Shapes and Vanishing Point Detection

Computer Vision course, a.y. 2021/2022

Del Col Jacopo Edoardo

Doveri Riccardo

INTRODUCTION

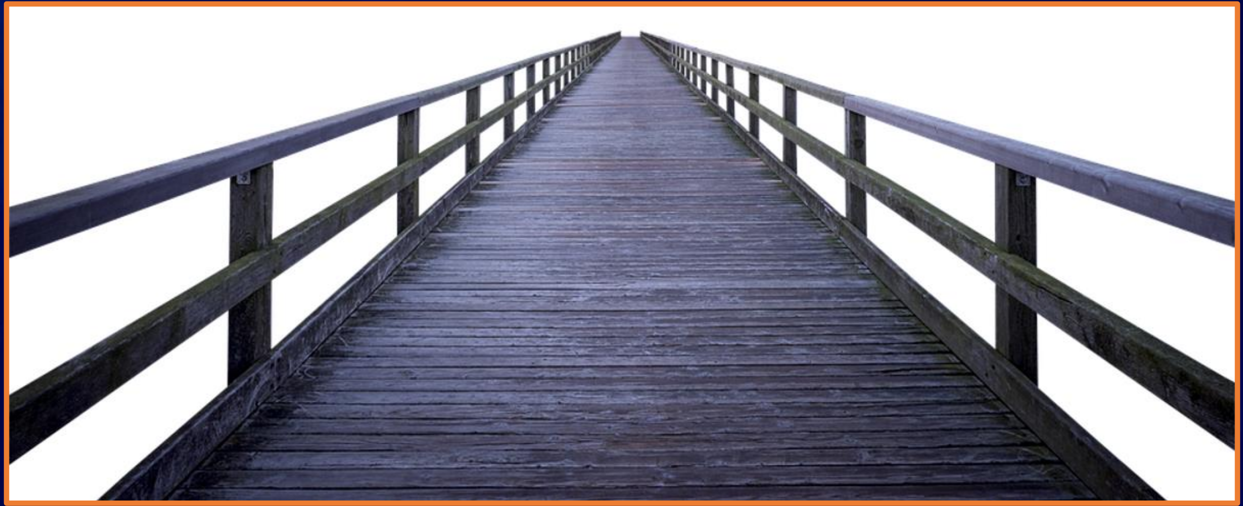
The aim of this project is the implementation of a program that can solve two different computer vision problems:

- Detection of the vanishing point
- Circle detection

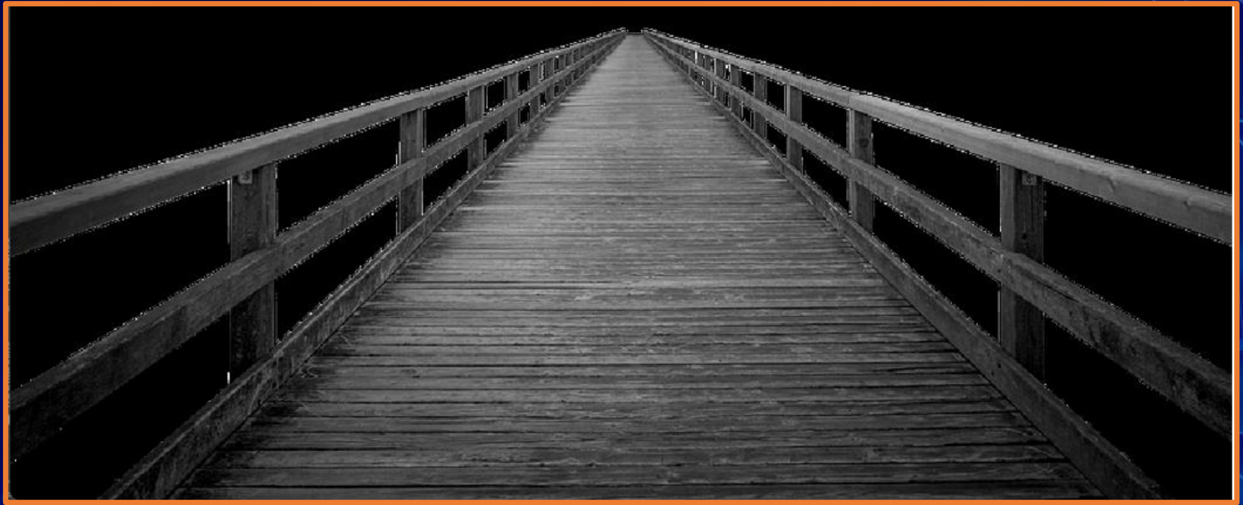
All the code is in Java programming language and the only library used is OpenCV (Open Source Computer Vision Library), a library of programming functions used for image processing and computer vision tasks.

VANISHING POINT

Starting from an appropriate image, the first step is to obtain the corresponding greyscale image.



Original image



Greyscale image

VANISHING POINT

Then smoothing and Canny Edge Detector are applied to the greyscale image.

