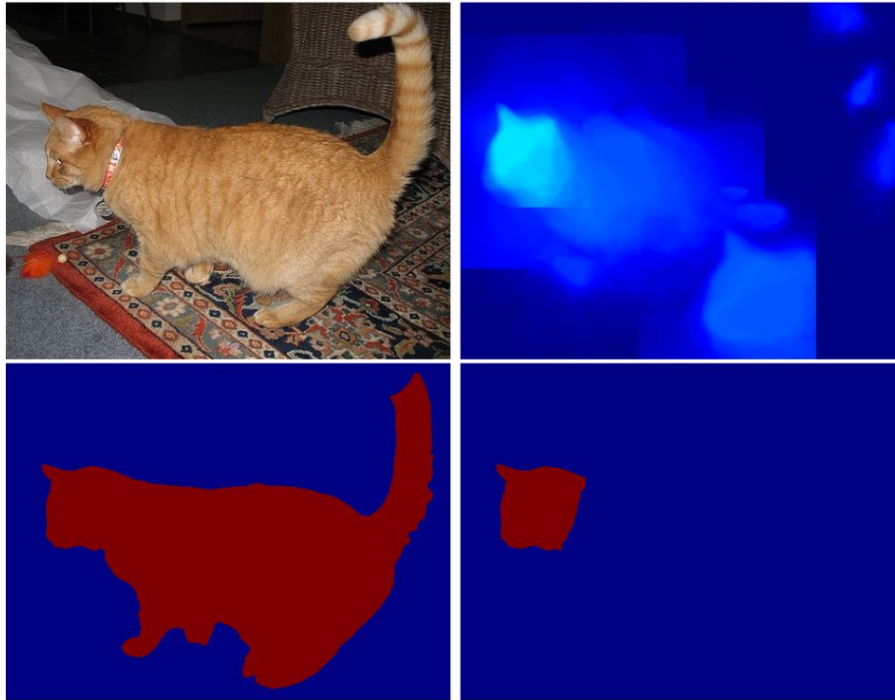


Groeidocument

Vision C++ voor gevorderden



Studenten:

Kenneth van Ewijk (2085204)

Davey Mathijssen (2075116)

Vak:

Vision C++ voor Gevorderden

Klas:

A&B

Datum:

24-11-2017

Fase 1: Feature Detection	2
Opdracht 1.a: Oriëntatie op de feature detection	2
Opdracht 1.b	2
Opdracht 1.c	2
Opdracht 2: Moore Boundary Tracking Algorithm	3
Opdracht 2.1: Afbeeldingen	3
Opdracht 2.2: Code	5
Opdracht 3: Bending Energy	8
Opdracht 3.1: Afbeeldingen	8
Opdracht 3.2: Code	10

Fase 1: Feature Detection

Opdracht 1.a: Oriëntatie op de feature detection

In de theorieles zijn een groot aantal features aan de orde geweest die je kunt meten aan objecten. Features die betrekking hebben op de boundary en features die betrekking hebben op de region. We stellen ons in deze opdracht de volgende onderzoeksvragen:

- “Welke van de in de theorieles besproken features worden door OpenCV ondersteund en wat zijn de namen van de bijbehorende functies?”

We hebben de volgende functies teruggevonden in opencv:

Feature	Functie
Contour representatie	findContours
Contour oppervlakte	contourArea
Contour lengte	arcLength
Contour vorm	convexHull, isContourConvex
Bounding box	boundingRect
Mean	mean

Opdracht 1.b

Welke nieuwe features kom je tegen? Geef ook daarvan de namen van de bijbehorende functies.

We hebben de volgende functies teruggevonden in opencv:

Feature	Functie
Moments	moments
Contour approximation	approxPolyDP
Enclosing circle	minEnclosingCircle
Fitting ellipse	fitEllipse

Opdracht 1.c

Stel je wil de cijfers 1 t/m 9 van rummikub kunnen herkennen. Welke feature set zou je kunnen gebruiken om de cijfers te onderscheiden?

Tip: Maak gebruik van VisionLab. Figuur 1 is digitaal beschikbaar op Bb.

