begin{cases} 0 & \text{if } src(x,y) > \text{thresh} \\ \text{maxval} & \text{otherwise} \end{cases}$
THRESH_TRUNC Python: cv.THRESH_TRUNC	$dst(x,y) = \begin{cases} \text{threshold} & \text{if } src(x,y) > \text{thresh} \\ src(x,y) & \text{otherwise} \end{cases}$
THRESH_TOZERO Python: cv.THRESH_TOZERO	$dst(x,y) = \begin{cases} src(x,y) & \text{if } src(x,y) > \text{thresh} \\ 0 & \text{otherwise} \end{cases}$
THRESH_TOZERO_INV Python: cv.THRESH_TOZERO_INV	$dst(x,y) = \begin{cases} 0 & \text{if } src(x,y) > \text{thresh} \\ src(x,y) & \text{otherwise} \end{cases}$
THRESH_MASK Python: cv.THRESH_MASK	
THRESH_OTSU Python: cv.THRESH_OTSU	flag, use Otsu algorithm to choose the optimal threshold value
THRESH_TRIANGLE Python: cv.THRESH_TRIANGLE	flag, use Triangle algorithm to choose the optimal threshold value

https://docs.opencv.org/master/d7/d1b/group__imgproc__misc.html#gae8a4a146d1ca78c626a53577199e9c57

University of Zagreb

FSB — Faculty of mechanical engineering and naval arhitecture

Practical task 2

*Load image and perform **OTSU** thresholding.*
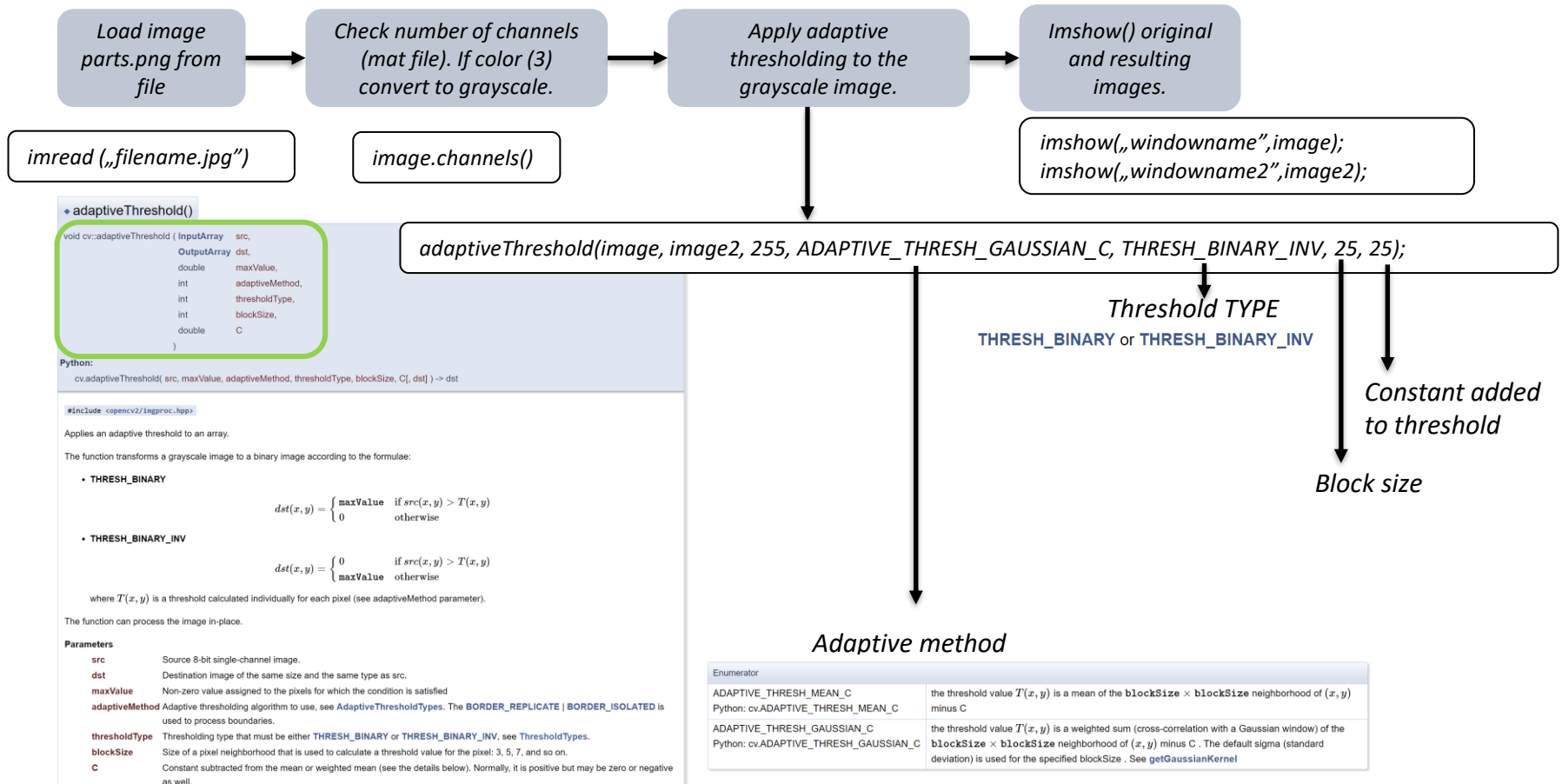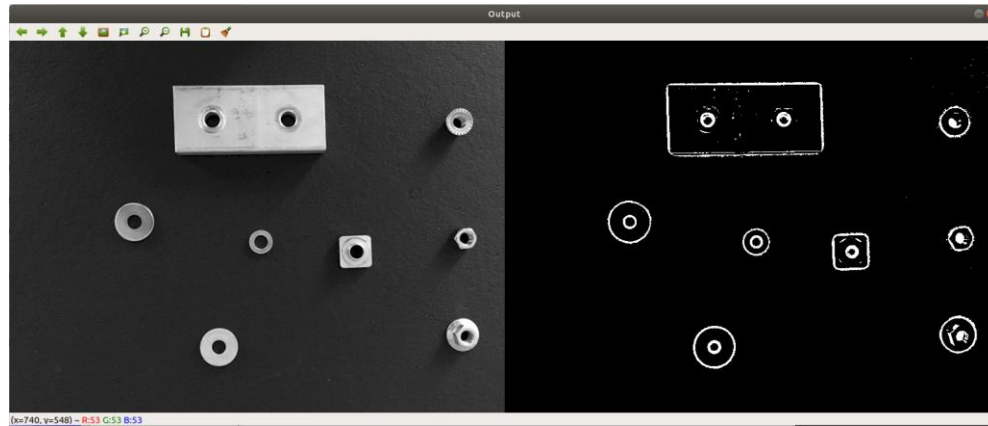
RESULT: calculated Otsu threshold should be 116.

