# Computer Vision Portfolio Template

*Author name:* Matúš Stanko
*Education*: UCL Seebladsgade, IT

---

## Table of contents

# Introduction

## Use case 1

Problem statement: During this use case I will use computer vision to detect coins in a picture. Then determinate which coin corresponds to its value and finally calculate the sum of the different coins.

## Use case 2

Problem statement: During case number 2 I will use computer vision to detect light color on a traffil light. I will use picture with multiple traffic lights in background to demonstrate that my program is able to recognize them as well

## Use case 3

Problem statement: Last use case will be about detecting licence plates on a different cars in a video. I will use my own video with 3 different cars with different licence plates.

# Methodology and process

## Use case 1: Coins detector and calculator

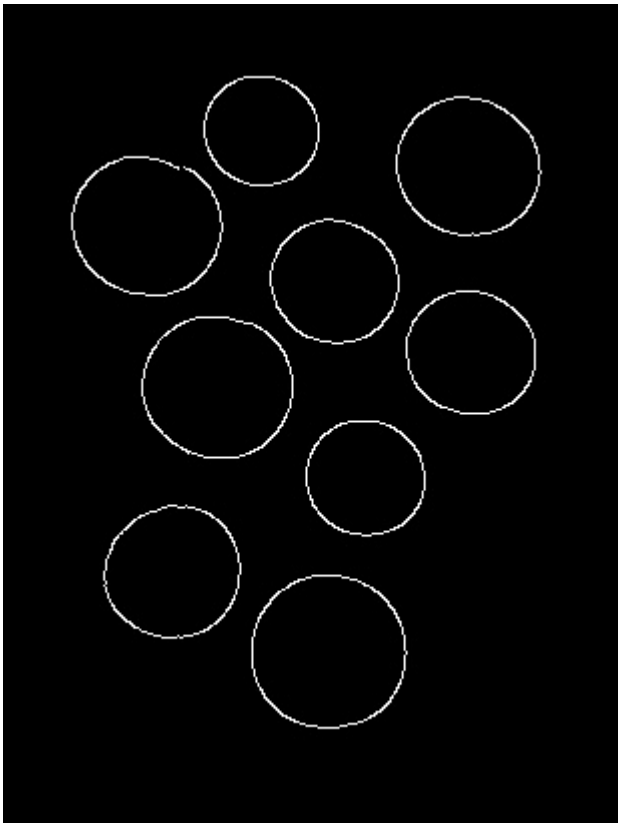
Firstly image will be loaded into the program

Now original image will be converted to grayscale. It is because Its easier for algorithms. Also here, I don't need filtering based on colors, but I will use edges.

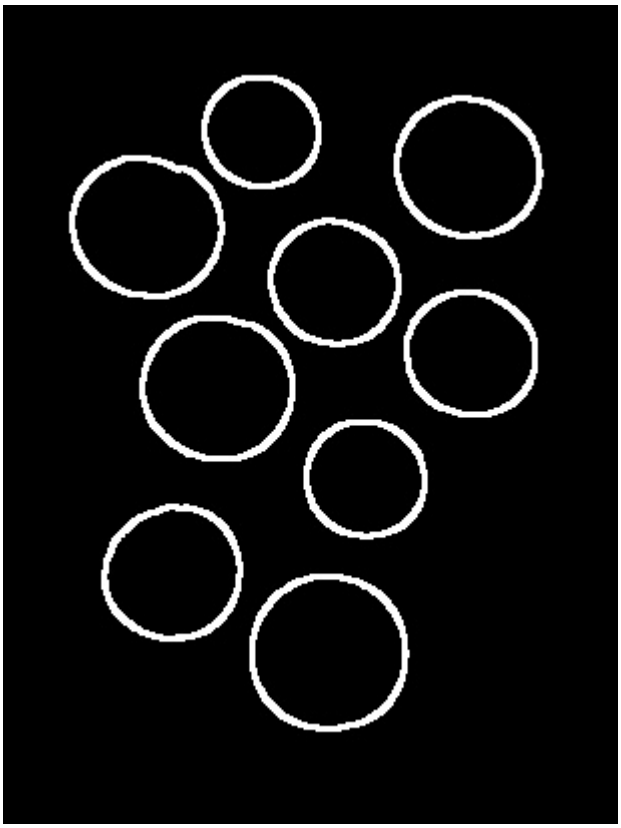
After converting an image to grayscale, we can apply Gaussian blur to filter out the noise kernel size of 17 by

17. This will make image smooth. Gaussian blur makes all pixels in the image be an average of the neighbouring pixel values and itself, and thereby smooths out the image.

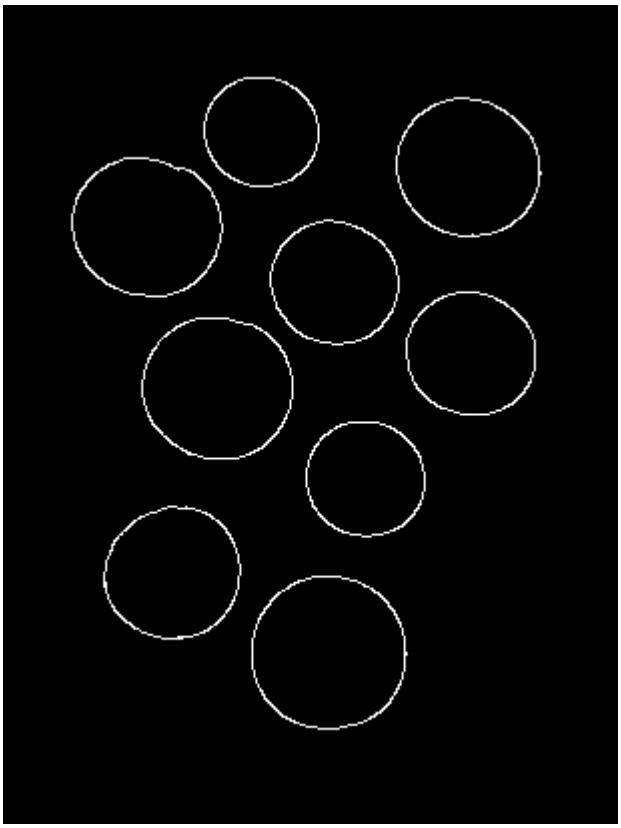
Now I am ready to use Canny edge detection. This algorithm will reduce noise and will find edges in pictures. As a Thresholding point I will use 30 and 90. This will keep edges that are important to me and reduce some useless noise in the picture.