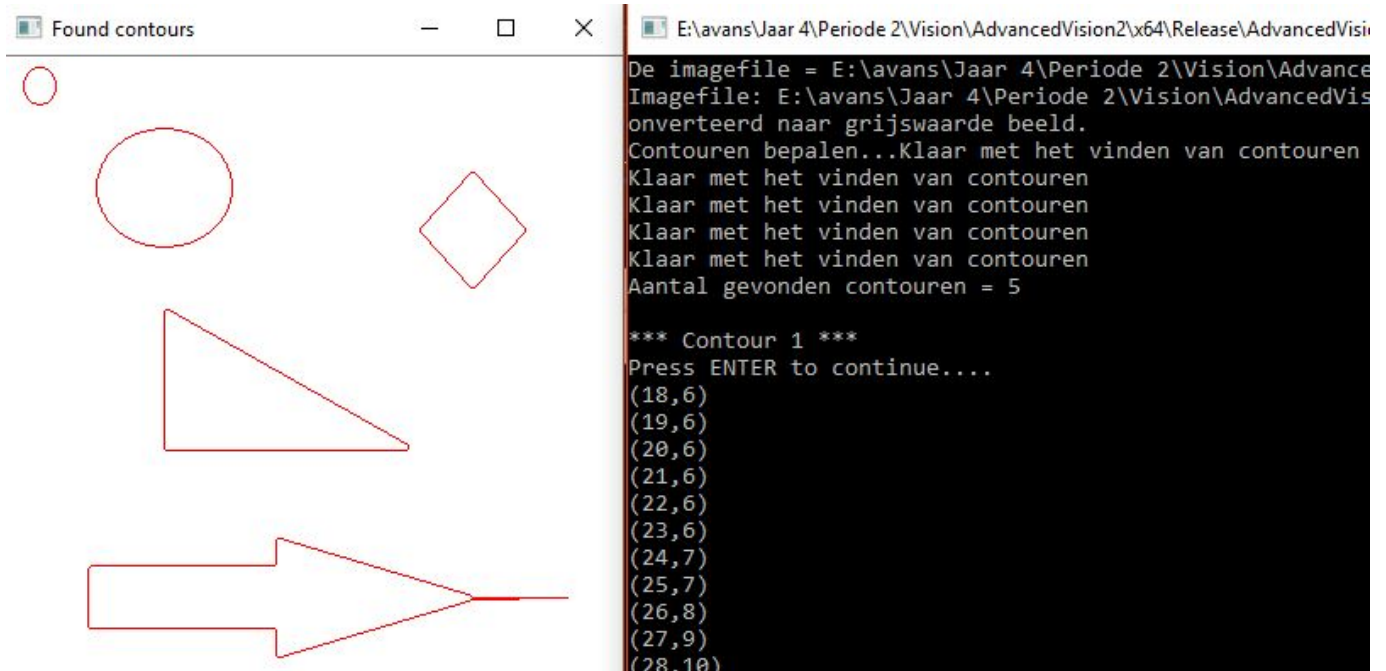
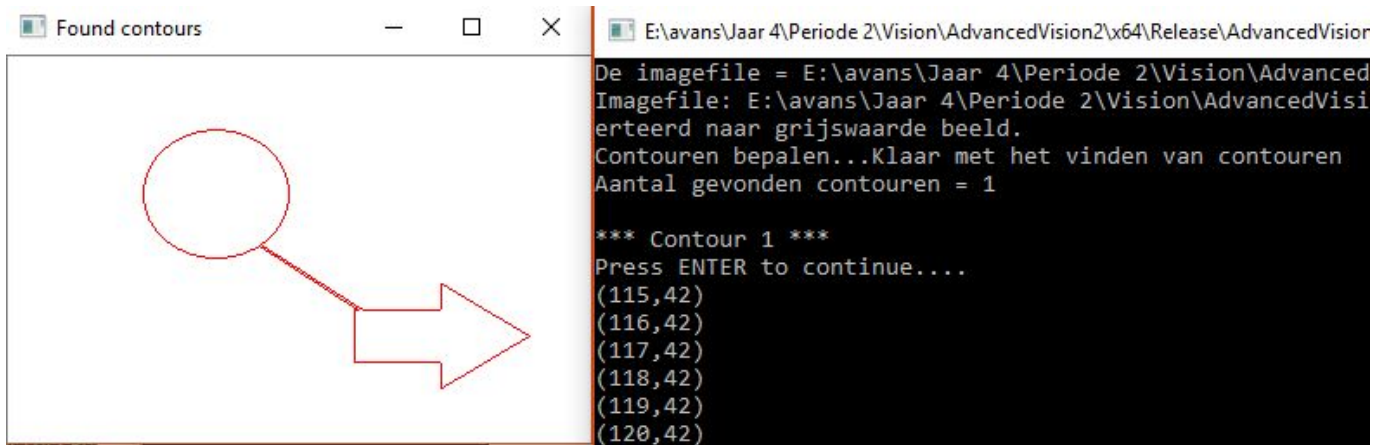
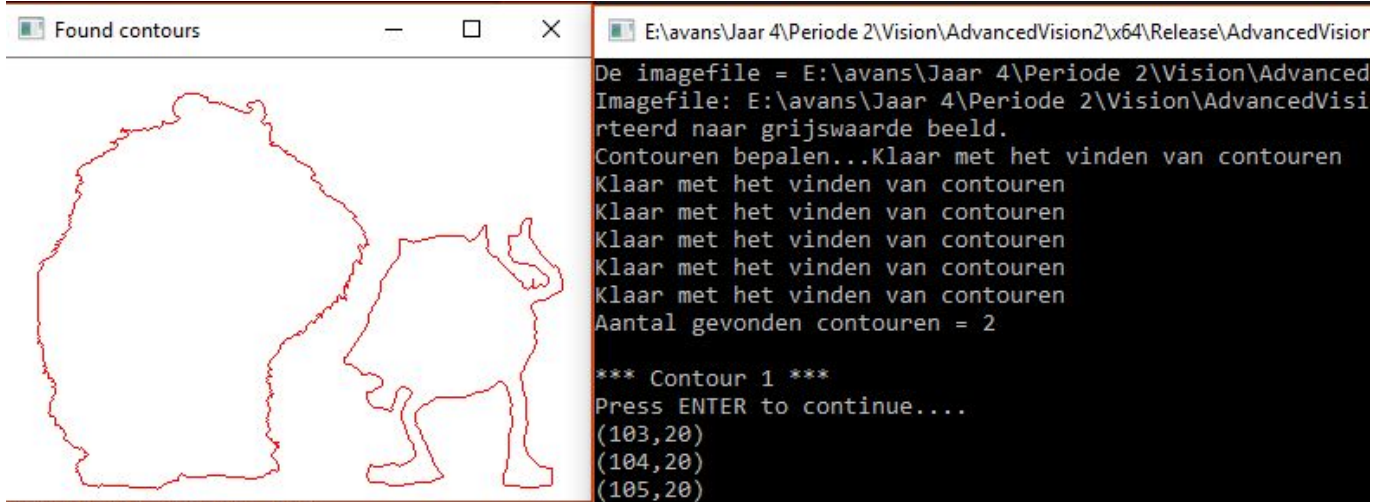
Met de volgende features zou je de cijfers kunnen herkennen:

- Bending Energy
- Circularity
- Nr of Holes
- Area / Omtrek = Compactness

Opdracht 2: Moore Boundary Tracking Algorithm

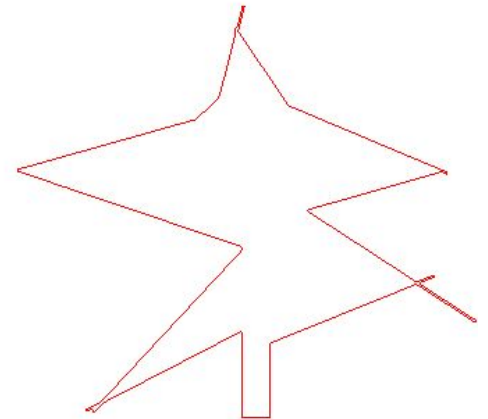
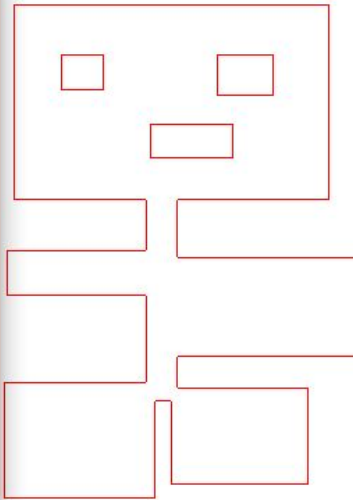
Opdracht 2.1: Afbeeldingen

We hebben onze code getest met de voorbeeldafbeeldingen van BlackBoard en daarnaast zijn er ook nog door onszelf test afbeeldingen gemaakt. Deze zijn hieronder te zien. Bij elke afbeelding is ook weergegeven hoeveel contouren door de code gevonden zijn.