Practical task 3

*Load image and perform **ADAPTIVE** thresholding:*

– *Test different threshold values and neighbourhood sizes*

Version 1

Load image parts.png from file	→	Check number of channels (mat file). If color (3) convert to grayscale.	→	Apply adaptive thresholding to the grayscale image.	→	Imshow() original and resulting images.

imread („filename.jpg")

image.channels()

imshow(„windowname",image);
imshow(„windowname2",image2);

◆ adaptiveThreshold()

```
void cv::adaptiveThreshold ( InputArray    src,
                             OutputArray   dst,
                             double        maxValue,
                             int           adaptiveMethod,
                             int           thresholdType,
                             int           blockSize,
                             double        C
                           )
```

Python:
    cv.adaptiveThreshold(src, maxValue, adaptiveMethod, thresholdType, blockSize, C[, dst]) -> dst

```
#include <opencv2/imgproc.hpp>
```

Applies an adaptive threshold to an array.

The function transforms a grayscale image to a binary image according to the formulae:

- **THRESH_BINARY**

$$dst(x,y) = \begin{cases} \texttt{maxValue} & \text{if } src(x,y) > T(x,y) \\ 0 & \text{otherwise} \end{cases}$$

- **THRESH_BINARY_INV**

$$dst(x,y) = \begin{cases} 0 & \text{if } src(x,y) > T(x,y) \\ \texttt{maxValue} & \text{otherwise} \end{cases}$$

where $T(x,y)$ is a threshold calculated individually for each pixel (see adaptiveMethod parameter).

