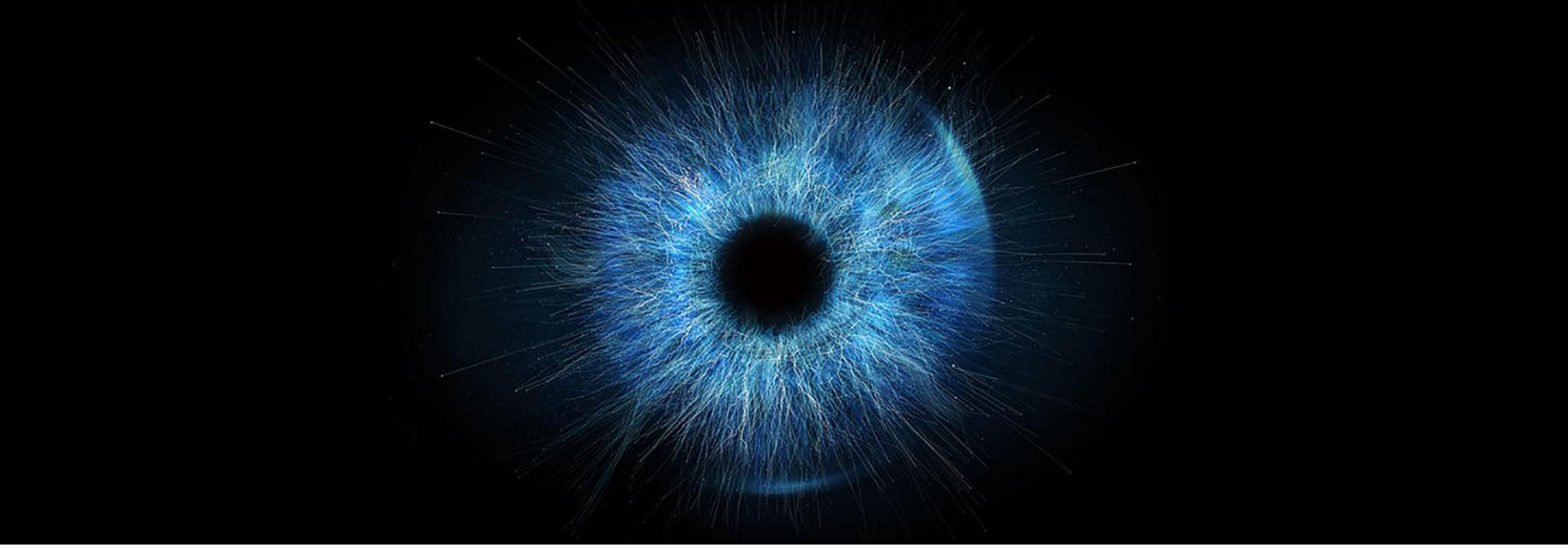


Exercise 5

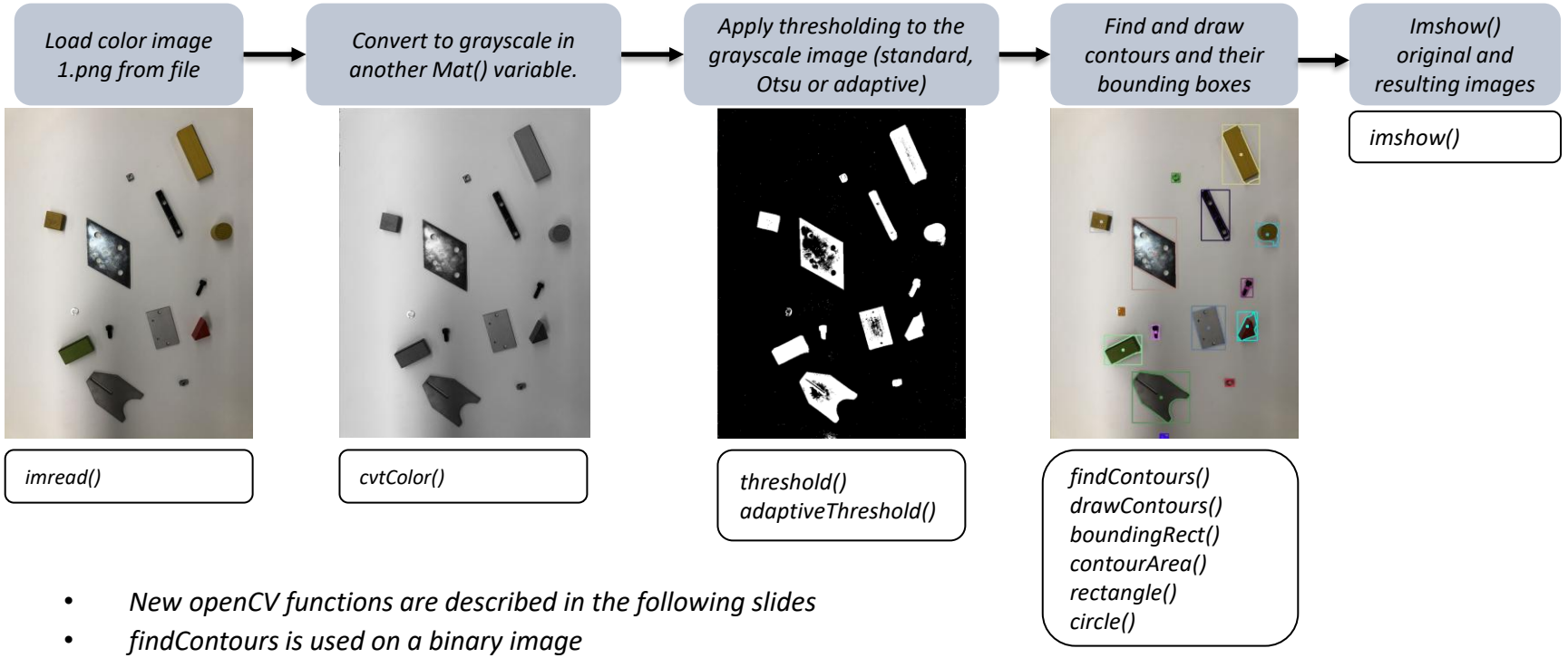
Fundamentals of machine vision algorithms

dr.sc. Filip Šuligoj
fsuligoj@fsb.hr

University of Zagreb
FSB Faculty of mechanical engineering and naval arhitecture

Practical task 1

Object localization. Generate contours around thresholded parts:



- *New openCV functions are described in the following slides*
- *findContours is used on a binary image*
- *Image processing prior to contour extraction is crucial for good results*
  - *Thresholding method and parameters should be found that give best possible results*
  - *Small particles can be removed with morphological operations (opening) – Practical task 2*
- *Shadows and light reflection can deteriorate localization results (notice reflective surfaces)*
- *Contours can be filtered based on area size – Practical task 3*
- *Objects can be filtered based on color – Practical task 4*

https://docs.opencv.org/master/d3/dc0/group__imgproc__shape.html#gadf1ad6a0b82947fa1fe3c3d497f260e0

Practical task 1

findContours()

https://docs.opencv.org/master/d3/dc0/group__imgproc__shape.html#gadf1ad6a0b82947fa1fe3c3d497f260e0

https://docs.opencv.org/master/d3/dc0/group__imgproc__shape.html#ga819779b9857cc2f8601e6526a3a5bc71

◆ findContours() [1/2]

```
void cv::findContours ( InputArray          image,
                        OutputArrayOfArrays contours,
                        OutputArray         hierarchy,
                        int                 mode,
                        int                 method,
                        Point               offset = Point()
                      )
```

vector<vector<Point> > contours;

Python:

```
cv.findContours( image, mode, method[, contours[, hierarchy[, offset]]] ) -> contours, hierarchy
```

Enumerator
RETR_EXTERNAL Python: cv.RETR_EXTERNAL
RETR_LIST Python: cv.RETR_LIST
RETR_CCOMP Python: cv.RETR_CCOMP
RETR_TREE Python: cv.RETR_TREE
RETR_FLOODFILL Python: cv.RETR_FLOODFILL

https://docs.opencv.org/master/d3/dc0/group__imgproc__shape.html#ga4303f45752694956374734a03c54d5ff

CHAIN_APPROX_NONE Python: cv.CHAIN_APPROX_NONE
CHAIN_APPROX_SIMPLE Python: cv.CHAIN_APPROX_SIMPLE
CHAIN_APPROX_TC89_L1 Python: cv.CHAIN_APPROX_TC89_L1
CHAIN_APPROX_TC89_KCOS Python: cv.CHAIN_APPROX_TC89_KCOS

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

Finds contours in a binary image.

The function retrieves contours from the binary image using the algorithm **[233]** . The contours are a useful tool for shape analysis and object detection and recognition. See squares.cpp in the OpenCV sample directory.

Note

Since opencv 3.2 source image is not modified by this function.

Parameters

image	Source, an 8-bit single-channel image. Non-zero pixels are treated as 1's. Zero pixels remain 0's, so the image is treated as binary . You can use **compare**, **inRange**, **threshold** , **adaptiveThreshold**, **Canny**, and others to create a binary image out of a grayscale or color one. If mode equals to **RETR_CCOMP** or **RETR_FLOODFILL**, the input can also be a 32-bit integer image of labels (CV_32SC1).
contours	Detected contours. Each contour is stored as a vector of points (e.g. std::vector<std::vector<cv::Point> >).
hierarchy	Optional output vector (e.g. std::vector<cv::Vec4i>), containing information about the image topology. It has as many elements as the number of contours. For each i-th contour contours[i], the elements hierarchy[i][0] , hierarchy[i][1] , hierarchy[i][2] , and hierarchy[i][3] are set to 0-based indices in contours of the next and previous contours at the same hierarchical level, the first child contour and the parent contour, respectively. If for the contour i there are no next, previous, parent, or nested contours, the corresponding elements of hierarchy[i] will be negative.
mode	Contour retrieval mode, see **RetrievalModes**
method	Contour approximation method, see **ContourApproximationModes**
offset	Optional offset by which every contour point is shifted. This is useful if the contours are extracted from the image ROI and then they should be analyzed in the whole image context.

Practical task 1

drawContours(), boundingRect(), rectangle(), circle()

drawContours()

https://docs.opencv.org/master/d6/d6e/group__imgproc__draw.html#ga746c0625f1781f1ffc9056259103edbc

```
void cv::drawContours ( InputOutputArray    image,
                        InputArrayOfArrays  contours,
                        int                 contourIdx,
                        const Scalar &      color,
                        int                 thickness = 1 ,
                        int                 lineType = LINE_8 ,
                        InputArray          hierarchy = noArray() ,
                        int                 maxLevel = INT_MAX ,
                        Point               offset = Point()
                      )
```

vector<vector<Point> > contours;

Contour number (0....X)

Scalar (…,…,…)

Python:

```
cv.drawContours( image, contours, contourIdx, color[, thickness[, lineType[, hierarchy[, maxLevel[, offset]]]]] ) -> image
```

boundingRect()

```
Rect cv::RotatedRect::boundingRect ( ) const
```

returns the minimal up-right integer rectangle containing the rotated rectangle

Rect bounding = boundingRect(contours.at(i)); //creates bounding rectangle around contour "i"

```
1 of 2 ▼ void rectangle(InputOutputArray img, Rect rec, const Scalar &color, int thickness = 1, int lineType = LINE_8, int shift = 0)
rectangle(rectangle1,Rect(20,20,160,160),Scalar(255),-1,8,0);
void circle(InputOutputArray img, Point center, int radius, const Scalar &color, int thickness = 1, int lineType = LINE_8, int shift = 0)
circle(circle1,cv::Point(100,100),80,Scalar(255),-1,8,0)
```