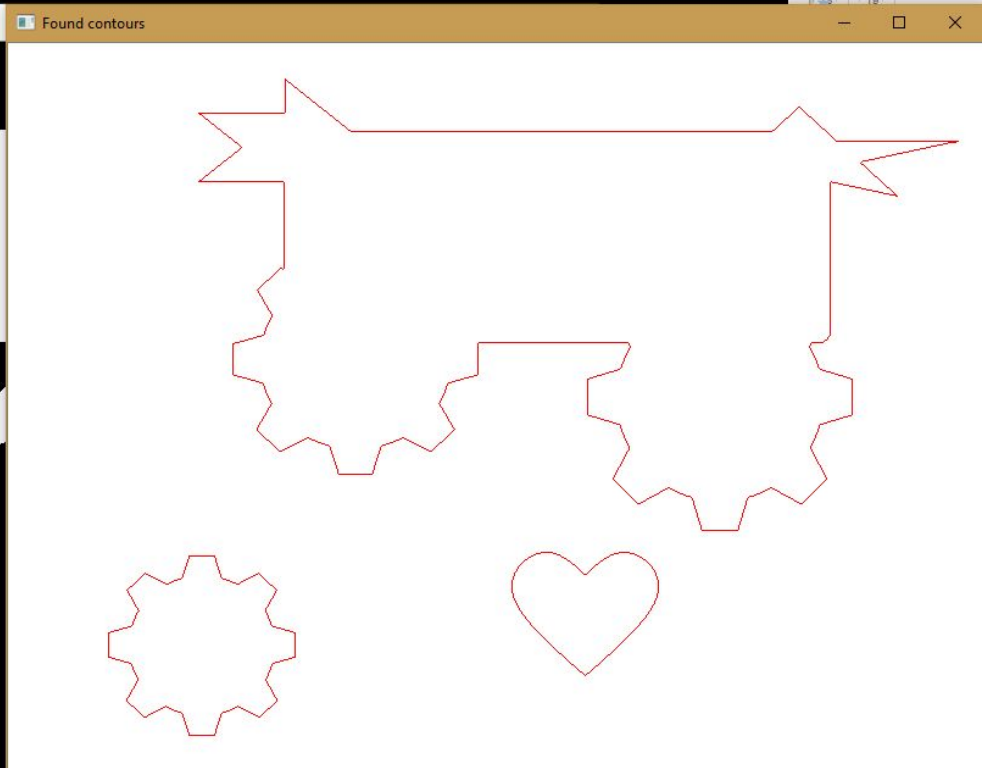
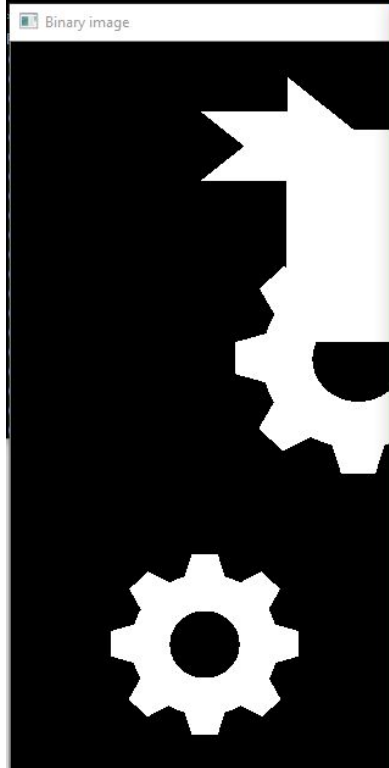


```
E:\avans\Jaar 4\Periode 2\...
De imagefile = E:\avans\Jaar 4\Period
e 2\Vision\AdvancedVision2\AdvancedVi
sion2\input\Week 2\test.jpg
Imagefile: E:\avans\Jaar 4\Periode 2\
Vision\AdvancedVision2\AdvancedVision
2\input\Week 2\test.jpg met succes ge
converteerd naar grijswaarde beeld.
Contouren bepalen...Klaar met het vin
den van contouren
Klaar met het vinden van contouren
Klaar met het vinden van contouren
Klaar met het vinden van contouren
Klaar met het vinden van contouren
Aantal gevonden contouren = 5

*** Contour 1 ***
Press ENTER to continue...
(142,154)
(143,154)
(144,154)
(145,154)
(146,154)
(147,154)
(148,154)
(149,154)
(150,154)
(151,154)
(152,154)
(153,154)
(154,154)
```



```
D:\Code\OpenCV\AdvancedVision2\AdvancedVision\x64\Release\AdvancedVision.exe
De imagefile = D:\Code\OpenCV\AdvancedVision2\AdvancedVision\input\besteplaatjeooit.png
Imagefile: D:\Code\OpenCV\AdvancedVision2\AdvancedVision\input\besteplaatjeooit.png met succes geconverteerd naar grijswa
arde beeld.
Contouren bepalen...Klaar met het vinden van contouren
Klaar met het vinden van contouren
Klaar met het vinden van contouren
Aantal gevonden contouren = 3
```



Opdracht 2.2: Code

Onze code is opgesplitst in verschillende functies. Per functie hebben we toelichting toegevoegd.