The function can process the image in-place.

Parameters

src	Source 8-bit single-channel image.
dst	Destination image of the same size and the same type as src.
maxValue	Non-zero value assigned to the pixels for which the condition is satisfied
adaptiveMethod	Adaptive thresholding algorithm to use, see AdaptiveThresholdTypes. The BORDER_REPLICATE	BORDER_ISOLATED is used to process boundaries.
thresholdType	Thresholding type that must be either THRESH_BINARY or THRESH_BINARY_INV, see ThresholdTypes.
blockSize	Size of a pixel neighborhood that is used to calculate a threshold value for the pixel: 3, 5, 7, and so on.
C	Constant subtracted from the mean or weighted mean (see the details below). Normally, it is positive but may be zero or negative as well.

adaptiveThreshold(image, image2, 255, ADAPTIVE_THRESH_GAUSSIAN_C, THRESH_BINARY_INV, 25, 25);

Threshold TYPE
THRESH_BINARY or **THRESH_BINARY_INV**

Constant added to threshold

Block size

Adaptive method

Enumerator	
ADAPTIVE_THRESH_MEAN_C Python: cv.ADAPTIVE_THRESH_MEAN_C	the threshold value $T(x,y)$ is a mean of the **blockSize** × **blockSize** neighborhood of (x,y) minus C
ADAPTIVE_THRESH_GAUSSIAN_C Python: cv.ADAPTIVE_THRESH_GAUSSIAN_C	the threshold value $T(x,y)$ is a weighted sum (cross-correlation with a Gaussian window) of the **blockSize** × **blockSize** neighborhood of (x,y) minus C . The default sigma (standard deviation) is used for the specified blockSize . See getGaussianKernel

https://docs.opencv.org/master/d7/d1b/group__imgproc__misc.html#ga72b913f352e4a1b1b397736707afcde3

University of Zagreb

FSB

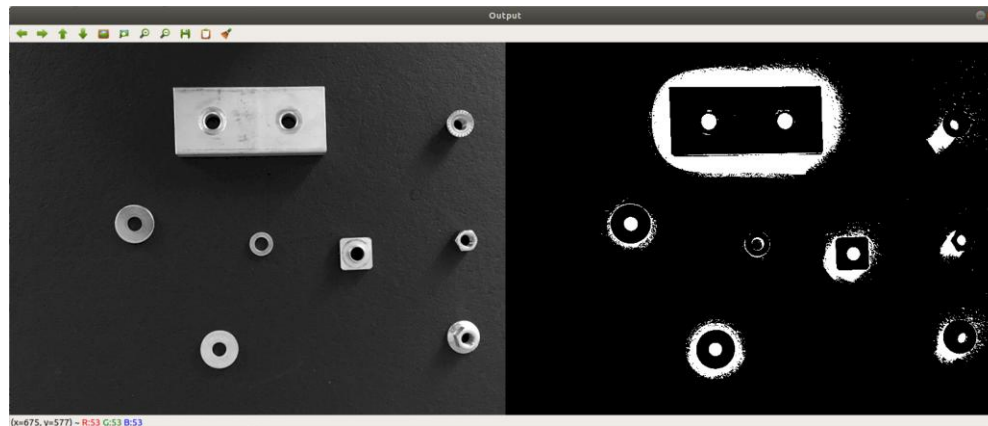Faculty of mechanical engineering and naval architecture

Practical task 3

*Load image and perform **ADAPTIVE** thresholding.*

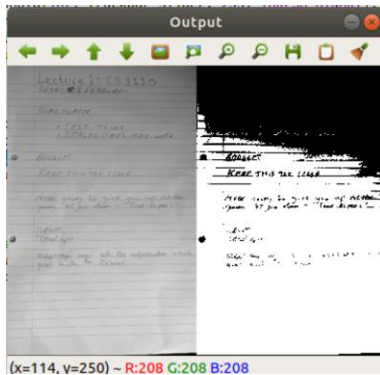*RESULT:*   *THRESHOLD 25, Block size 25*
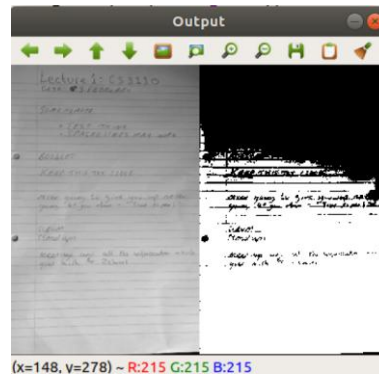

