

# COMPUTER VISION

## Python / OpenCV



## DO-IT\_YOURSELF workshop

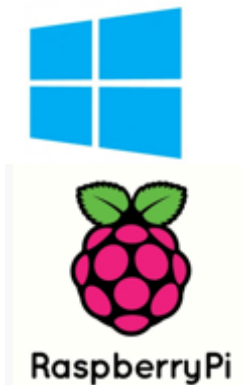
- Hands on – *aan het werk*
- 4 – 8 uren *intensief* programmeren
- Opdrachten afwerken (*stapelen*)
- Geen toets, wel contrôle op deelname
- **BLACKBOARD -> Computer Vision**

## BENODIGDE SOFTWARE

### PYTHON



- Versie 3.7.4 (juli 2019)
- <https://www.python.org/downloads>
- Vink bij installatie aan dat *Python* in het pad komt

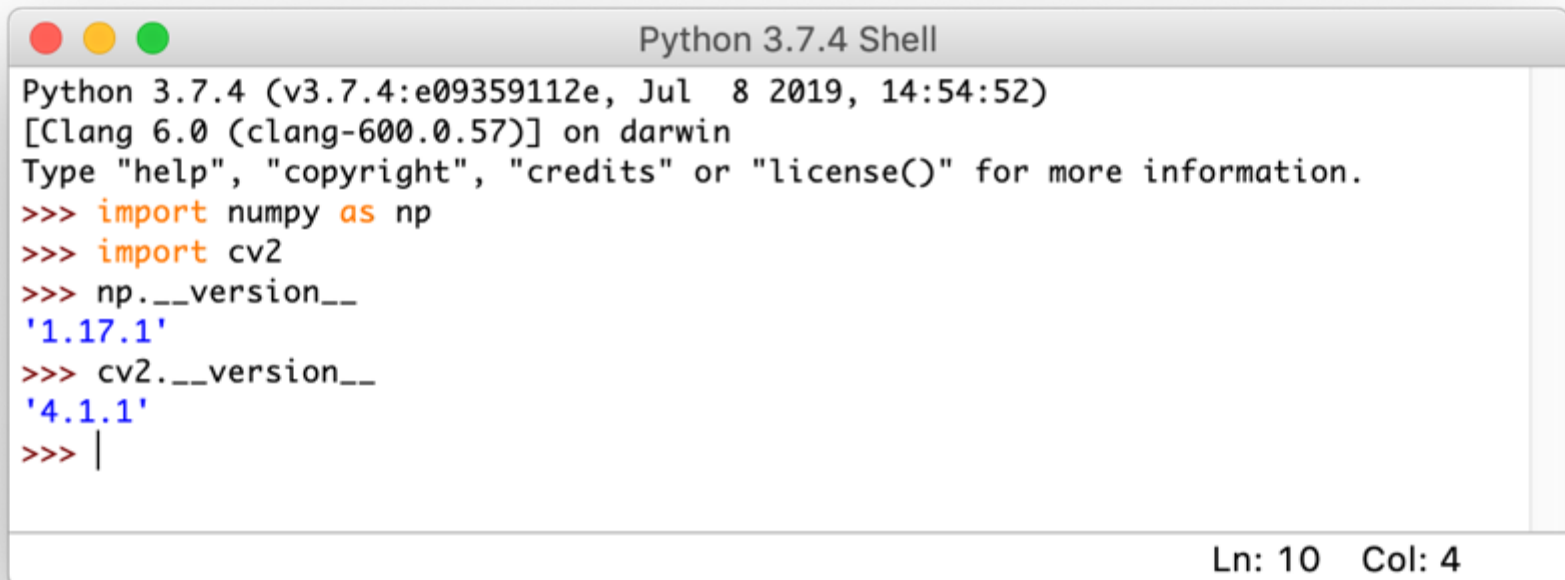


### OPEN CV

- Voor Windows in console (CMD):  
> pip install opencv-python
- Voor raspberry pi vanuit terminal:  
> sudo apt-get install libopencv-dev python-opencv
- Zie appendix voor (kleine) verschillen tussen OpenCV 2.x en OpenCV 3.x