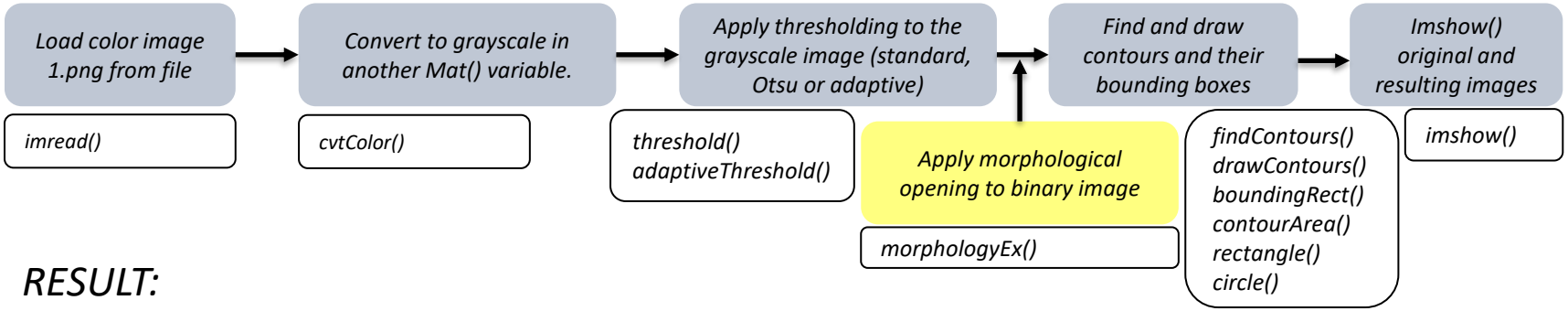
Example code for drawing random color contours and their bounding boxes in the color image:

```
RNG rng(12345); //range for random number generation
 for (size_t i = 0; i < contours.size(); i++)//for each contour
 {
      Scalar color = Scalar(rng.uniform(0, 256), rng.uniform(0, 256), rng.uniform(0, 256));                    //generate random color
      drawContours(image, contours, (int)i, color, 2, LINE_8);                                                //draw contour "i"
      Rect bounding = boundingRect(contours.at(i));                                                            //creates bounding rectangle around contour "i"
      rectangle(image,bounding,color,2,LINE_8,0);                                                              //draw bounding rectangle
      circle(image,Point(bounding.x+bounding.width/2,bounding.y+bounding.height/2),5,color,-1,LINE_8,0);  // draw center of bounding rectange
  }
```

Practical task 2

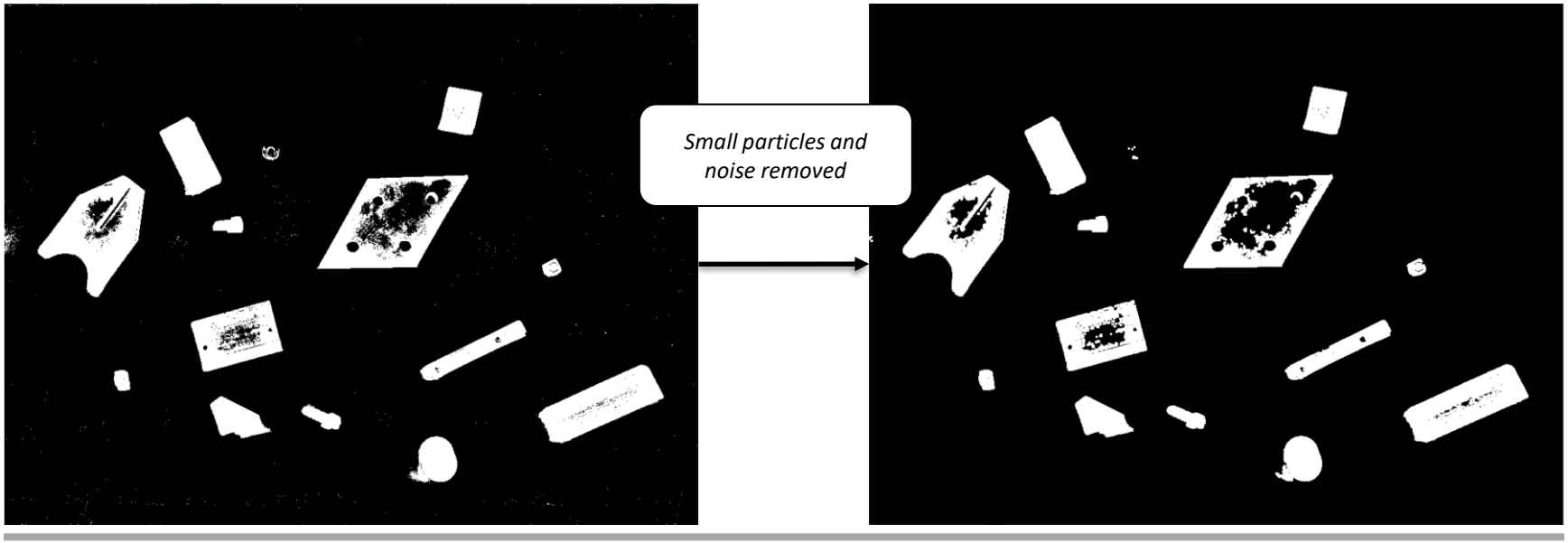
Upgrade Practical task 1 by removing small particles or parts using:

Morphological opening (use code from previous excersises)



RESULT:

adaptiveThreshold(image_gray, image_binary, 255,
ADAPTIVE_THRESH_GAUSSIAN_C, THRESH_BINARY_INV, 131, 5);



Practical task 3

Upgrade Practical task 1 by removing small particles or parts using:

Contour area based filtering

Load color image 1.png from file → *Convert to grayscale in another Mat() variable.* → *Apply thresholding to the grayscale image (standard, Otsu or adaptive)* → *Find contours, test contourArea and draw only contours and their bounding boxes larger than 500 pixels.* → *Imshow() original and resulting images*

imread()

cvtColor()

threshold()
adaptiveThreshold()