Canny Edge Detector (CED) is a technique used to detect the edges of an image. Its algorithm can be summarized in four steps:

- Apply Gaussian filter to smooth the image and reduce noise
- Find magnitude and orientation of the gradient
- Perform non-maximum suppression to thin multi-pixel wide ridges
- Apply hysteresis defining two thresholds to track the edges

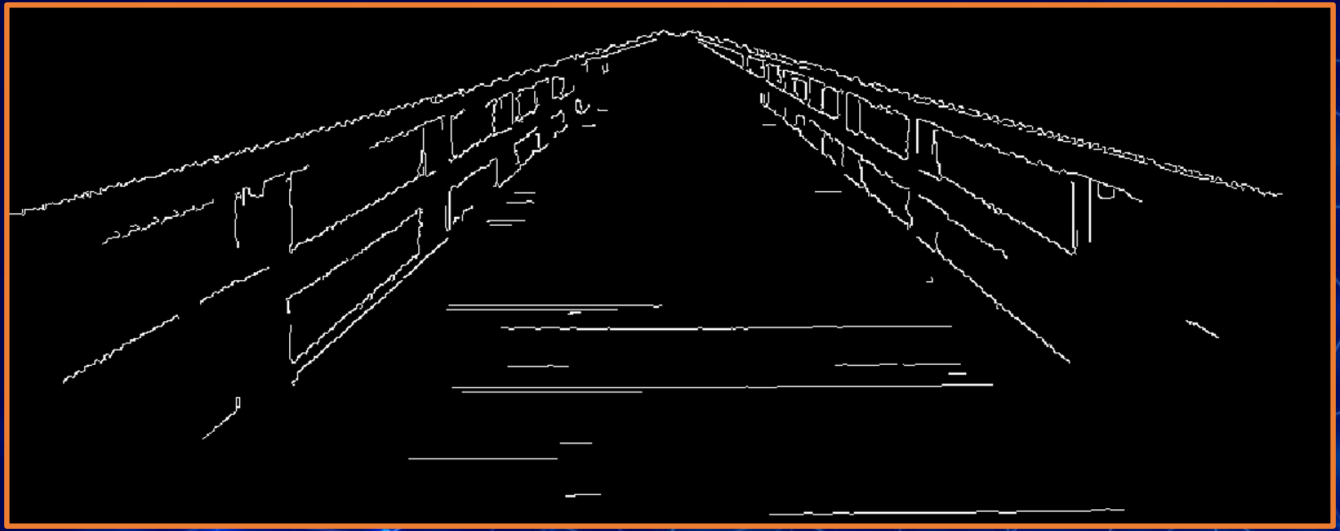


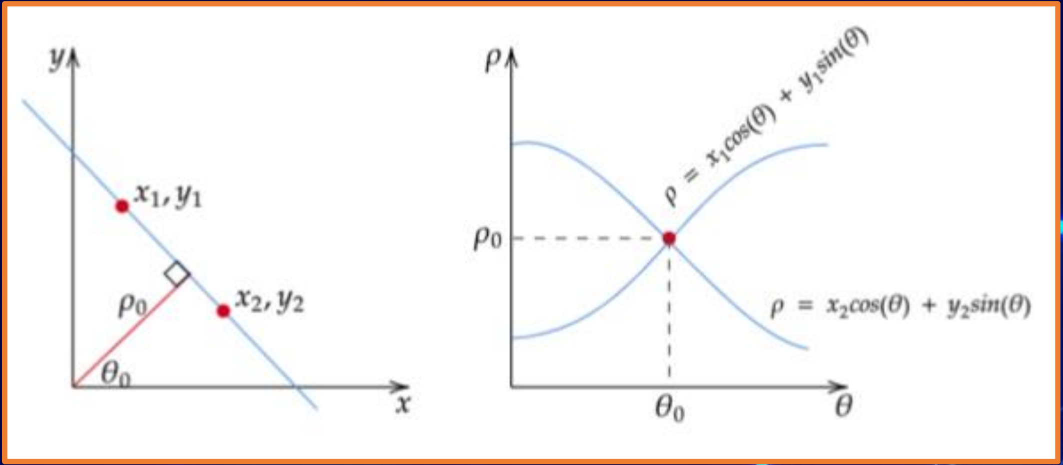
Image passed through CED

VANISHING POINT

The image is then passed through a Hough Transform to detect straight lines, which are represented in polar coordinates:

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

This introduces a new parameter space (ρ, θ) where $0 < \rho < L\sqrt{2}$ and $-\pi < \theta < \pi$, in the case of a $L \times L$ image.