## TEST DE ONTWIKKELOMGEVING

### IDLE (Python GUI)

A screenshot of a Python 3.7.4 Shell window. The window has a title bar with three colored buttons (red, yellow, green) and the text "Python 3.7.4 Shell". The main area contains the following text:

```
Python 3.7.4 (v3.7.4:e09359112e, Jul  8 2019, 14:54:52)
[Clang 6.0 (clang-600.0.57)] on darwin
Type "help", "copyright", "credits" or "license()" for more information.
>>> import numpy as np
>>> import cv2
>>> np.__version__
'1.17.1'
>>> cv2.__version__
'4.1.1'
>>> |
```

The status bar at the bottom right shows "Ln: 10 Col: 4".

## Test de programmeeromgeving

- Download materiaal vanuit de ELO omgeving (in aparte map)
- Open `voorbeeld.py` in IDLE
- Kies vanuit het menu Run-> Run module

Na de prompt `>>>` mag je opdrachten geven

```
>>> sum(x**2 for x in range(10))
```

- *Wat betekent dit?*

## OPDRACHT 1: Logo tonen

```
test1.py - /Users/Bas/Documents/NHL 2019 Q1/OPENCV Workshop/test1.py (3.7.4)
import numpy as np
import cv2
import math

def test1():
    img = cv2.imread('/Users/Bas/Documents/NHL 2019 Q1/OPENCV Workshop/tek1.png')
    cv2.imshow('logo',img)
    k = cv2.waitKey(0)
    print(k)
    cv2.destroyAllWindows()

test1()
```



## OPDRACHT 2: Video capture

● ● ● \*test2.py - /Users/Bas/Documents/test2.py (3.7.4)\*

```
import numpy as np
import cv2
import math

cap = cv2.VideoCapture(0)
while(True):
    ret, frame = cap.read()
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    cv2.imshow('frame', gray)
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break
cap.release()
cv2.destroyAllWindows()
```