After using Cannys edge detection, I found edges. But some of them are not following the coin shape completely. Therefor, I will use functions: dilate and erode. Those functions are working as a brother, and they will join edges that are similar and delete alone edges. Kernel size of 3 x 3 is being used.

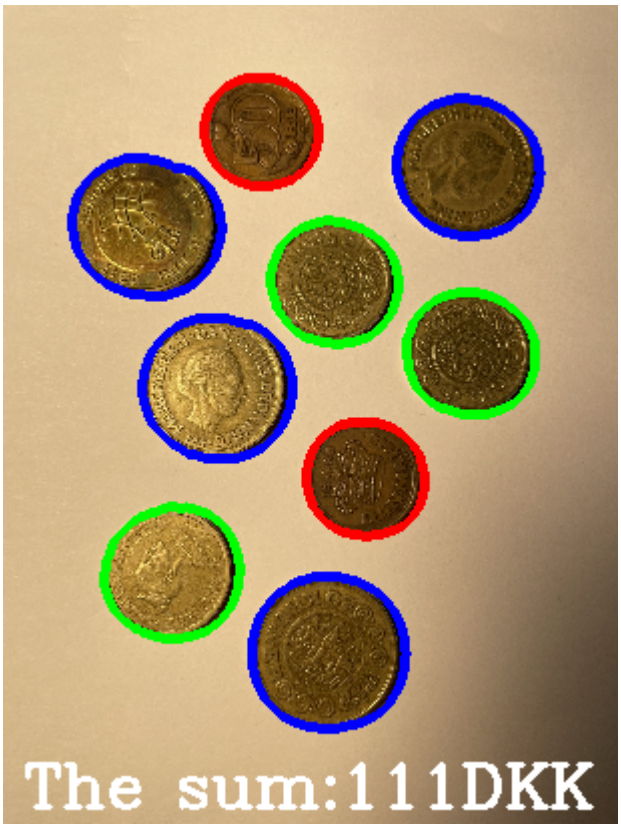


Now it's time to erode.



Finally, I can draw contours in the original image.

I will calculate area of each contour. Based on are the sizes I can find out what coin is where and I can count them. Also, I will use different colors to draw contours around different coins. The sum will be written in the picture.