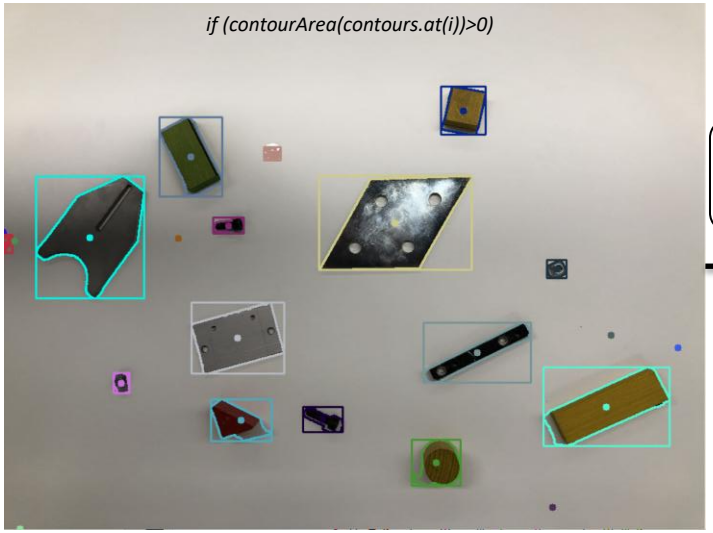
findContours()
drawContours()
boundingRect()
contourArea()
rectangle()
circle()

imshow()

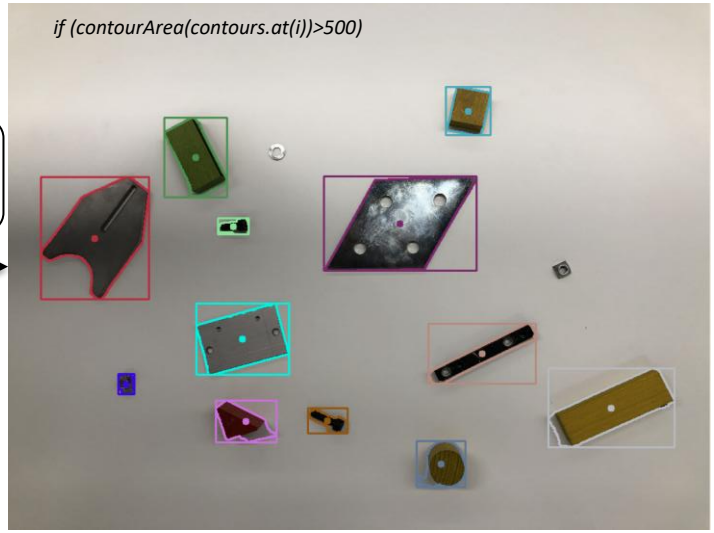
contourArea()

double cv::contourArea (InputArray contour,
bool oriented = false
)

RESULT:



Contours with small area found but not shown



Practical task 4

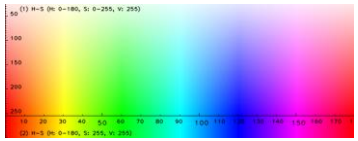
Upgrade Practical task 1 by converting the image to HSV color space and creating a binary image based on color of interest:

Load color image 1.png from file → *Convert image to HSV colorspace* → *Filter non-yellow parts using inRange function* → *Find and draw contours and their bounding boxes* → *Imshow() original and resulting images*

imread()

cvtColor(image,image,COLOR_BGR2HSV)

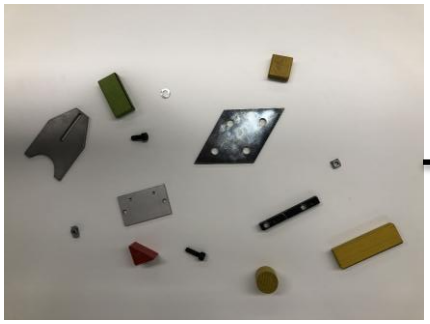
inRange()

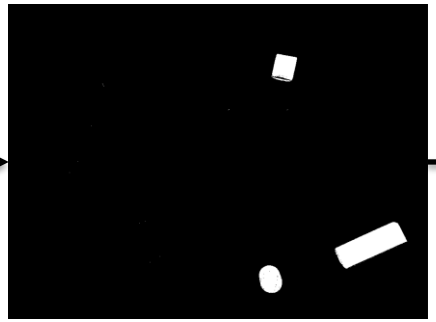


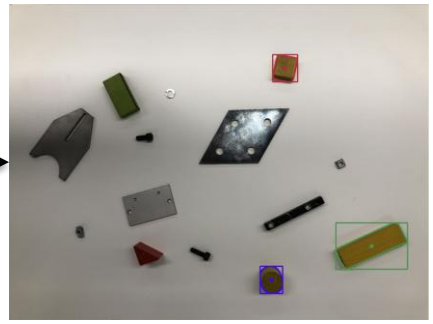
Hue is in range 0-179. Yellow hue is approximatley in range 15-30.

findContours()
drawContours()
boundingRect()
contourArea()
rectangle()
circle()

imshow()







Practical task 4

inRange()

◆ inRange()

```
void cv::inRange ( InputArray  src,
                   InputArray  lowerb,
                   InputArray  upperb,
                   OutputArray dst
                 )
```

Scalar (0-179, 0-255, 0-255)

Scalar (0-179, 0-255, 0-255)

Python:

```
cv.inRange( src, lowerb, upperb[, dst] ) -> dst
```

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

Checks if array elements lie between the elements of two other arrays.