Ln: 13 Col: 23

## OPDRACHT 3a: "Knippen in een Plaatje"

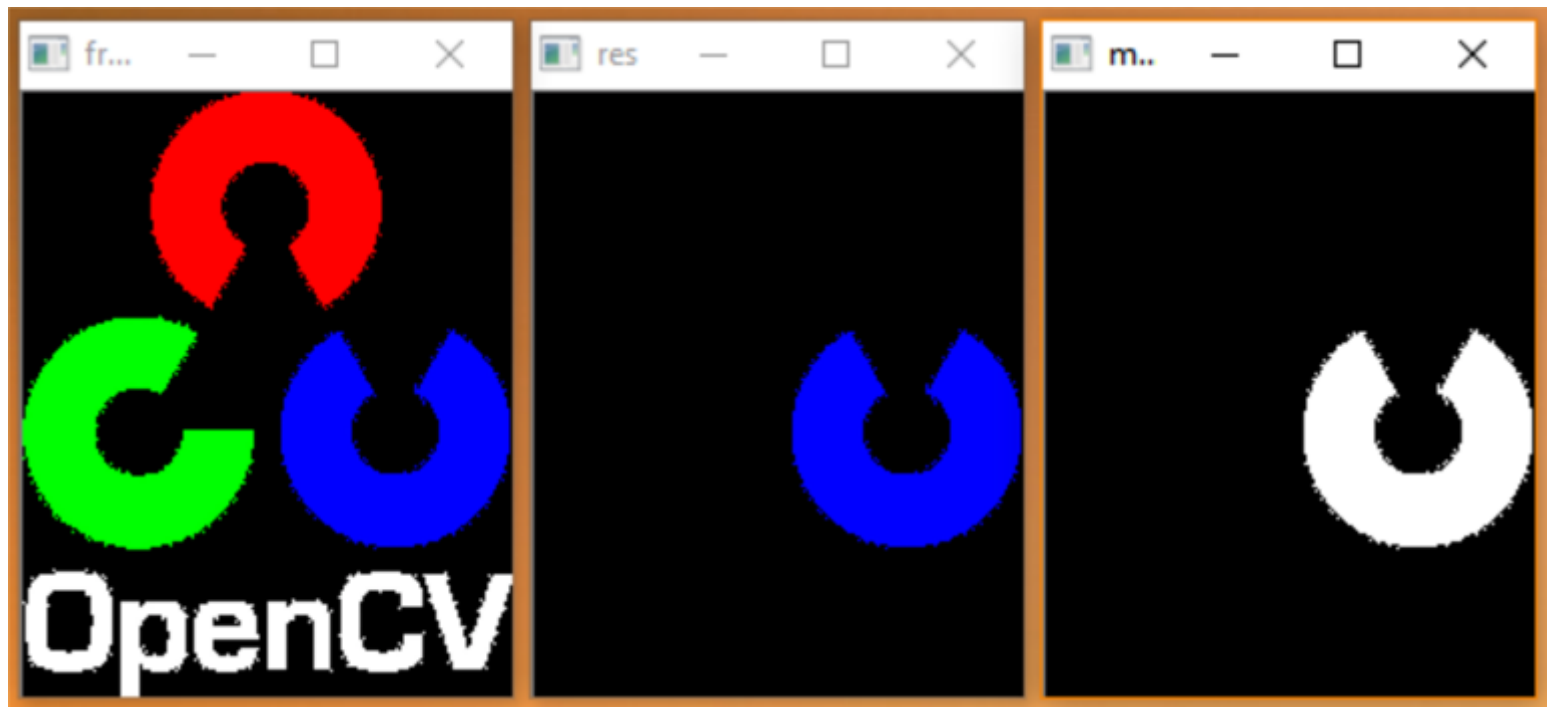
<https://opencv-python-tutroals.readthedocs.io/en/latest>  
<http://docs.opencv.org/3.4.7>

- Zoek in de eerste tutorial naar
  - › *OpenCV-Python Tutorials*
  - › *Image Processing in OpenCV*
  - › *Changing Colorspaces*
  - › *Object Tracking*
- Pas de code zodanig aan dat je het **blauwe** deel van het OpenCV logo (`tek2.png`) los kan laten zien
- Doe dit ook voor het **groene** en het **rode** deel