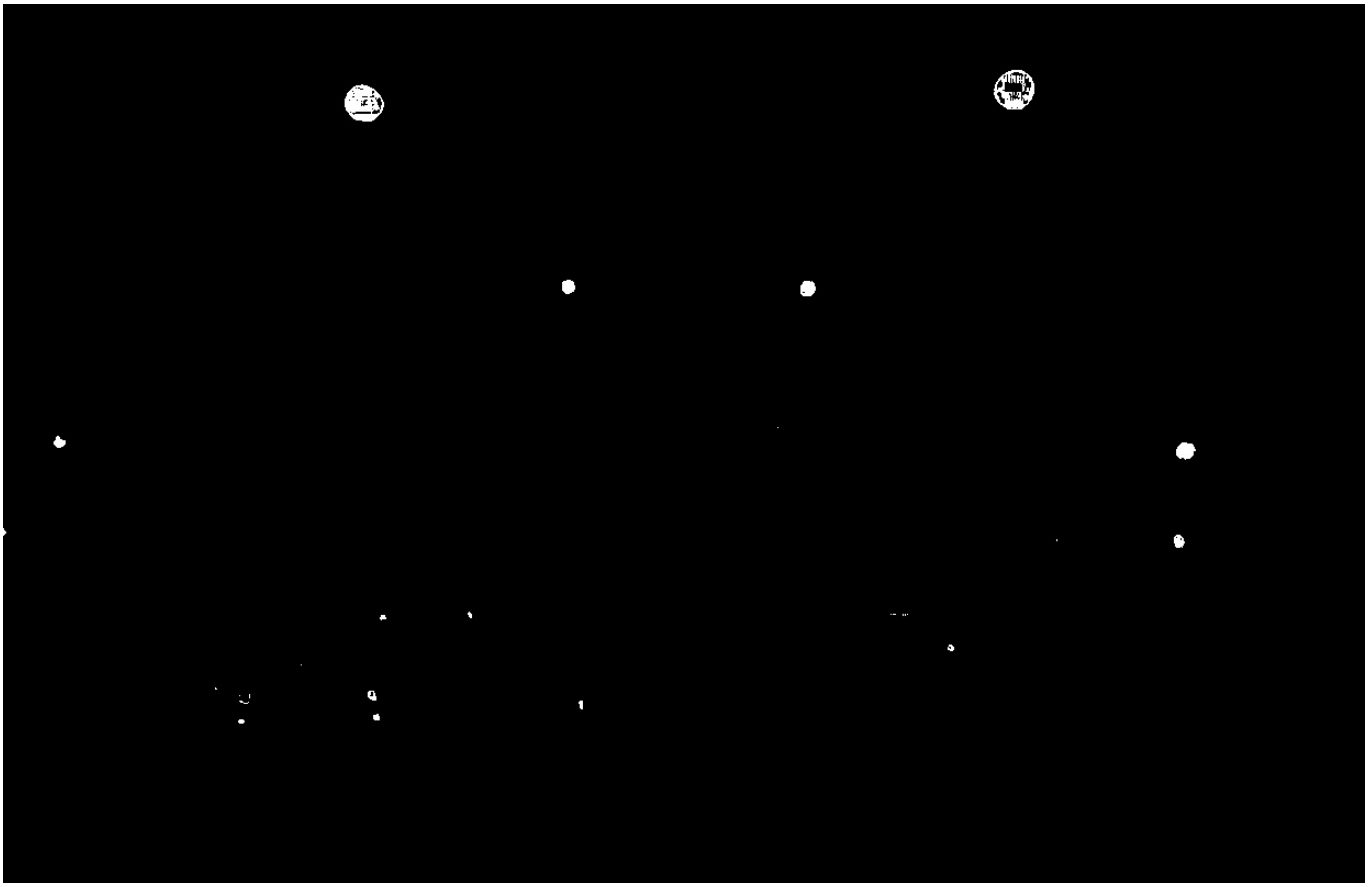


## Use case 2 - Traffic lights detector

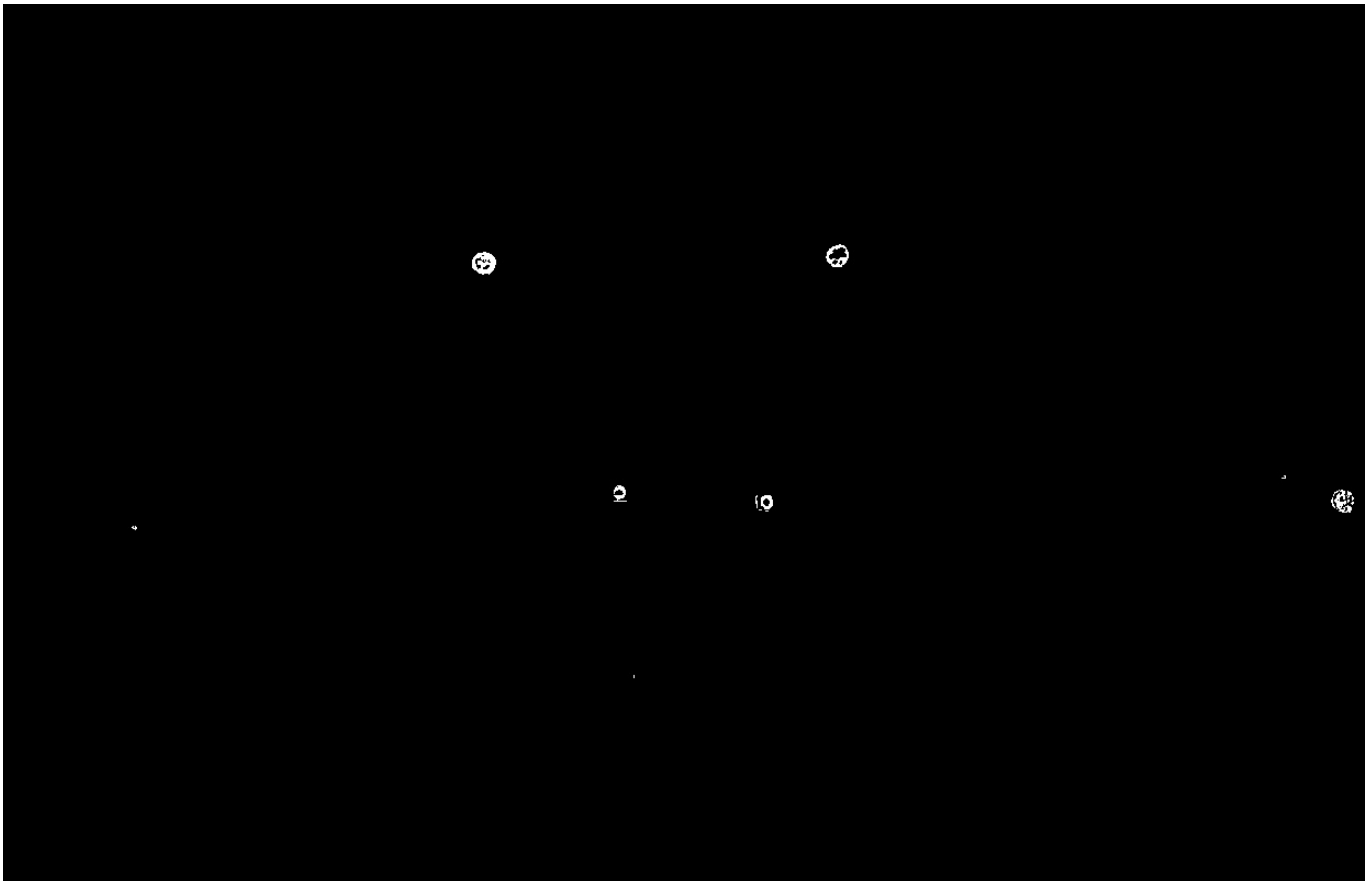
Firstly, I will load original image.


Now I will convert it to HSV. Its because RGB uses 3 channels - Red Green and Blue. HSV use the first channel for hue, second for saturation and third for brightness. In HSV Its easier to detect color in computer vision.