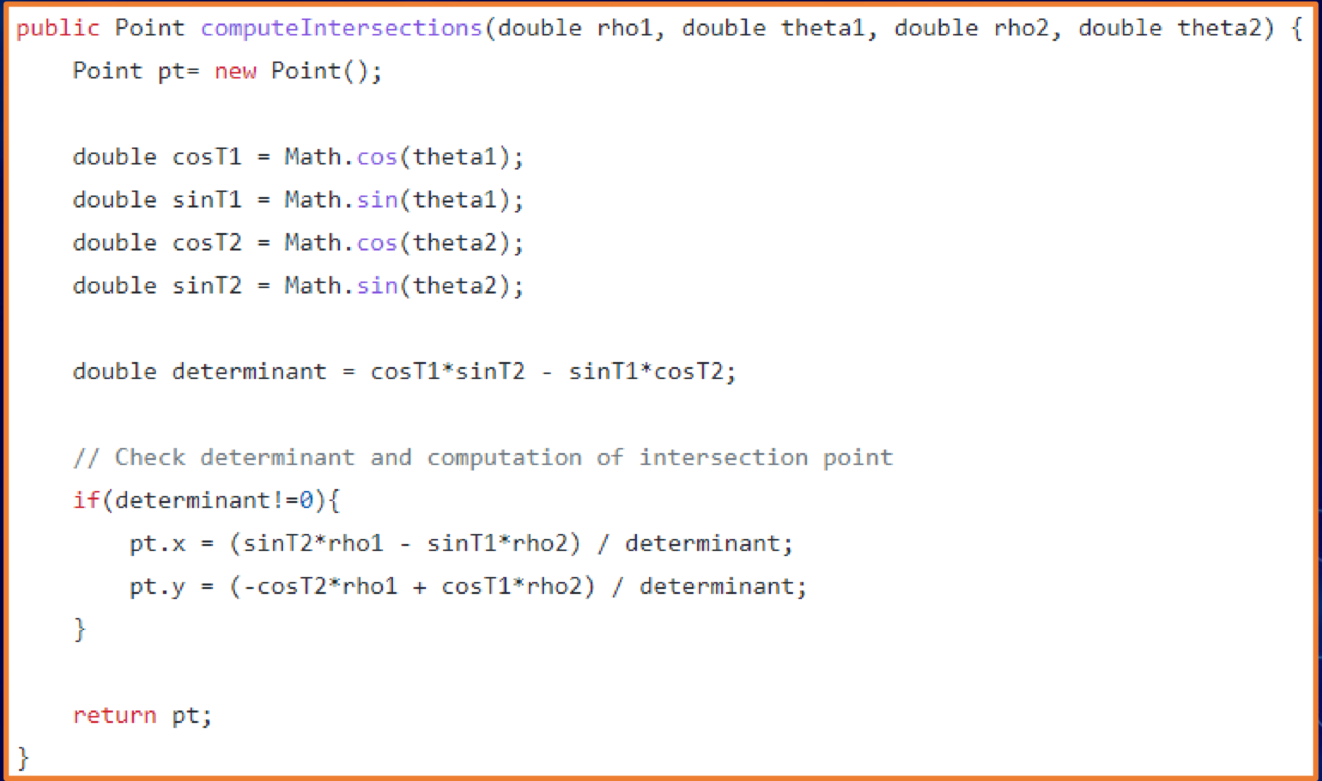
y
x_1, y_1
ρ_0
x_2, y_2
θ_0
x
ρ
$\rho = x_1\cos(\theta) + y_1\sin(\theta)$
ρ_0
$\rho = x_2\cos(\theta) + y_2\sin(\theta)$
θ_0
θ

Representation of the new parameter space

If two edge points lay on the same line, their corresponding cosine curves will intersect each other on a specific (ρ, θ) pair. Thus, the Hough Transform algorithm detects lines by finding the (ρ, θ) pairs that have a number of intersections larger than a certain threshold, which can be decided by the program user from command line.

VANISHING POINT

In the last two steps, the intersection between lines are first computed using the Cramer’s resolution for linear systems…