The function checks the range as follows:

- For every element of a single-channel input array:

$$\texttt{dst} (I)= \texttt{lowerb} (I)_0 \leq \texttt{src} (I)_0 \leq \texttt{upperb} (I)_0$$

- For two-channel arrays:

$$\texttt{dst} (I)= \texttt{lowerb} (I)_0 \leq \texttt{src} (I)_0 \leq \texttt{upperb} (I)_0 \land \texttt{lowerb} (I)_1 \leq \texttt{src} (I)_1 \leq \texttt{upperb} (I)_1$$

- and so forth.

That is, dst (I) is set to 255 (all 1 -bits) if src (I) is within the specified 1D, 2D, 3D, ... box and 0 otherwise.

When the lower and/or upper boundary parameters are scalars, the indexes (I) at lowerb and upperb in the above formulas should be omitted.

Parameters

src	first input array.
lowerb	inclusive lower boundary array or a scalar.
upperb	inclusive upper boundary array or a scalar.
dst	output array of the same size as src and CV_8U type.

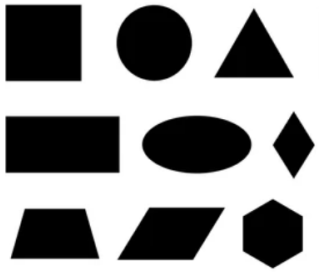
Example:
inRange(image_hsv,Scalar(15,50,50),Scalar(30,255,255),image_binary); //yellow objects only

https://docs.opencv.org/3.4/d2/de8/group__core__array.html#ga48af0ab51e36436c5d04340e036ce981

Practical task 5

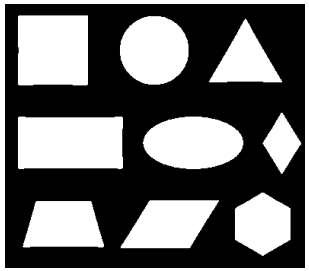
Shape detection

Load color image shapes.png from file → Convert to grayscale in another Mat() variable. → Apply thresholding to the grayscale image → Find and draw contours and approximate the number of edges → Show images, and writeout number of edges for all contours

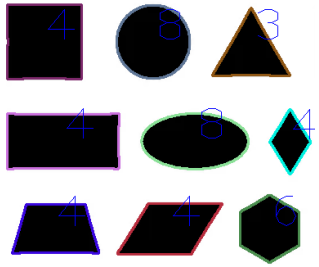


imread()

cvtColor()



threshold()
adaptiveThreshold()



findContours()
drawContours()
approxPolyDP()
putText()

imshow()

- New openCV functions are described in the following slides
- findContours is used on a binary image
- Contours can be filtered based on area size
- approxPolyDP() uses Douglas-Peucker algorithm to approximate a curve or a polygon with another curve/polygon with less vertices
- Writeout the number of edges next to any shape

Practical task 5

approxPolyDP()

https://docs.opencv.org/3.4/d3/dc0/group__imgproc__shape.html#ga0012a5fdaea70b8a9970165d98722b4c

https://docs.opencv.org/master/d3/dc0/group__imgproc__shape.html#ga819779b9857cc2f8601e6526a3a5bc71

◆ approxPolyDP()

```
void cv::approxPolyDP ( InputArray  curve,
                        OutputArray approxCurve,
                        double      epsilon,
                        bool        closed
                      )
```

A memeber contours[i] in an vector<vector<Point> > contours

vector<Point> approx;

*Can be related to contour arc lenght: arcLength(contours[i], true)*0.02*

Python:

```
cv.approxPolyDP( curve, epsilon, closed[, approxCurve] ) -> approxCurve
```

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

Approximates a polygonal curve(s) with the specified precision.

The function **cv::approxPolyDP** approximates a curve or a polygon with another curve/polygon with less vertices so that the distance between them is less or equal to the specified precision. It uses the Douglas-Peucker algorithm http://en.wikipedia.org/wiki/Ramer-Douglas-Peucker_algorithm

Parameters

curve	Input vector of a 2D point stored in std::vector or **Mat**
approxCurve	Result of the approximation. The type should match the type of the input curve.
epsilon	Parameter specifying the approximation accuracy. This is the maximum distance between the original curve and its approximation.
closed	If true, the approximated curve is closed (its first and last vertices are connected). Otherwise, it is not closed.

Examples:

samples/cpp/contours2.cpp, **samples/cpp/squares.cpp**, and **samples/tapi/squares.cpp**.

Practical task 5

drawContours(), putText()

drawContours()

https://docs.opencv.org/master/d6/d6e/group__imgproc__draw.html#ga746c0625f1781f1ffc9056259103edbc

```
void cv::drawContours ( InputOutputArray    image,
                        InputArrayOfArrays  contours,
                        int                 contourIdx,
                        const Scalar &      color,
                        int                 thickness = 1 ,
                        int                 lineType = LINE_8 ,
                        InputArray          hierarchy = noArray() ,
                        int                 maxLevel = INT_MAX ,
                        Point               offset = Point()
                      )
```

vector<vector<Point> > contours;

Contour number (0....X)

Scalar (...,...,...)

Python:

```
cv.drawContours( image, contours, contourIdx, color[, thickness[, lineType[, hierarchy[, maxLevel[, offset]]]]] ) -> image
```

putText()

```
void cv::putText ( InputOutputArray  img,
                   const String &    text,
                   Point             org,
                   int               fontFace,
                   double            fontScale,
                   Scalar            color,
                   int               thickness = 1 ,
                   int               lineType = LINE_8 ,
                   bool              bottomLeftOrigin = false
                 )
```

Python:

```
cv.putText( img, text, org, fontFace, fontScale, color[, thickness[, lineType[, bottomLeftOrigin]]] ) -> img
```

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

Draws a text string.

The function **cv::putText** renders the specified text string in the image. Symbols that cannot be rendered using the specified font are replaced by question marks. See **getTextSize** for a text rendering code example.

Parameters

img	Image.
text	Text string to be drawn.
org	Bottom-left corner of the text string in the image.
fontFace	Font type, see **HersheyFonts**.
fontScale	Font scale factor that is multiplied by the font-specific base size.
color	Text color.
thickness	Thickness of the lines used to draw a text.
lineType	Line type. See **LineTypes**
bottomLeftOrigin	When true, the image data origin is at the bottom-left corner. Otherwise, it is at the top-left corner.