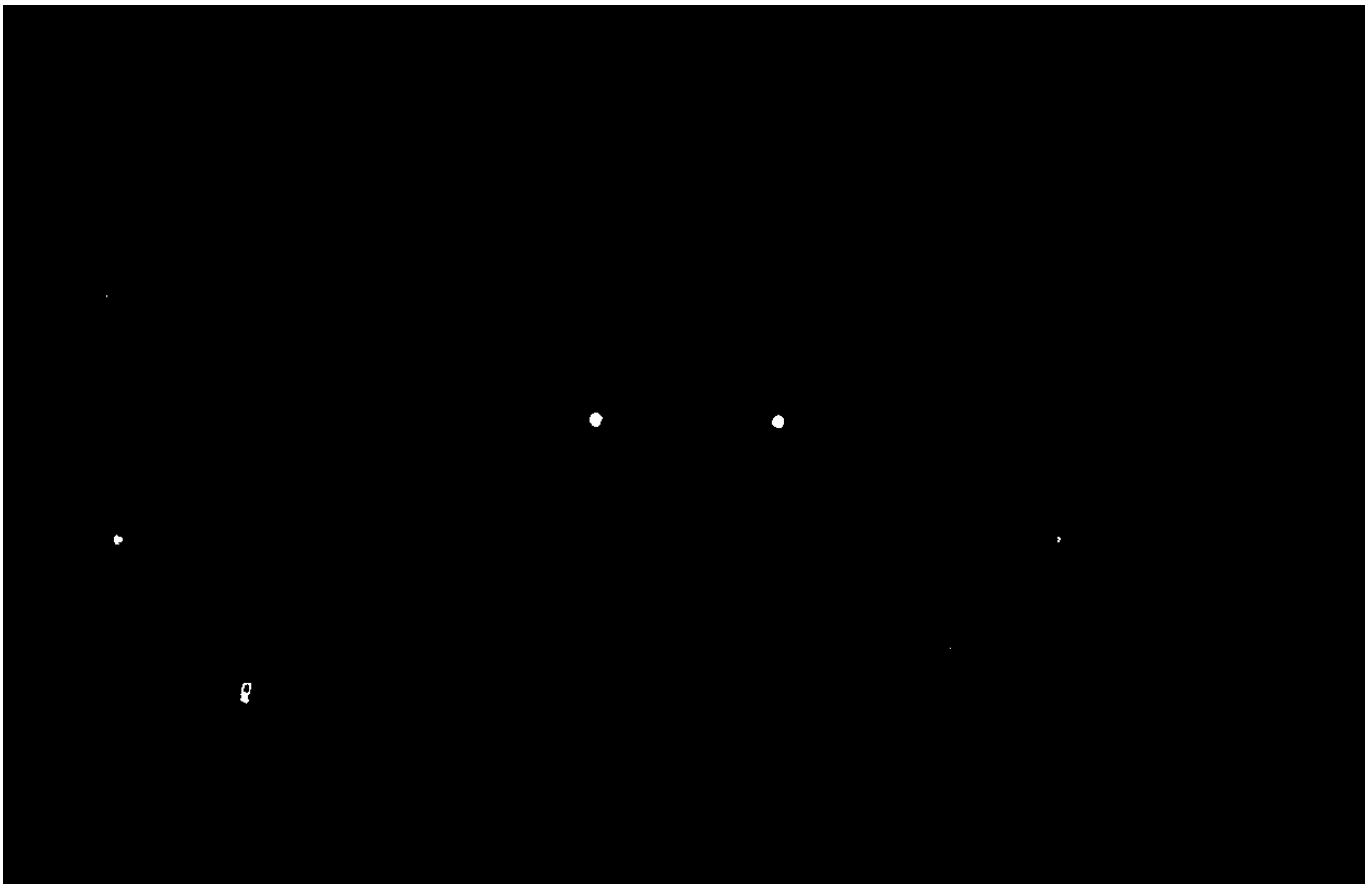
After converting the image to HSV I will use Threshold to create masks of traffic light colors. Thresholding is a process where I will choose border points - min and max. Everything that is under mine and more than max point will be black. Other pixels will be white. Lets threshold to have red mask.



Do it again with different border points for green mask.



And again for orange mask.



Based on masks draw contours in the original picture. Red contours.


Green contours.


And orange contours.


There are very small contours that are not important to me. I will draw only larger contours. Let's start with red.


Green.

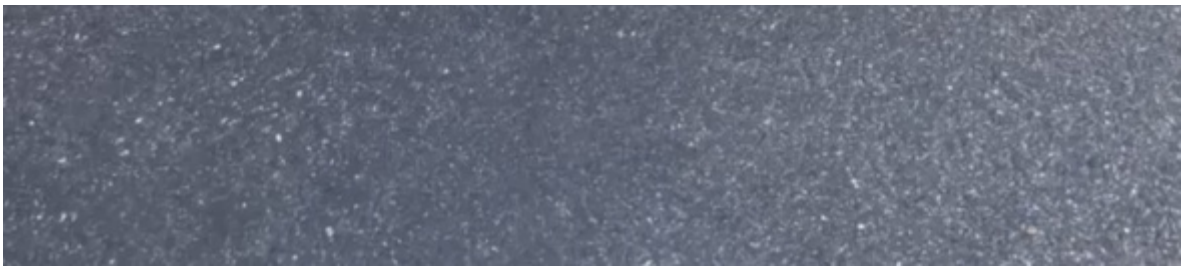

Orange.


Everything seems good. So let's draw contours into the original picture with different colors.