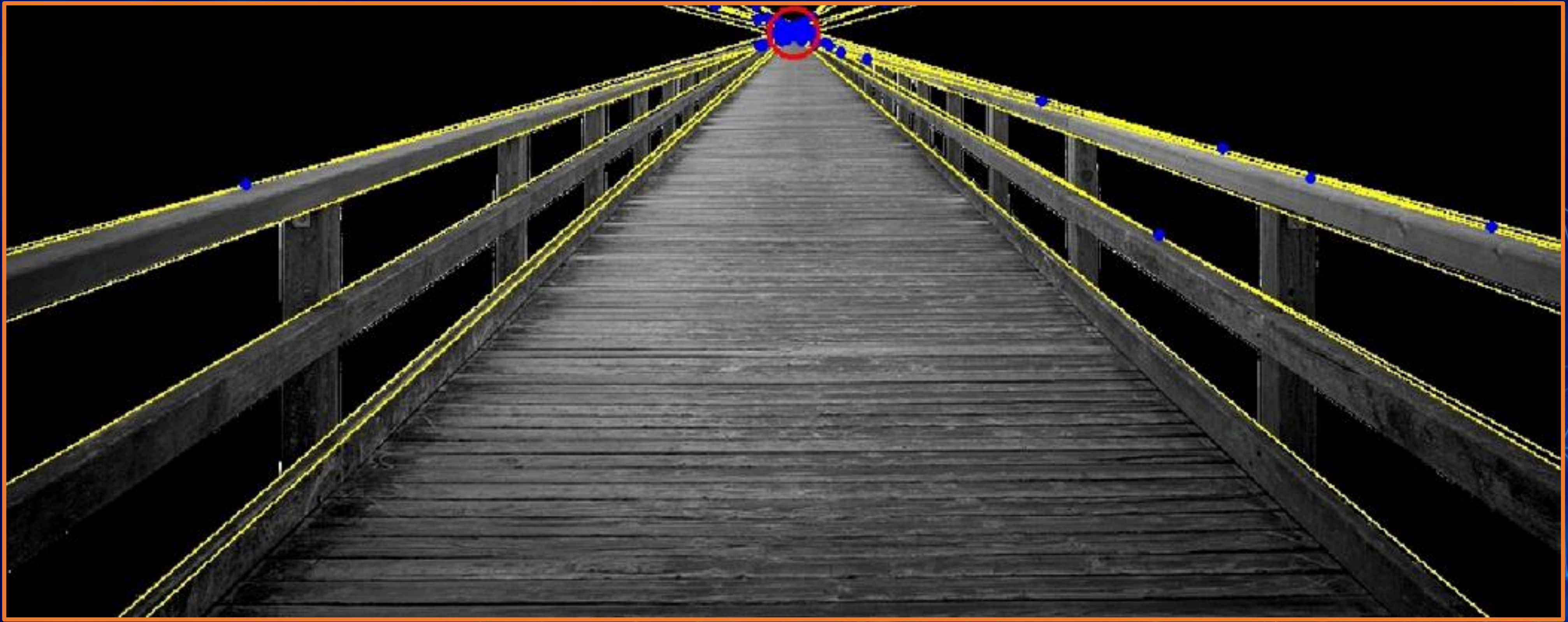
```
public Point computeIntersections(double rho1, double theta1, double rho2, double theta2) {
    Point pt= new Point();

    double cosT1 = Math.cos(theta1);
    double sinT1 = Math.sin(theta1);
    double cosT2 = Math.cos(theta2);
    double sinT2 = Math.sin(theta2);

    double determinant = cosT1*sinT2 - sinT1*cosT2;

    // Check determinant and computation of intersection point
    if(determinant!=0){
        pt.x = (sinT2*rho1 - sinT1*rho2) / determinant;
        pt.y = (-cosT2*rho1 + cosT1*rho2) / determinant;
    }

    return pt;
}
```

Computing the intersection point using Cramer’s method. The lines are represented in polar coordinates using rho and theta parameters.

… and the vanishing point is then chosen as the point with minimum distance from the others.

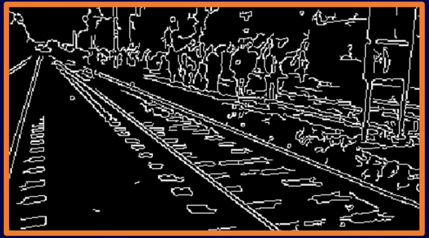
VANISHING POINT

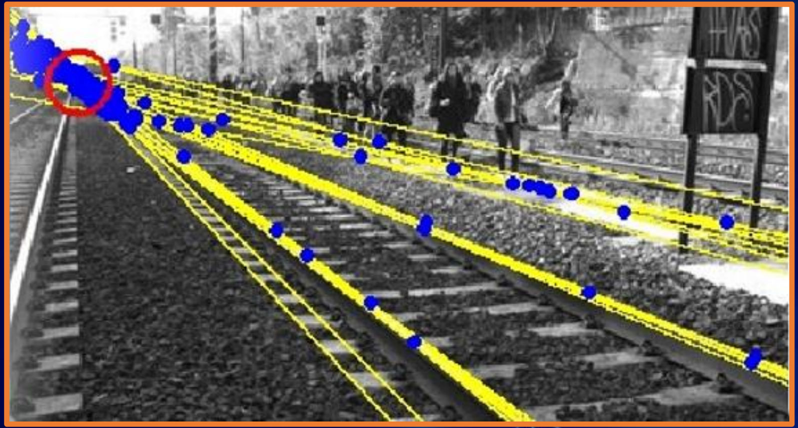
Here the intersection points are colored in blue, while the vanishing point is pointed out with a red circle.