THRESHOLD 25, Block size 201

Practical task 4

*Compare results of **BASIC, OTSU and ADAPTIVE** thresholding on the following images that have varying lighting conditions (gray_text.png and threshold_example.png):*

BASIC
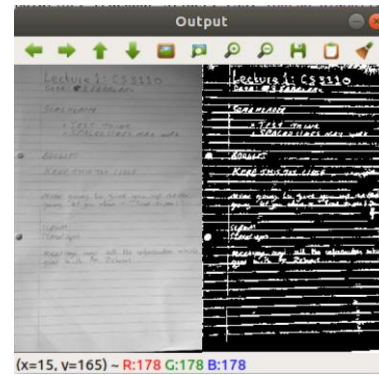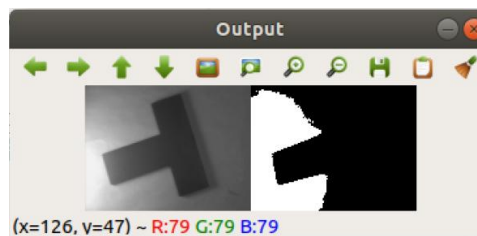

(x=114, v=250) ~ R:208 G:208 B:208
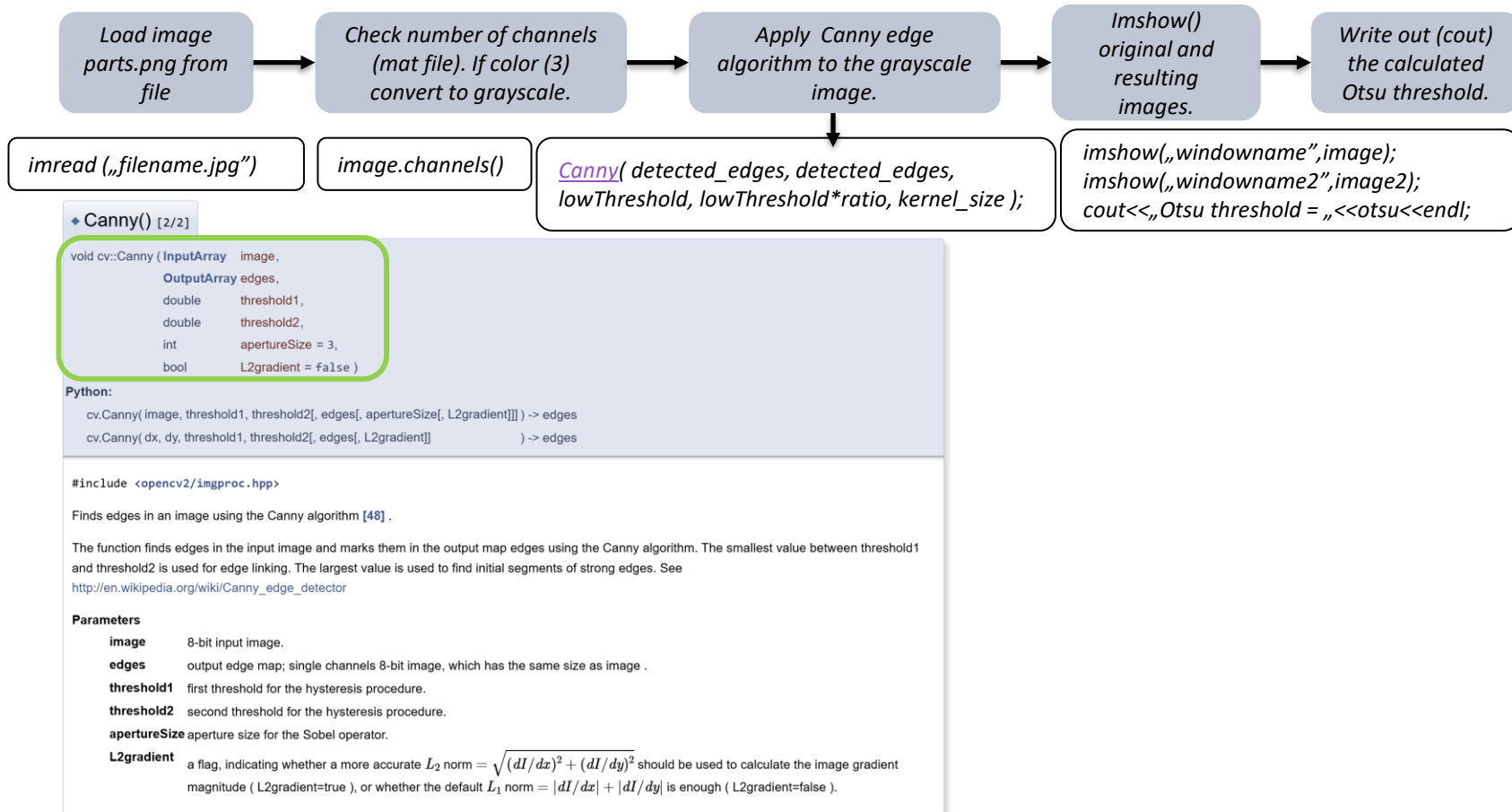
OTSU

(x=148, v=278) ~ R:215 G:215 B:215

ADAPTIVE

(x=15, v=165) ~ R:178 G:178 B:178

BASIC

(x=35, v=91) ~ R:155 G:155 B:155

OTSU

(x=126, v=47) ~ R:79 G:79 B:79

ADAPTIVE

(x=26, v=78) ~ R:181 G:181 B:181

Practical task 5

Load image and perform edge detection using canny edge algorithm:

| Load image parts.png from file | → | Check number of channels (mat file). If color (3) convert to grayscale. | → | Apply Canny edge algorithm to the grayscale image. | → | Imshow() original and resulting images. | → | Write out (cout) the calculated Otsu threshold. |

imread („filename.jpg")

image.channels()

Canny(detected_edges, detected_edges, lowThreshold, lowThreshold*ratio, kernel_size);

imshow(„windowname",image);
imshow(„windowname2",image2);
cout<<„Otsu threshold = „<<otsu<<endl;

◆ Canny() [2/2]

```
void cv::Canny ( InputArray   image,
                 OutputArray edges,
                 double       threshold1,
                 double       threshold2,
                 int          apertureSize = 3,
                 bool         L2gradient = false )
```

Python:
```
cv.Canny( image, threshold1, threshold2[, edges[, apertureSize[, L2gradient]]] ) -> edges