AllContours(Mat binaryImage, vector<vector<Point>>& contourVecVec)

De twee arrays (rotateX en rotateY) helemaal bovenaan worden gebruikt om de relatieve pixels te vinden bij het draaien om een andere pixel. De volgorde komt overeen met:

7	0	1
6	X	2
5	4	3

Om de eerste pixels van de BLOBS te vinden gebruiken we de labelBLOBsInfo functie uit de Avans lib. Hieruit gebruiken we de *firstPixelVec2* om de juiste startpositie te krijgen voor ons algoritme. Vervolgens kijken we voor elk van die punten of we het contour kunnen bepalen.

Onze eerste stap is om b0, c0 en firstb0 op te slaan. Deze zijn afgeleid van de verkregen startposities. Hierna gebruiken we de andere functies die later toegelicht zullen worden om de (relatieve-) richting en positie van de eerst volgende randpixel te vinden. Deze slaan we vervolgens op. Hierna wordt dit herhaald in een while loop totdat we weer terug op de beginpositie zijn.

De contourpunten worden alleen opgeslagen als er per contour meer dan 4 punten zijn (om ruis weg te filteren). Daarnaast wordt als laatste het aantal gevonden contouren gereturned na afloop van de functie en de contourpunten via de contourVecVec parameter/variabele meegegeven.

```
volatile int rotateX[] = { 0, 1, 1, 1, 0, -1, -1, -1 };
volatile int rotateY[] = { -1, -1, 0, 1, 1, 1, 0, -1 };

int allContours(Mat binaryImage, vector<vector<Point>>& contourVecVec)
{
    Mat labeledImage;
    vector<Point2d *> firstPixelVec2;
    vector<Point2d *> posVec2;
    vector<int> areaVec2;
    int numberOfBlobs = labelBLOBsInfo(binaryImage, labeledImage, firstPixelVec2, posVec2, areaVec2);

    Point b0, b1, c0, c1, firstb0, firstb1;

    for (Point2d *ptr : firstPixelVec2) {
        vector<Point> contourVec;

        firstb0 = Point((int)ptr->y, (int)ptr->x);
        b0 = Point((int)ptr->y, (int)ptr->x);
        c0 = Point((int)(ptr->y - 1), (int)ptr->x);

        int firstDirection = discoverNext(binaryImage, b0, 6);
        b1 = getDirPos(b0, firstDirection);
        firstb1 = getDirPos(b0, firstDirection);
        c1 = getDirPos(b0, (firstDirection + 7) % 8);

        contourVec.push_back(b0);
        contourVec.push_back(b1);

        do {

            b0 = Point(b1.x, b1.y);
            c0 = Point(c1.x, c1.y);

            int dir = discoverNext(binaryImage, b0,
                                  discoverNextRelativeDirection(b0, c0));
            b1 = getDirPos(b0, dir);
            c1 = getDirPos(b0, (dir + 7) % 8);

            if (!contains(contourVec, b1))
                contourVec.push_back(b1);

        } while (b1 != firstb1);

        if (contourVec.size() > 4)
            contourVecVec.push_back(contourVec);
    }
}
```

```

        } while (b0 != firstb0 && b1 != firstb1);

        cout << "Klaar met het vinden van contouren" << endl;

        if(contourVec.size() > 4)
            contourVecVec.push_back(contourVec);
    }

    return numberOfBlobs; //number of objects
}

```

discoverNext(Mat &binaryImage, const cv::Point &pos, int beginDirection)