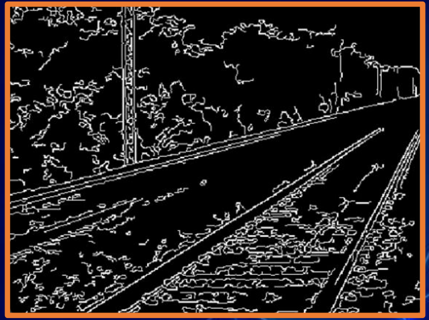


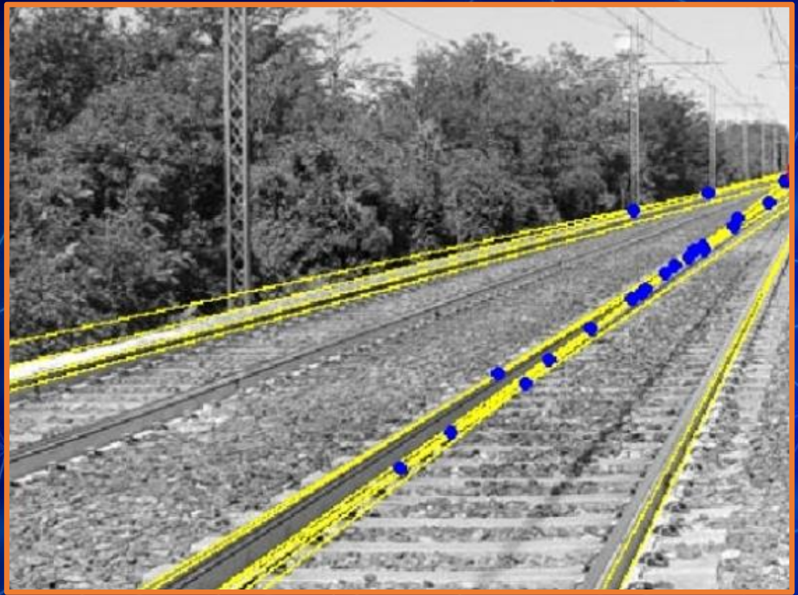
Vanishing point

VANISHING POINT - EXAMPLES









Noticed how, in this case, the vanishing point is exterior to the image

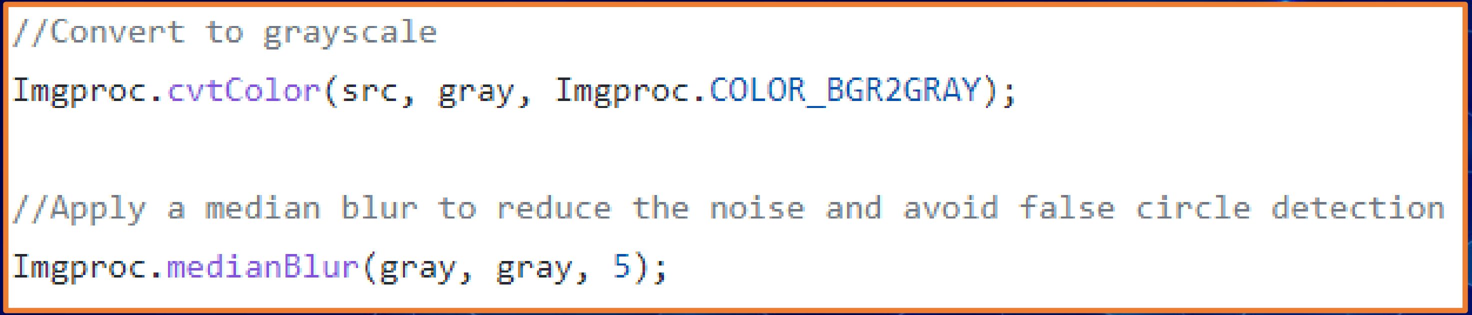
CIRCLE DETECTION

The goal here was to create an algorithm that could recognise certain shape inside an image, in this particular case circles.

Similarly to what previously done, the first step performed is to convert the original image in a greyscale image.

Then the resulting image is processed through a median filter in order to reduce noise.

This filter, which is a particular case of rank filter, operates over a window assigning to a pixel the median value of the neighborhood.



```
//Convert to grayscale
Imgproc.cvtColor(src, gray, Imgproc.COLOR_BGR2GRAY);

//Apply a median blur to reduce the noise and avoid false circle detection
Imgproc.medianBlur(gray, gray, 5);
```

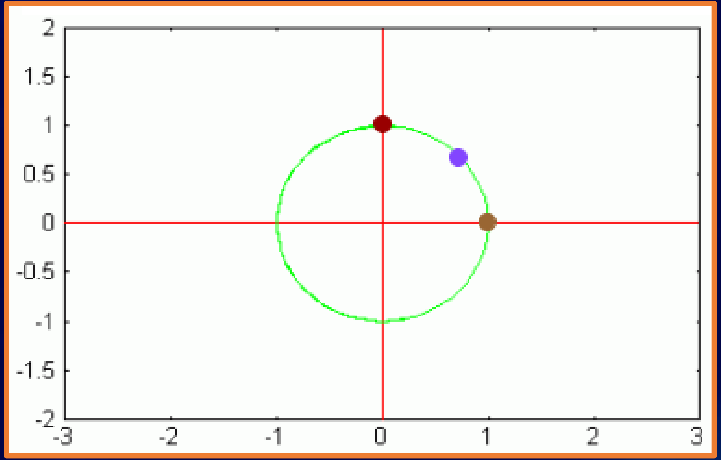
Code snippet for color conversion and median filter application