Example:

putText(image,"text",Point2i(100,100),1,4,Scalar(255,0,0),1,LINE_8);

Practical task 5

Use basic code from prior excersise to get a faster start:

```
int main(){
 Mat image=imread("1.png",1);
 Mat image_gray=imread("1.png",0);
 Mat image_binary,image_hsv;
//threshold(image_gray,image_binary,100,255,THRESH_BINARY_INV+THRESH_OTSU);
 //threshold(image_gray,image_binary,100,255,THRESH_BINARY_INV);
 adaptiveThreshold(image_gray, image_binary, 255, ADAPTIVE_THRESH_GAUSSIAN_C, THRESH_BINARY_INV, 131, 5);
 vector<vector<Point> > contours;
 findContours(image_binary,contours,RETR_EXTERNAL,1);
 RNG rng(12345); //range for random number generation
 for (size_t i = 0; i < contours.size(); i++)//for each contour
 {
   if (contourArea(contours.at(i))>100) //filter for removing small contours
   {
      Scalar color = Scalar(rng.uniform(0, 256), rng.uniform(0, 256), rng.uniform(0, 256)); //generate random color
      drawContours(image, contours, (int)i, color, 2, LINE_8);//draw contour "i"
      Rect bounding = boundingRect(contours.at(i)); //creates bounding rectangle around contour "i"

          // approximate contour with accuracy proportional to the contour perimeter
   }
 }
  imshow("Color", image);
  imshow("Gray", image_gray);
  imshow("Binary", image_binary);
 waitKey(0);
 return 1;
}
```

Practical task 6

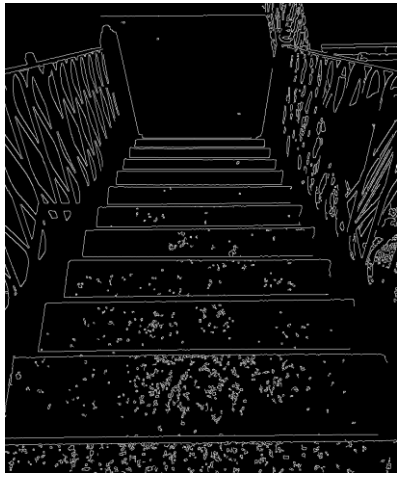
Detect lines

Load color image lines.png from file	Convert to grayscale in another Mat() variable.	Apply Canny edge detection algorithm	Find lines	Draw lines	Imshow() original and resulting images
imread()	cvtColor()	Canny()	HoughLines() HoughLinesP()	line()	imshow()

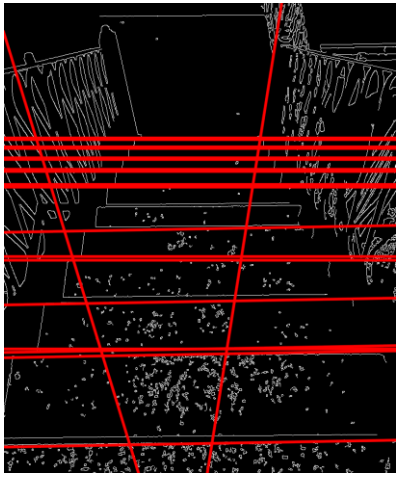
RESULT:

Grayscale

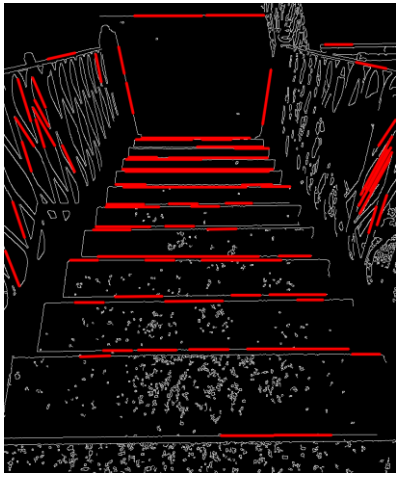
Canny



Houghlines



HoughlinesP



Practical task 6

HoughLines(), HoughLinesP()

https://docs.opencv.org/3.4/dd/d1a/group__imgproc__feature.html#ga46b4e588934f6c8dfd509cc6e0e4545a

HoughLines()

```
void cv::HoughLines ( InputArray  image,
                      OutputArray lines,
                      double      rho,
                      double      theta,
                      int         threshold,
                      double      srn = 0,
                      double      stn = 0,
                      double      min_theta = 0,
                      double      max_theta = CV_PI
                    )
```

Python:

```
cv.HoughLines( image, rho, theta, threshold[, lines[, srn[, stn[, min_theta[, max_theta]]]]] ) -> lines
```

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

Finds lines in a binary image using the standard Hough transform.

The function implements the standard or standard multi-scale Hough transform algorithm for line detection. See http://homepages.inf.ed.ac.uk/rbf/HIPR2/hough.htm for a good explanation of Hough transform.

Parameters

image	8-bit, single-channel binary source image. The image may be modified by the function.
lines	Output vector of lines. Each line is represented by a 2 or 3 element vector (ρ, θ) or $(\rho, \theta, \text{votes})$. ρ is the distance from the coordinate origin $(0,0)$ (top-left corner of the image). θ is the line rotation angle in radians ($0 \sim \text{vertical line}, \pi/2 \sim \text{horizontal line}$). votes is the value of accumulator.
rho	Distance resolution of the accumulator in pixels.
theta	Angle resolution of the accumulator in radians.
threshold	Accumulator threshold parameter. Only those lines are returned that get enough votes ($>$ `threshold`).
srn	For the multi-scale Hough transform, it is a divisor for the distance resolution rho . The coarse accumulator distance resolution is rho and the accurate accumulator resolution is rho/srn . If both srn=0 and stn=0 , the classical Hough transform is used. Otherwise, both these parameters should be positive.
stn	For the multi-scale Hough transform, it is a divisor for the distance resolution theta.
min_theta	For standard and multi-scale Hough transform, minimum angle to check for lines. Must fall between 0 and max_theta.
max_theta	For standard and multi-scale Hough transform, maximum angle to check for lines. Must fall between min_theta and CV_PI.