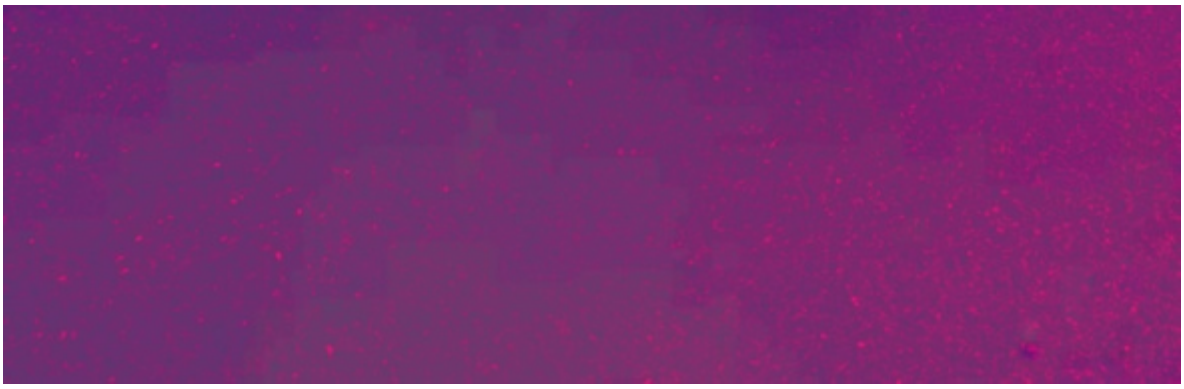

As you can see everything looks perfect. The traffic light detector is ready to use.

## Use case 3 - Licence plate detector - video

The opencv will firstly divide the video into frames. The frame is just a picture - and video is just a bunch of frames in the row. Therefor, I will firstly work on a single image. Lets load one of the frames.

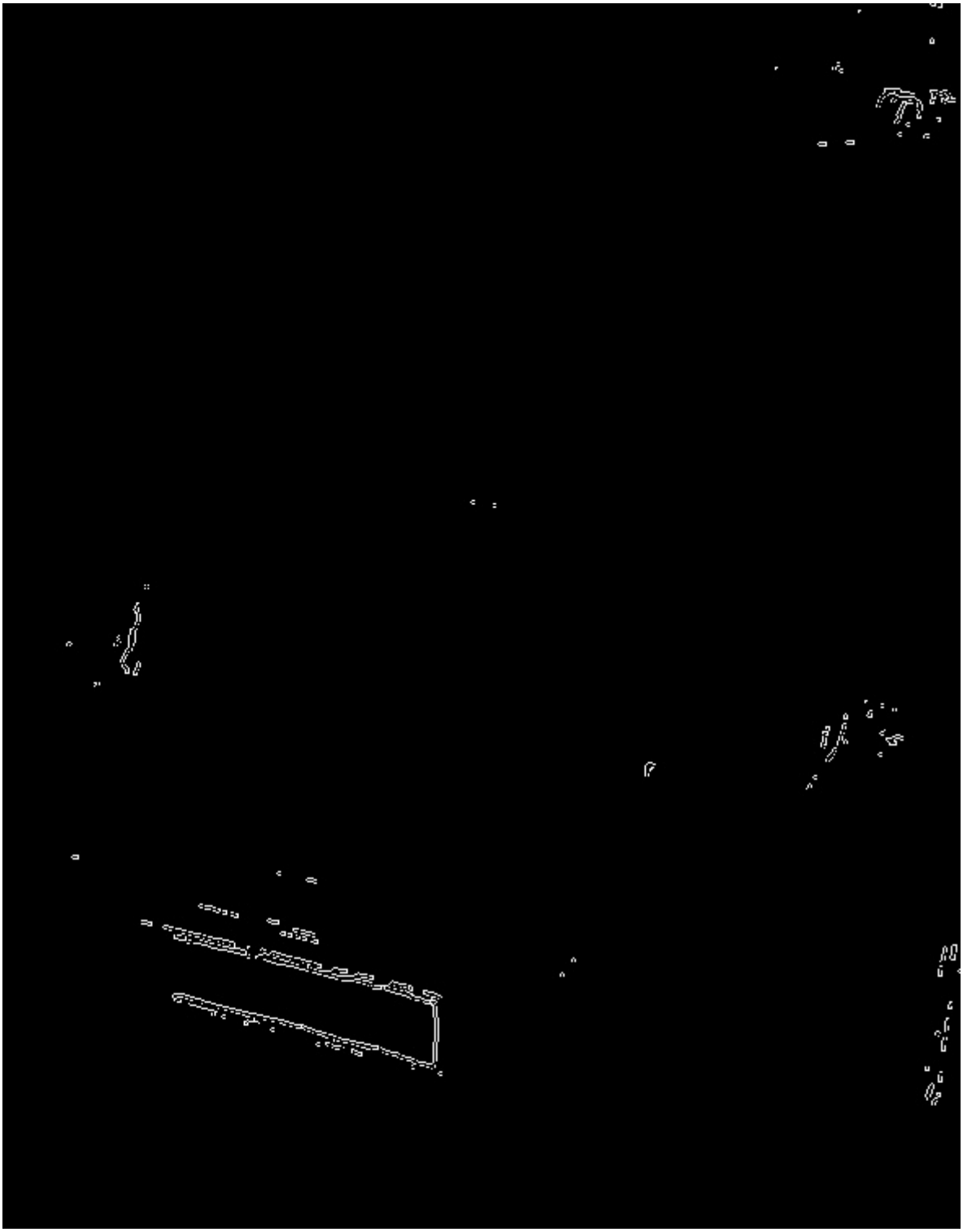


Now I will convert frame into HSV.