Deze functie zoekt vanaf een punt (x, y), en met een bepaalde starttrichting, naar de eerstvolgende randpixel door met de klok mee om dit punt te draaien. Hiervoor worden de rotatie-arrays gebruikt die in de vorige functiebeschrijving staan. Vervolgens geven we de richting terug als een pixel gevonden is.

```

int discoverNext(Mat &binaryImage, const cv::Point &pos, int beginDirection)
{
    if (beginDirection < 0 || beginDirection > 8) {
        cout << "Begin direction is invalid : " << beginDirection << endl;
        return -1;
    }
    for (int i = 0; i < 8; i++) {
        int dir = (i + beginDirection) % 8;

        int newX = pos.x + rotateX[dir];
        int newY = pos.y + rotateY[dir];

        if (getEntryImage(binaryImage, newY, newX) == 1) {
            return dir;
        }
    }

    return -1;
}

```

discoverNextRelativeDirection(const cv::Point &pos, const cv::Point &target)

In deze functie zoeken we naar de richting die tussen twee pixels zit. We kijken of de pixel positie van target overeenkomt met een relatieve positie van pos met een bepaalde richting. Als een richting gevonden is, geven we deze terug.

```

int discoverNextRelativeDirection(const cv::Point &pos, const cv::Point &target)
{
    for (int i = 0; i < 8; i++) {
        int dir = i;

        int newX = pos.x + rotateX[dir];
        int newY = pos.y + rotateY[dir];

        if (target.x == newX && target.y == newY)
            return dir;
    }

    return -1;
}

```

getDirPos(const cv::Point &pos, int dir)

Met deze functie kunnen we met een gegeven positie en richting, een nieuwe pixel positie opvragen, zoals b1 en c1.

```
cv::Point getDirPos(const cv::Point &pos, int dir)
{
    int x, y;
    x = pos.x + rotateX[dir];
    y = pos.y + rotateY[dir];

    return cv::Point(x, y);
}
```

contains(vector<Point> &vec, Point p)

Om te controleren of een punt nog niet gevonden is, controleren we met deze functie of hij al in een lijst zit.