Example:
HoughLines(dst, lines, 1, CV_PI/180, 180, 0, 0);

HoughLinesP()

```
void cv::HoughLinesP ( InputArray  image,
                       OutputArray lines,
                       double      rho,
                       double      theta,
                       int         threshold,
                       double      minLineLength = 0,
                       double      maxLineGap = 0
                     )
```

Python:

```
cv.HoughLinesP( image, rho, theta, threshold[, lines[, minLineLength[, maxLineGap]]] ) -> lines
```

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

Finds line segments in a binary image using the probabilistic Hough transform.

The function implements the probabilistic Hough transform algorithm for line detection, described in [147]

Parameters

image	8-bit, single-channel binary source image. The image may be modified by the function.
lines	Output vector of lines. Each line is represented by a 4-element vector (x_1, y_1, x_2, y_2), where (x_1, y_1) and (x_2, y_2) are the ending points of each detected line segment.
rho	Distance resolution of the accumulator in pixels.
theta	Angle resolution of the accumulator in radians.
threshold	Accumulator threshold parameter. Only those lines are returned that get enough votes ($>$ `threshold`).
minLineLength	Minimum line length. Line segments shorter than that are rejected.
maxLineGap	Maximum allowed gap between points on the same line to link them.

Example:
HoughLinesP(dst, linesP, 1, CV_PI/180, 50, 50 , 1);

Practical task 6

Draw lines based on HoughLines output:

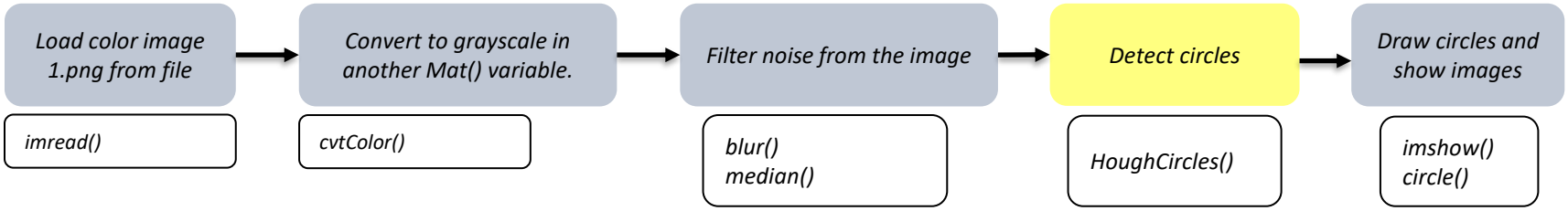
```
vector<Vec2f> lines; // will hold the results of the detection
HoughLines(dst, lines, 1, CV_PI/180, 180, 0, 0 ); // runs the actual detection
// Draw the lines
for( size_t i = 0; i < lines.size(); i++ )
{
   float rho = lines[i][0], theta = lines[i][1];
   Point pt1, pt2;
   double a = cos(theta), b = sin(theta);
   double x0 = a*rho, y0 = b*rho;
   pt1.x = cvRound(x0 + 1000*(-b));
   pt1.y = cvRound(y0 + 1000*(a));
   pt2.x = cvRound(x0 - 1000*(-b));
   pt2.y = cvRound(y0 - 1000*(a));
   line( cdst, pt1, pt2, Scalar(0,0,255), 3, LINE_AA);
}
```

Draw lines based on HoughLinesP output:

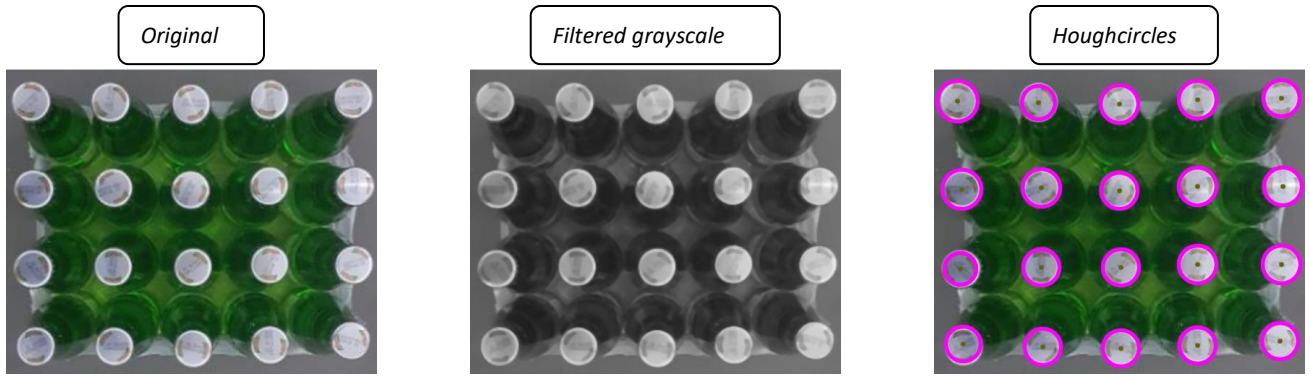
```
// Probabilistic Line Transform
vector<Vec4i> linesP; // will hold the results of the detection
HoughLinesP(dst, linesP, 1, CV_PI/180, 50, 50 , 1 ); // runs the actual detection
// Draw the lines
for( size_t i = 0; i < linesP.size(); i++ )
{
   Vec4i l = linesP[i];
   line( cdstP, Point(l[0], l[1]), Point(l[2], l[3]), Scalar(0,0,255), 3, LINE_AA);
}
```

Practical task 7

Detect all the bottle caps as circles



RESULT:



Practical task 7

HoughCircles()

HoughCircles()

```
void cv::HoughCircles ( InputArray  image,
                        OutputArray circles,
                        int         method,
                        double      dp,
                        double      minDist,
                        double      param1 = 100 ,
                        double      param2 = 100 ,
                        int         minRadius = 0 ,
                        int         maxRadius = 0
                      )
```

Python:

```
cv.HoughCircles( image, method, dp, minDist[, circles[, param1[, param2[, minRadius[, maxRadius]]]]] ) -> circles
```

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

Finds circles in a grayscale image using the Hough transform.