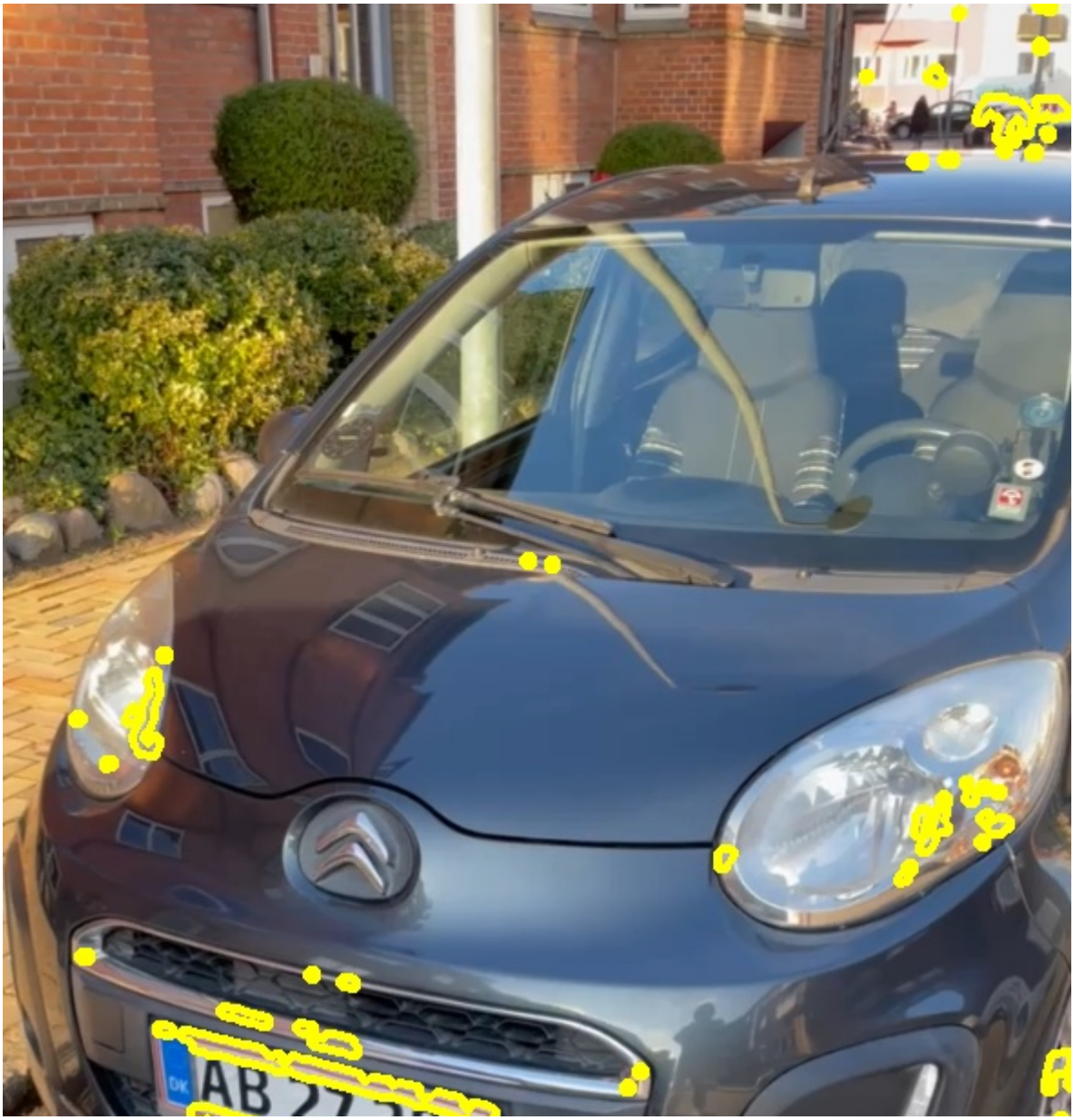
After converting to HSV I will do tresholding to filter out licence plate color.

Now I will use canny edge detection to detect edges of the licence plate.



Edges are there, but they are separated. I need to join them together - to have a complete edge of the plate. For that I will use dilate function.

Now it's looking better. Lets draw contours.



There are small contours that won't be needed. So lets get rid of them. Draw only larger than 1700.


Now its looking a way better. Now Its time to find out the area and draw rectangle around the plate.


So this is at first licence plate. If I run the video - I will get also other licence plates.


This is at 2nd license plate.


And the last.


As you can see, the licence plate detector works just fine. Now the new goal is to find a way how to read numbers and letters from the plate.

# Results and evaluation

## Use case 1

I found out that firstly I need to convert images to grayscale in order to use Cannys edge detection. Before that I used Gaussian blur to filer out noise from the picture with a kernel size of 17 x 17 - the largest number the smoothest picture. Then I used dilate and erode functions with a kernel size of 3 x 3. Finally, after calculating all areas the program can find out different coin values. Now it's really easy to count areas and multiply them with coin value. Write and count all together.

## Use case 2

In case two, Firstly it was needed to do thresholding based on colors. To do that I firstly converted RGB picture to HSV. After thresholding 3 different colors - Red, Green and Orange, I got binary masks. Based on that I was able to draw all contours. I found out that some of them are mistakes - areas that are smaller than 50 in red and green, and larger than 100 in orange case. Finally I will draw only areas between borders and draw them with different colors - to see a difference.

## Use case 3

This use case was the most complicated because its video. In video brightness, colors and saturations are changing every second - depends on the recording device - therefor it was hard to find a thresholding value. At the begging I used just a one frame and test it. I converted frame from RGB to HSV. Then I Found out thresholding points to focus on licence plates - minimal values: (150, 1, 170) and maximal: (180, 60, 235). Based on that I was able to detect edges using a canny edge detector with low point 250 and high point 750. Then I used a delete function with a kernel size of 4 x 4. Later I draw contours but only larger than 1700, and draw red bounding rectangles round them. Area of rectangles should be licence plate. But sometimes program was detecting other similar objects as a plate - for example white walls. So I decided to do following conditions: Rectangle width must be higher than 150 smaller than 215, also height higher than 60and lower than 160. This was still not enough because the program was sometimes detecting only half of the plates, so I was counting only if cols minus rows of plate were higher than 87 and lower than 110. Now it's working just fine.