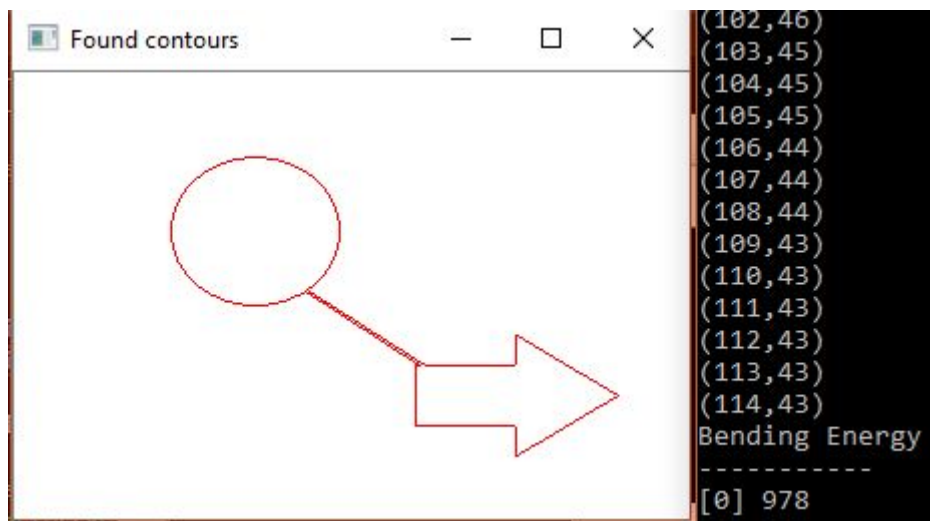
```
bool contains(vector<Point> &vec, Point p)
{
    for (Point pnt : vec)
    {
        if (pnt.x == p.x && pnt.y == p.y)
            return true;
    }

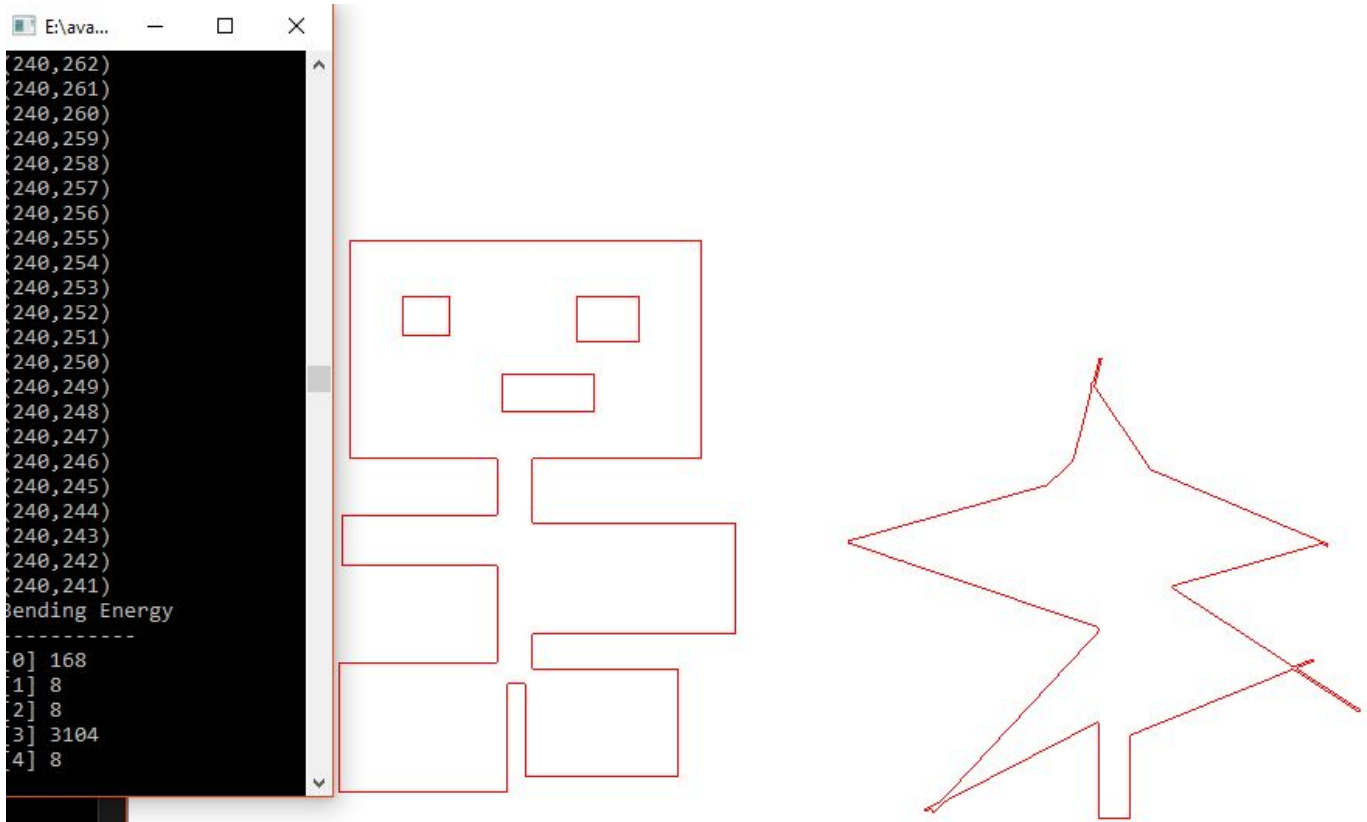
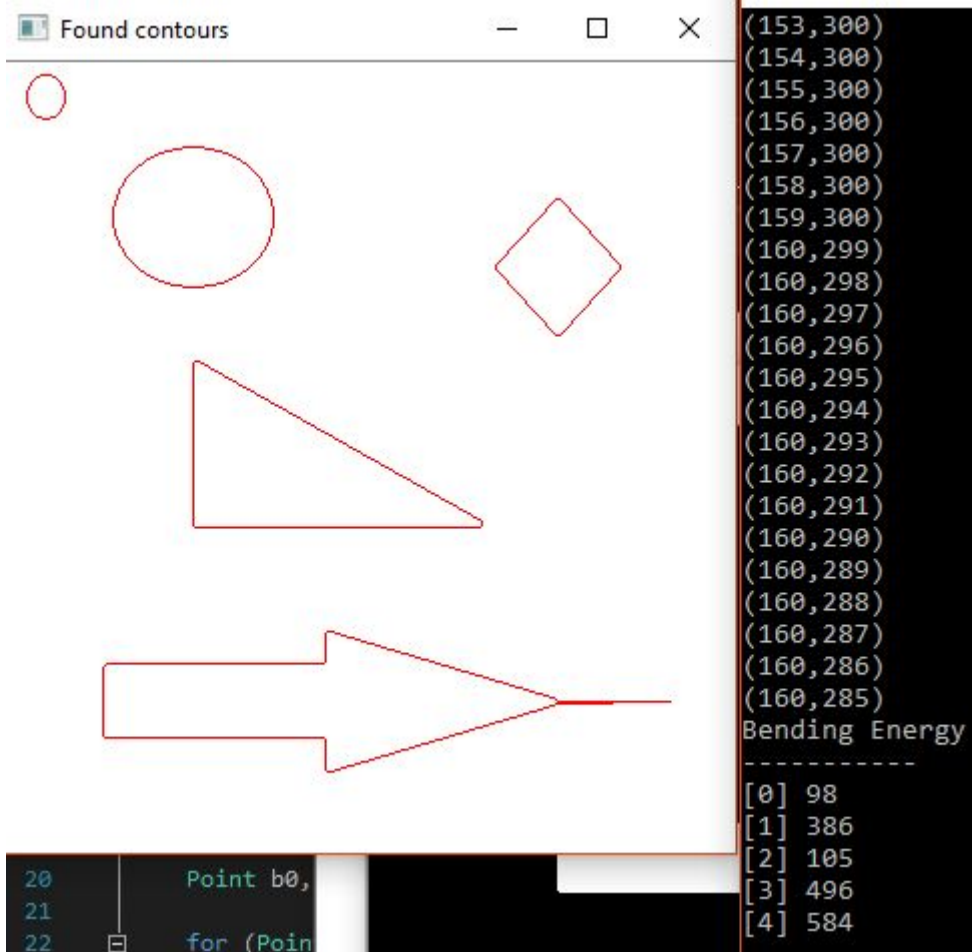
    return false;
}
```