The function finds circles in a grayscale image using a modification of the Hough transform.

Note

Usually the function detects the centers of circles well. However, it may fail to find correct radii. You can assist to the function by specifying the radius range (minRadius and maxRadius) if you know it. Or, you may set maxRadius to a negative number to return centers only without radius search, and find the correct radius using an additional procedure.

Parameters

image	8-bit, single-channel, grayscale input image.
circles	Output vector of found circles. Each vector is encoded as 3 or 4 element floating-point vector $(x, y, radius)$ or $(x, y, radius, votes)$.
method	Detection method, see **HoughModes**. Currently, the only implemented method is **HOUGH_GRADIENT**
dp	Inverse ratio of the accumulator resolution to the image resolution. For example, if dp=1 , the accumulator has the same resolution as the input image. If dp=2 , the accumulator has half as big width and height.
minDist	Minimum distance between the centers of the detected circles. If the parameter is too small, multiple neighbor circles may be falsely detected in addition to a true one. If it is too large, some circles may be missed.
param1	First method-specific parameter. In case of **HOUGH_GRADIENT** , it is the higher threshold of the two passed to the Canny edge detector (the lower one is twice smaller).
param2	Second method-specific parameter. In case of **HOUGH_GRADIENT** , it is the accumulator threshold for the circle centers at the detection stage. The smaller it is, the more false circles may be detected. Circles, corresponding to the larger accumulator values, will be returned first.
minRadius	Minimum circle radius.
maxRadius	Maximum circle radius. If <= 0, uses the maximum image dimension. If < 0, returns centers without finding the radius.

Example:

```
HoughCircles(gray, circles, HOUGH_GRADIENT, 1,
             gray.rows/16,  // change this value to detect circles with different distances to each other
             100, 30, 1, 30 // change the last two parameters
           // (min_radius & max_radius) to detect larger circles
      );
```

https://docs.opencv.org/3.4/dd/d1a/group__imgproc__feature.html#ga47849c3be0d0406ad3ca45db65a25d2d

Practical task 7

Draw circles based on the output from the HoughCricles:

Example:

```
for( size_t i = 0; i < circles.size(); i++ )
      {
         Vec3i c = circles[i];
         Point center = Point(c[0], c[1]); // circle center
         circle( src, center, 1, Scalar(0,100,100), 3, LINE_AA); // circle outline
         int radius = c[2];
         circle( src, center, radius, Scalar(255,0,255), 3, LINE_AA);
      }
```

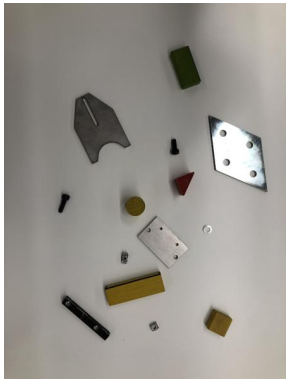
https://docs.opencv.org/3.4/dd/d1a/group__imgproc__feature.html#ga47849c3be0d0406ad3ca45db65a25d2d

Student assignment (seminar)

Student assignment - seminar

Shape detection – test the algorithm in a realistic example

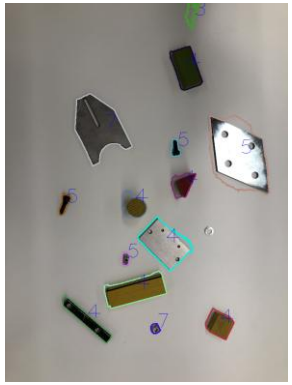
Load color image shapes.png from file → *Convert to grayscale in another Mat() variable.* → *Apply thresholding to the grayscale image* → *Find and draw contours and approximate the number of edges* → *Show images, and write out the number of edges for all contours*



imread()

cvtColor()

threshold()
adaptiveThreshold()



findContours()
drawContours()
approxPolyDP()
putText()

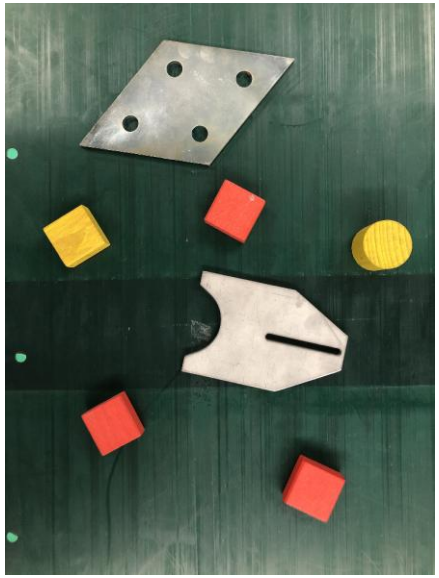
imshow()

- *approxPolyDP() uses Douglas-Peucker algorithm to approximate a curve or a polygon with another curve/polygon with less vertices*
- *Write out the number of edges next to any shape*
- *NOTICE → image noise and approximation epsilon can give wrong results, test different settings to get best results*

Student assignment - seminar

Add trackbar and change the threshold value and inRange lower and upper limits, for Practical tasks 1, 2, 3 and 4.

Test out the object localization on other images („2.png” and „3.png”):



For trackbar use example :

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

Student assignment - seminar

Detect lines – test the algorithm on road.jpg example and change the parameters (canny edge, hough lines,…) to get better results