# Appendix

## Use case 1

C++ Code:

```
#include <opencv2/imgcodecs.hpp>
#include <opencv2/highgui.hpp>
#include <opencv2/imgproc.hpp>
#include <iostream>

using namespace cv;
using namespace std;

int main() {
    Mat img = imread("newimg.jpeg"); //Read image
    Mat imgGray, imgBlur, edges, imgready, mask; Mat copy; //Define variables
    resize(img, img, Size(307.2,409.6 ), INTER_LINEAR); //resize image to 307x409

    imshow("original image", img); //Show original resized image

    img.copyTo(copy);  //make a copy of img
    int canny = 30; //set canny treshold number

    cvtColor(img, imgGray, COLOR_BGR2GRAY); //convert image to grayscale

    GaussianBlur(imgGray, imgBlur, Size(17, 17), 0); //Gaussian blur

    Canny(imgBlur, edges, canny, (3 * canny)); //Canny edge detection

    Mat kernel = getStructuringElement(MORPH_RECT, Size(3, 3)); // Dilate and
erode
    dilate(edges, edges, kernel);
    erode(edges, edges, kernel);

    vector<vector<Point>> contours; //create vector "contours"
    vector<Vec4i>hierarchy; //create vector hierarchy

    findContours(edges, contours, hierarchy, RETR_EXTERNAL, CHAIN_APPROX_SIMPLE);
//Find contours, external (only the largest in a group), keep it simple (save only
important points)
    cout << "NUMBER OF COINS:  " << contours.size() << endl;  //Print contours
sise = number of coins

    int twenty = 0, ten = 0, half = 0; // set number of coins to zero

    for (int i = 0; i < contours.size(); i++) // Iterate through every coin
    {
        if (contourArea(contours[i]) > 3600 && contourArea(contours[i]) < 4900) //
check areas and decide what coin is what
        {
```

```
            drawContours(img, contours, i, Scalar(255, 0, 0), 3);  //draw contours
            twenty++;     //count +1 twenty coin
        }
        else if (contourArea(contours[i]) > 2900 && contourArea(contours[i]) <
3500)
        {
            drawContours(img, contours, i, Scalar(0, 255, 0), 3); //draw contours
            ten++;     //count + 1 ten coin
        }
        else if (contourArea(contours[i]) > 2300 && contourArea(contours[i]) <
2700)
        {
            drawContours(img, contours, i, Scalar(0, 0, 255), 3); //draw contours
            half++;     // count + 1 0.5 coin
        }

    }

    //Print number of coins
    cout << " 20: " << twenty << endl;
    cout << " 10: " <<  ten << endl;
    cout << " 0.5: " << half << endl;
    //Create string - sum of the coins
    string message = ("The sum:" + to_string(int(twenty * 20 + ten * 10 + half *
0.5)) + "DKK");
    //Put sum of the coins into picture
    putText(img, message, Point(10, 400), FONT_HERSHEY_COMPLEX, 1, Scalar(255,
255, 255), 2);

    imshow("result", img); //Show result picture
    waitKey(0);
    return 0;
}
```

## Use case 2

C++ Code:

```
#include <opencv2/imgcodecs.hpp>
#include <opencv2/highgui.hpp>
#include <opencv2/imgproc.hpp>
#include <iostream>

using namespace cv;
using namespace std;

//Set treshold values
Scalar redLow = Scalar(0, 175, 204);
Scalar redHigh = Scalar(10, 242, 255);
Scalar greenLow = Scalar(85, 110, 140);
Scalar greenHigh = Scalar(95, 244, 357);
Scalar orangeLow = Scalar(12, 137, 206);
Scalar orangeHigh = Scalar(30, 255, 255);


int main() {
    Mat img = imread("light.png");  //Load original image
    imshow("Original image", img);  //Show original image

    Mat copy; Mat copy2; Mat red_con; Mat green_con; Mat orange_con; //Set
variables
    img.copyTo(copy); //Make a copy

    cvtColor(img, img, COLOR_BGR2HSV); //Convert image to HSV

    Mat mask_red; Mat mask_green; Mat mask_orange; Mat circless; //Set variables
to treshold
    inRange(img, redLow, redHigh, mask_red); //Red threshold
    inRange(img, greenLow, greenHigh, mask_green); //Green threshold
    inRange(img, orangeLow, orangeHigh, mask_orange); //Orange threshold

    vector<vector<Point>> contours_red; vector<vector<Point>> contours_green;
vector<vector<Point>> contours_orange; //Set vectors for contours
    findContours(mask_red, contours_red, RETR_EXTERNAL, CHAIN_APPROX_SIMPLE);
//Find red contours
    findContours(mask_green, contours_green, RETR_EXTERNAL, CHAIN_APPROX_SIMPLE);
//Find green contours
    findContours(mask_orange, contours_orange, RETR_EXTERNAL,
CHAIN_APPROX_SIMPLE); //Find orange contours

    //Set vectors and store variables into them
    vector<vector<Point> > contours_red_poly(contours_red.size());
vector<vector<Point> > contours_green_poly(contours_green.size());
vector<vector<Point> > contours_orange_poly(contours_orange.size());
    vector<Point2f>centers_red(contours_red.size());
vector<Point2f>centers_green(contours_green.size());
```

```
vector<Point2f>centers_orange(contours_orange.size());
    vector<float>radius_red(contours_red.size());
vector<float>radius_green(contours_green.size());
vector<float>radius_orange(contours_orange.size());

    for (size_t i = 0; i < contours_red.size(); i++)  //Iterate through red
contours
    {
        approxPolyDP(contours_red[i], contours_red_poly[i], 3, true);
//approximates each contour to polygon
        minEnclosingCircle(contours_red_poly[i], centers_red[i], radius_red[i]);
//Define min and max distance between circles
    }
    for (size_t i = 0; i < contours_red.size(); i++) // Iterate through the red
contours
    {
        drawContours(red_con, contours_red, i, Scalar(0, 255, 255), 3);  //draw
red contours
        if (contourArea(contours_red[i]) > 50) // Check if contour is bigger than
50
        {
            circle(copy, centers_red[i], (int)radius_red[i], Scalar(0, 255, 255),
2); //Draw circles around the colors
        }

    }

    //Very simular proccess as above, but for green
    for (size_t i = 0; i < contours_green.size(); i++)
    {
        approxPolyDP(contours_green[i], contours_green_poly[i], 3, true);
        minEnclosingCircle(contours_green_poly[i], centers_green[i],
radius_green[i]);
    }
    for (size_t i = 0; i < contours_green.size(); i++)
    {
        drawContours(green_con, contours_green, i, Scalar(0, 255, 255), 3);
        if (contourArea(contours_green[i]) > 50)
        {
            circle(copy, centers_green[i], (int)radius_green[i], Scalar(0, 0,
255), 2);
        }
    }

    //Very simular proccess as above, but for orange
    for (size_t i = 0; i < contours_orange.size(); i++)
    {
        approxPolyDP(contours_orange[i], contours_orange_poly[i], 3, true);
        minEnclosingCircle(contours_orange_poly[i], centers_orange[i],
radius_orange[i]);
    }
    for (size_t i = 0; i < contours_orange.size(); i++)
    {
        drawContours(orange_con, contours_orange, i, Scalar(0, 255, 255), 3);
```

```
        if (contourArea(contours_orange[i]) > 30 &&
contourArea(contours_orange[i]) < 100) // check if pixel is bigger than 30 and
lower than 100
        {
            circle(copy, centers_orange[i], (int)radius_orange[i], Scalar(255, 0,
0), 2);
        }
    }

    imshow("final.jpg", copy); //Show final image
    waitKey(0);
    return 0;
}
```