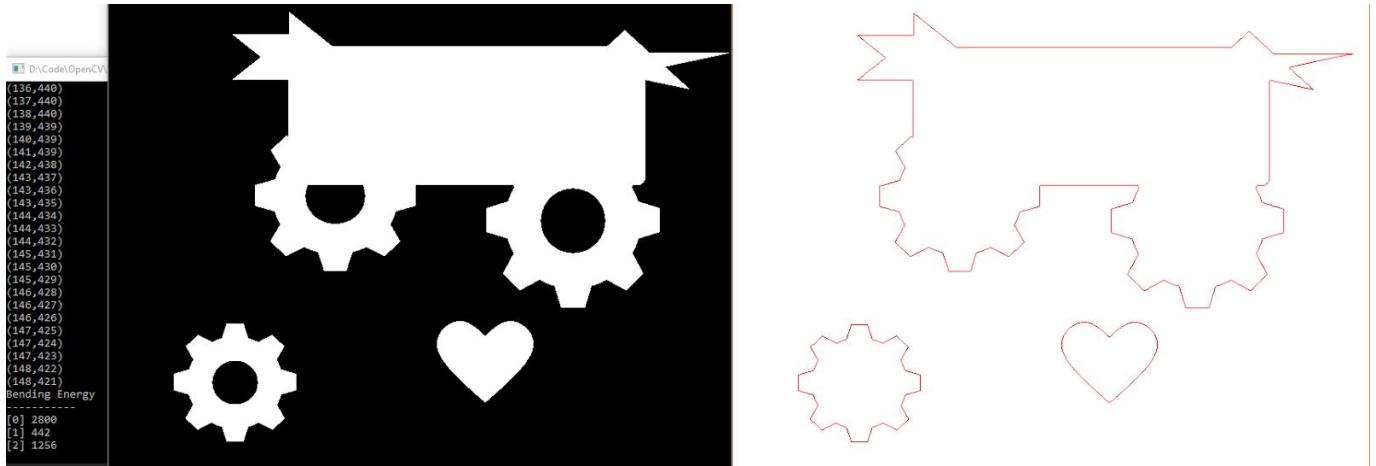
Opdracht 3: Bending Energy

Opdracht 3.1: Afbeeldingen

We hebben onze code getest met de voorbeeldafbeeldingen van BlackBoard en daarnaast zijn er ook nog door onszelf test afbeeldingen gemaakt. Deze zijn hieronder te zien. Bij elke afbeelding is ook weergegeven welke bending energy er gevonden is.







Opdracht 3.2: Code

Om de bending energy te vinden kijken we naar de relatieve richting tussen twee punten binnen het contour. Deze tellen we allemaal bij elkaar op en geven we vervolgens terug.

```
double getBendingEnergy(const vector<Point>& contour)
{
    double energy = 0;
    int dir = 0;
    int prevdir = 0;

    Point previousPoint = contour[contour.size() - 1];

    for (Point p : contour)
    {
        dir = discoverNextRelativeDirection(previousPoint, p);
        energy += (dir - prevdir + 8) % 8;
        prevdir = dir;
        previousPoint = p;
    }

    return energy;
}
```