CIRCLE DETECTION

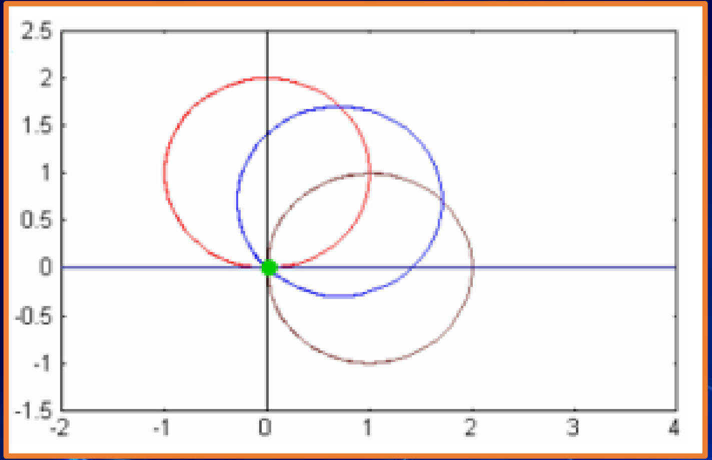
Hough Transform can be used to search for complex analytical curves too.

At this point a Hough Transform was then applied in order to detect the circles present in the image.

Each point of the image is considered as the center of a circle of radius R. The three circles intersect in the center of the original image circumference.

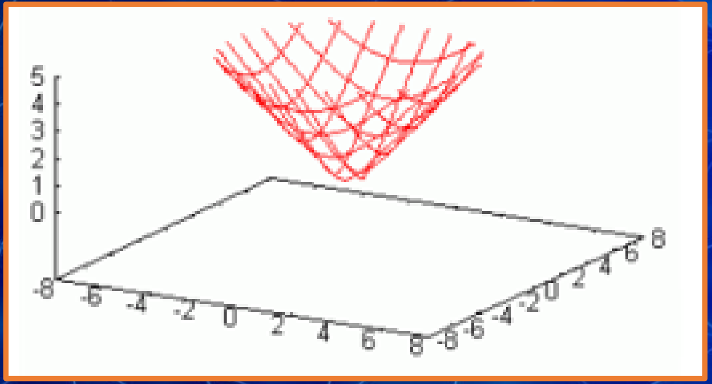


2D image space



2D parameter space

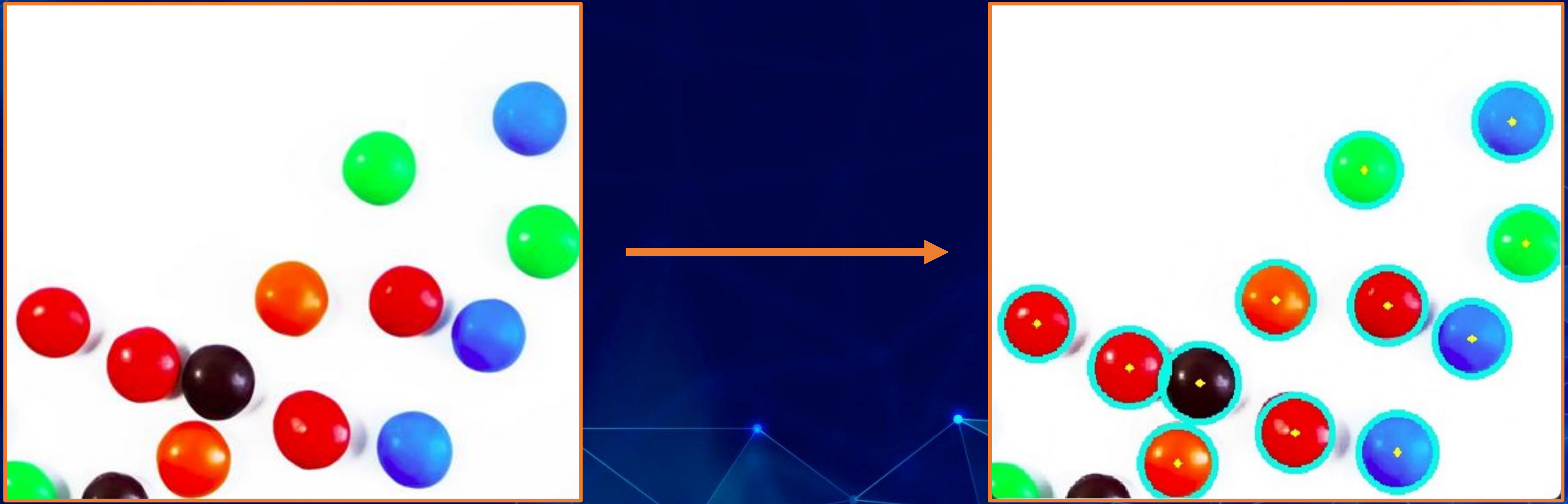
While in the case of an unknown radius, such as the present case, the mapping rule is a 3D cone.