cv.Canny( dx, dy, threshold1, threshold2[, edges[, L2gradient]]               ) -> edges
```

`#include <opencv2/imgproc.hpp>`

Finds edges in an image using the Canny algorithm [48] .

The function finds edges in the input image and marks them in the output map edges using the Canny algorithm. The smallest value between threshold1 and threshold2 is used for edge linking. The largest value is used to find initial segments of strong edges. See
http://en.wikipedia.org/wiki/Canny_edge_detector

Parameters

image	8-bit input image.
edges	output edge map; single channels 8-bit image, which has the same size as image .
threshold1	first threshold for the hysteresis procedure.
threshold2	second threshold for the hysteresis procedure.
apertureSize	aperture size for the Sobel operator.
L2gradient	a flag, indicating whether a more accurate L_2 norm $= \sqrt{(dI/dx)^2 + (dI/dy)^2}$ should be used to calculate the image gradient magnitude (L2gradient=true), or whether the default L_1 norm $= |dI/dx| + |dI/dy|$ is enough (L2gradient=false).

https://docs.opencv.org/3.4/da/d5c/tutorial_canny_detector.html

University of Zagreb

FSB Faculty of mechanical engineering and naval arhitecture

Student assignment (seminar)

University of Zagreb

FSB

Faculty of mechanical engineering and naval arhitecture

Student assignment - seminar

Add trackbar and change the ADAPTIVE threshold variables manually in code from Practical tasks 3.

For trackbar use example :

https://docs.opencv.org/master/da/d6a/tutorial_trackbar.html