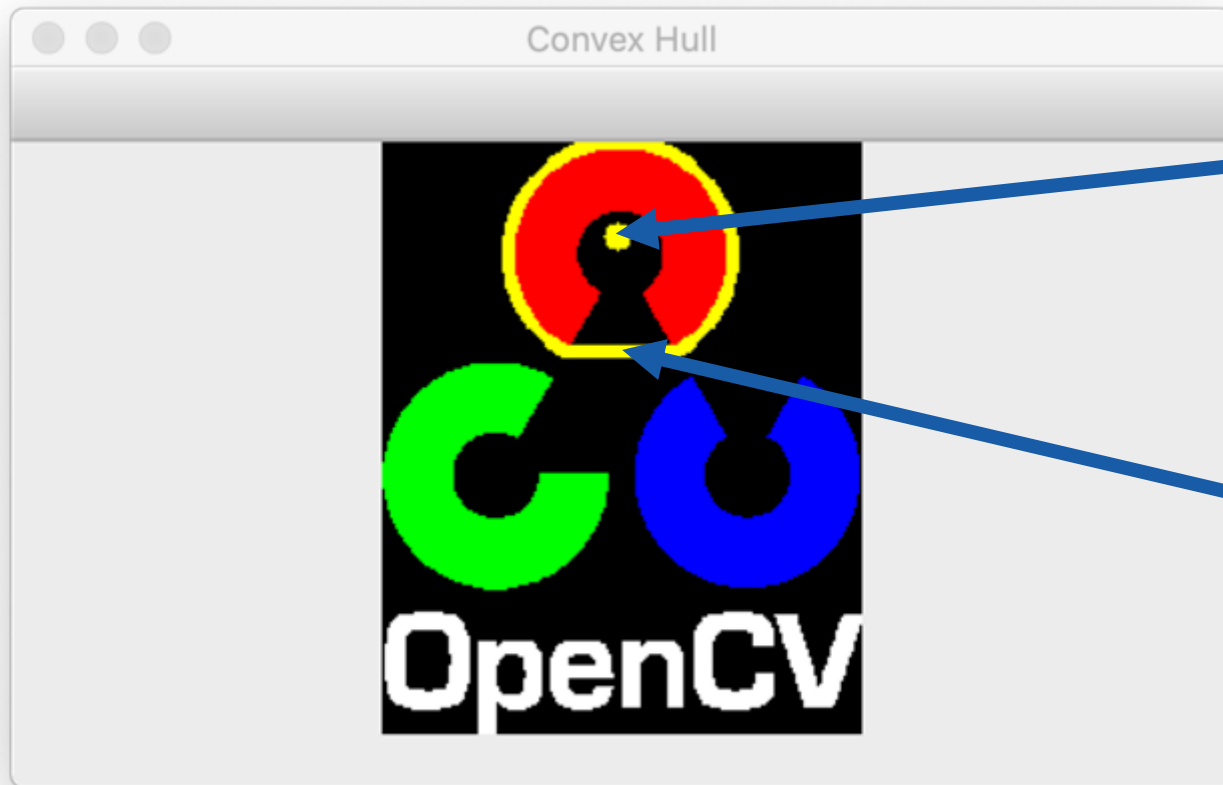


## OPDRACHT 3a: "Knippen in een Plaatje"

### Resultaat



## OPDRACHT 3b: Convex Hull en Centroid



Centroid

Convex Hull

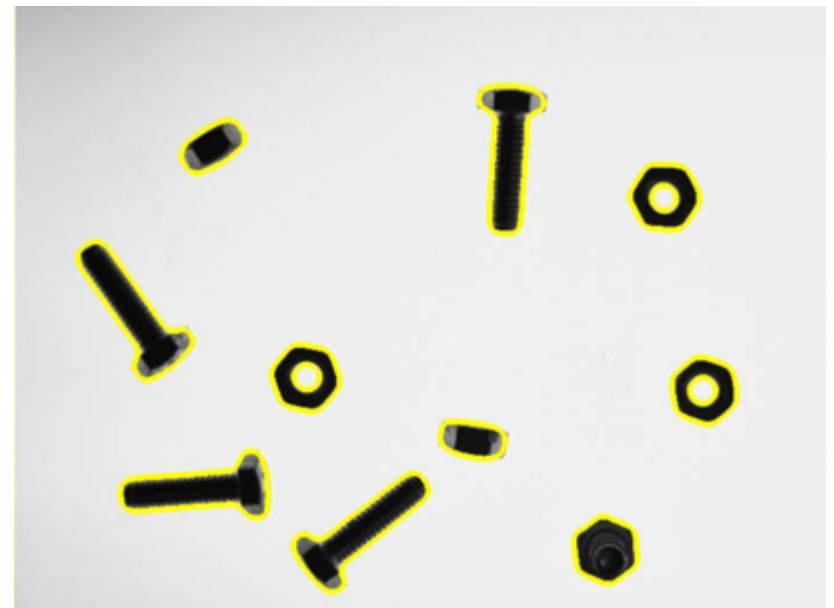
## OPDRACHT 3b: Aanwijzingen Convex Hull en Centroid