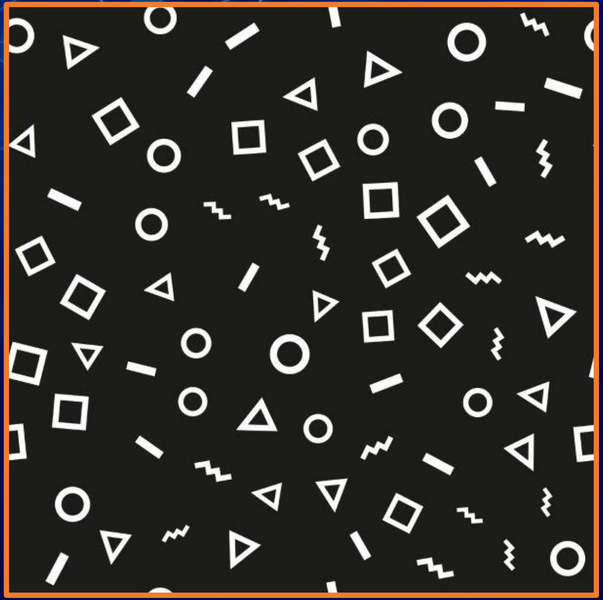
3D parameter space

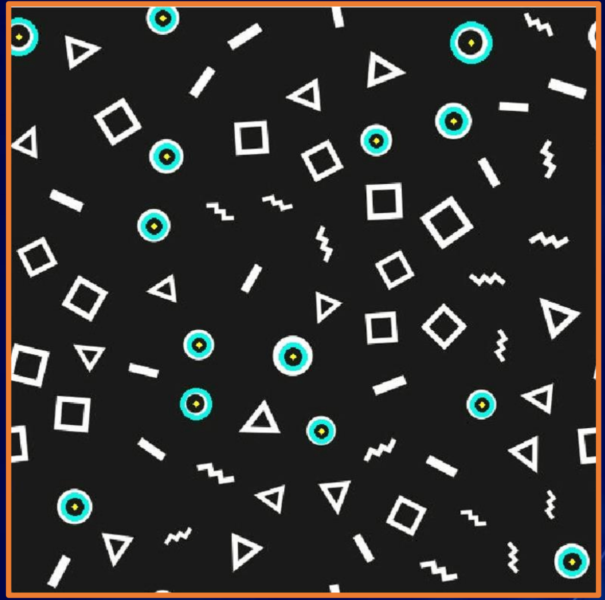
CIRCLE DETECTION

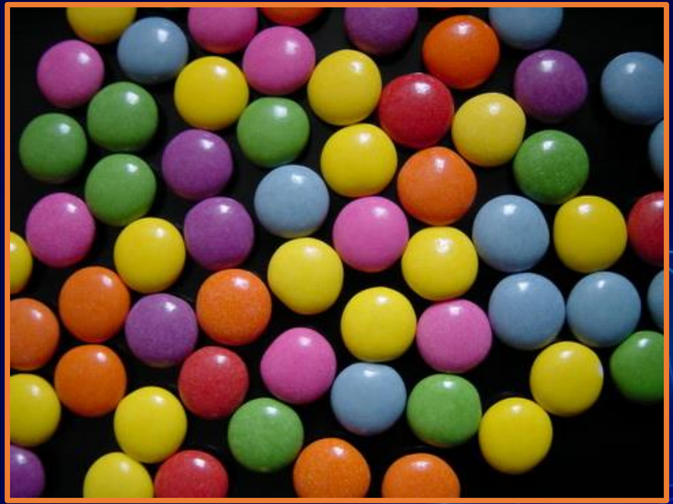
Finally the circles are highlighted in the original image, here in cyan.