## Use case 3

Link to video showing code in progress:
Youtube Video

C++ Code:

```
#include "opencv2/opencv.hpp"
#include <iostream>

using namespace std;
using namespace cv;

int main() {
    VideoCapture cap("car4.mp4");  //try to open video
    if (!cap.isOpened()) {    //Check if successful
        cout << "Error opening video stream or file" << endl;
        return -1;
    }

    while (1) {  //While loop

        Mat frame;  //Define frame
        cap >> frame;  //capture frame

        if (frame.empty())  //if frame is empty, start again
            break;

        rotate(frame, frame, ROTATE_180);  //Rotate frame 180 degrees

        Scalar PlateLow = Scalar(150, 1, 170);  //Low threshold points
        Scalar PlateHigh = Scalar(180, 60, 235); //High threshold points
        Mat img = frame;    //Save frame into img
        int cannyn = 250;  //Set canny low point to 250
        Mat out; Mat plates; Mat out2;  //Set variables
        img.copyTo(out);  // Copy img to out
        img.copyTo(out2); // Copy img to out2

        cvtColor(img, img, COLOR_BGR2HSV);  //Convert image to HSV

        inRange(img, PlateLow, PlateHigh, plates); //Do thresholding

        Canny(plates, plates, cannyn, (cannyn * 3)); //Do canny edge detection

        Mat kernel = getStructuringElement(MORPH_RECT, Size(4, 4)); //Dilate
        dilate(plates, plates, kernel);

        vector < vector < Point>> contours; //Set vector for contours
        findContours(plates, contours, RETR_LIST, CHAIN_APPROX_SIMPLE); //Find
contours

        //Set vectors
```

```
        vector<vector<Point> > contours_poly(contours.size());
        vector<Rect> boundRect(contours.size());
        vector<Point2f>centers(contours.size());
        vector<float>radius(contours.size());

        for (size_t i = 0; i < contours.size(); i++) //Iterate through contours
        {
            approxPolyDP(contours[i], contours_poly[i], 3, true); //approximates
each contour to polygon
            boundRect[i] = boundingRect(contours_poly[i]);
            minEnclosingCircle(contours_poly[i], centers[i], radius[i]);  //define
min and max distance values
        }
        Mat drawing = Mat::zeros(plates.size(), CV_8UC3);

        for (size_t i = 0; i < contours.size(); i++) // Iterate through contours
        {
            if (contourArea(contours[i]) > 1700)  //If contour area is larger than
1700
            {
                //If rect width > 150 and < 215,  and height is > 60 and < 160
                if (boundRect[i].width > 150 && boundRect[i].width < 215 &&
boundRect[i].height > 60 && boundRect[i].height < 160)
                {
                    Mat ROI; //Set ROI variable
                    ROI = out2(boundRect[i]);  //save rectangle into ROI
                    if (ROI.cols - ROI.rows > 87 && ROI.cols - ROI.rows < 110) {
//If cols-rows are > 87 and < 110
                        rectangle(out, boundRect[i].tl(), boundRect[i].br(),
Scalar(0, 0, 255), 5);   //Draw rectangle
                        imshow("Licence plate", ROI); //Show licence plate in new
window
                    }

                }
            }
        }
        imshow("Frame", out); //Show every frame

        char c = (char)waitKey(25);   //input of key
        if (c == 27)    //27 is ESC in ASCII
            break; //If ESC is pressed, stop
    }
    cap.release();  //Release capture

    waitKey(0);
    destroyAllWindows();

    return 0;
}
```