### Lees:

- Image Processing in OpenCV
  - › Contours: Getting Started
    - `cv2.findContours()`
    - `cv2.drawContours()`
  - › Contour Features
    - Moments: pas dit toe op `contours[0]`
    - Convex Hull: pas ook dit toe op `contours[0]`
- Gui Features in OpenCV
  - › Drawing Functions in OpenCV
    - Drawing Circle

## OPDRACHT 4a: Bouten en moeren

**Opdracht:**  
**Vind de belangrijkste contouren**

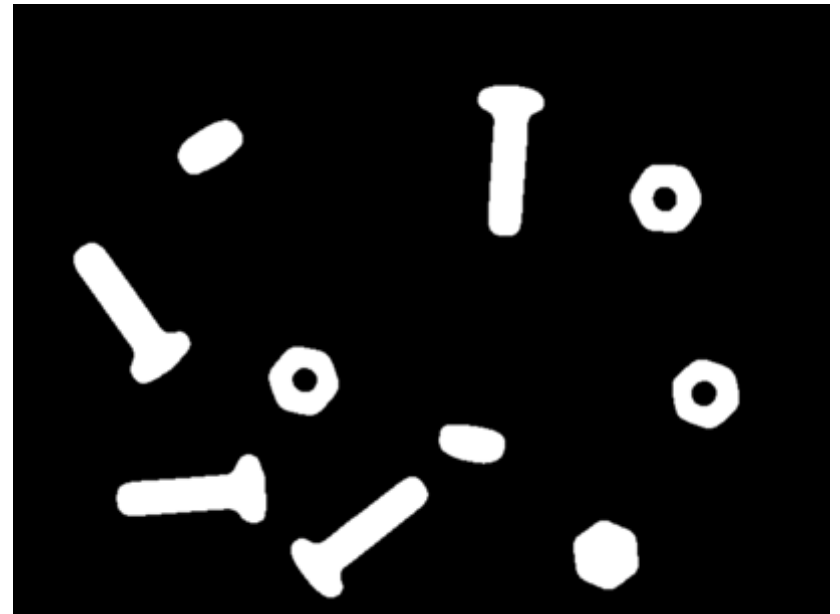
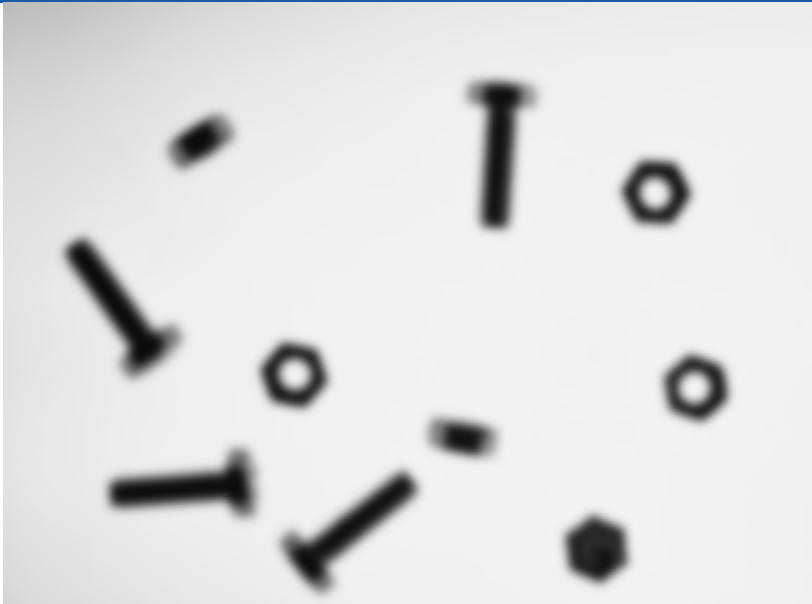


## OPDRACHT 4a: Aanwijzingen bij bouten en moeren



- Converteer naar gray-scale mbv  
`cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)`
- Blur mbv `cv2.GaussianBlur(img, (5, 5), 0)`
- Zie: Image Processing in OpenCV
  - > Smoothing Images
  - > Gaussian Filtering