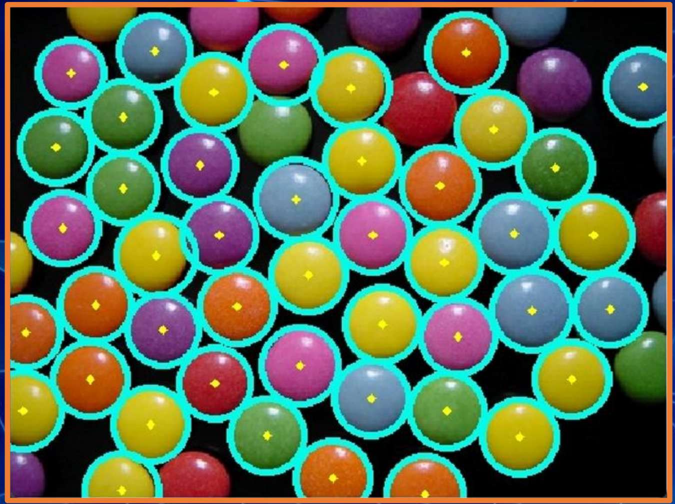


CIRCLE DETECTION - EXAMPLES









MODALITY OF USE

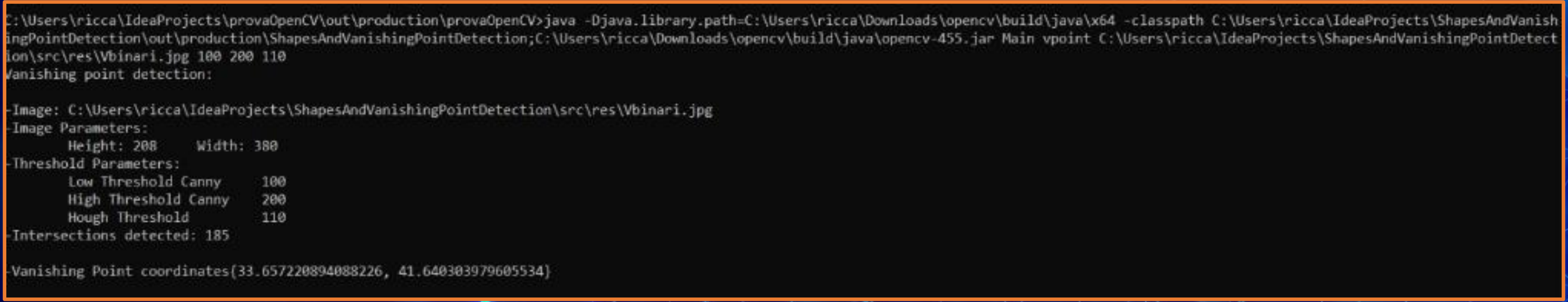
The command syntax to run the program on Windows is the following:

java [opencv path] [class path];[opencv-455.jar path] Main [vpoint or circles] [optional parameters]

The *optional parameters* are the following:

- Low Threshold for CED
- High Threshold for CED
- Hough Transform Threshold

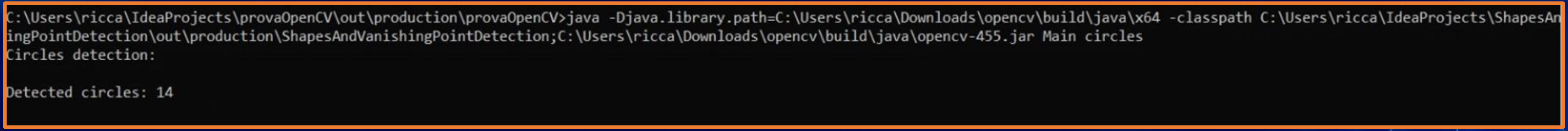
Output of the command line for vanishing point detection



```
:\Users\ricca\IdeaProjects\provaOpenCV\out\production\provaOpenCV>java -Djava.library.path=C:\Users\ricca\Downloads\opencv\build\java\x64 -classpath C:\Users\ricca\IdeaProjects\ShapesAndVanish
ingPointDetection\out\production\ShapesAndVanishingPointDetection;C:\Users\ricca\Downloads\opencv\build\java\opencv-455.jar Main vpoint C:\Users\ricca\IdeaProjects\ShapesAndVanishingPointDetect
ion\src\res\Vbinari.jpg 100 200 110
anishing point detection:

-Image: C:\Users\ricca\IdeaProjects\ShapesAndVanishingPointDetection\src\res\Vbinari.jpg
-Image Parameters:
        Height: 208     Width: 380
-Threshold Parameters:
        Low Threshold Canny      100
        High Threshold Canny     200
        Hough Threshold          110
-Intersections detected: 185

-Vanishing Point coordinates(33.657220894088226, 41.640303979605534)
```

MODALITY OF USE

The command line input in the case of circle detection is similar to the previous one, with the only difference that no threshold parameters are required.



```
C:\Users\ricca\IdeaProjects\provaOpenCV\out\production\provaOpenCV>java -Djava.library.path=C:\Users\ricca\Downloads\opencv\build\java\x64 -classpath C:\Users\ricca\IdeaProjects\ShapesAn
ingPointDetection\out\production\ShapesAndVanishingPointDetection;C:\Users\ricca\Downloads\opencv\build\java\opencv-455.jar Main circles
Circles detection:

Detected circles: 14
```

Output of the command line for circle detection

The output is instead different: the number of detected circles is printed here.

CONCLUSIONS

In this project a program, able to both detect vanishing points and circles, was created.

The final product has been tested on multiple images, always obtaining good results.

Different values of parameters, which can be customized by the user, were also evaluated in order to compare the outcomes and to determine which ones are the best, which were then set as the default values.

The program can be run both on Windows and on Linux and was tested on both.
Also, the structure of the code makes it really easy to further implement additional functionalities, e.g. other shapes detection.