## OPDRACHT 4a: Aanwijzingen bij bouten en moeren

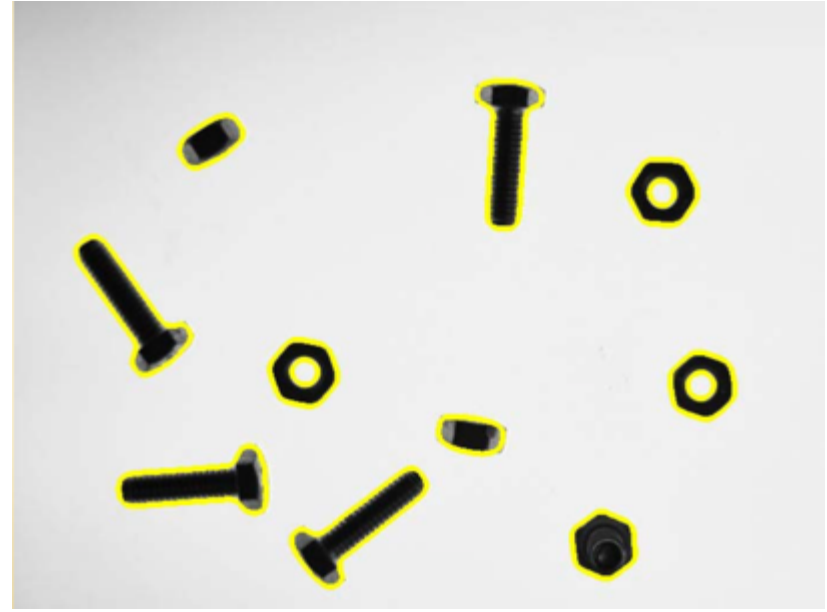
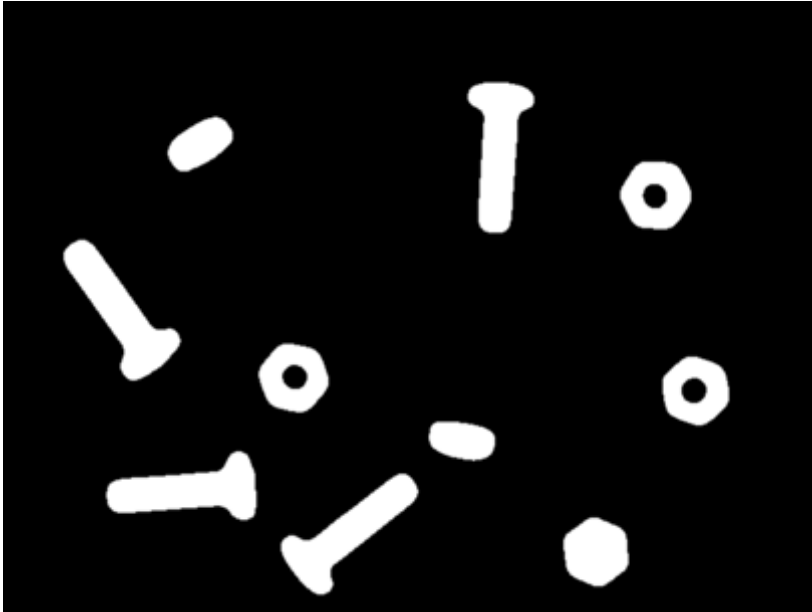


- Vind blobs mbv

```
ret,th = cv2.threshold(img,180,255,cv2.THRESH_BINARY_INV)
```

- Zie: Image Processing in OpenCV -> Image Tresholding

## OPDRACHT 4a: Aanwijzingen bij bouten en moeren



- Vind contouren mbv  
`contours, hierarchy`  
`= cv2.findContours(th, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)`
- Teken ze met `cv2.drawContours()`

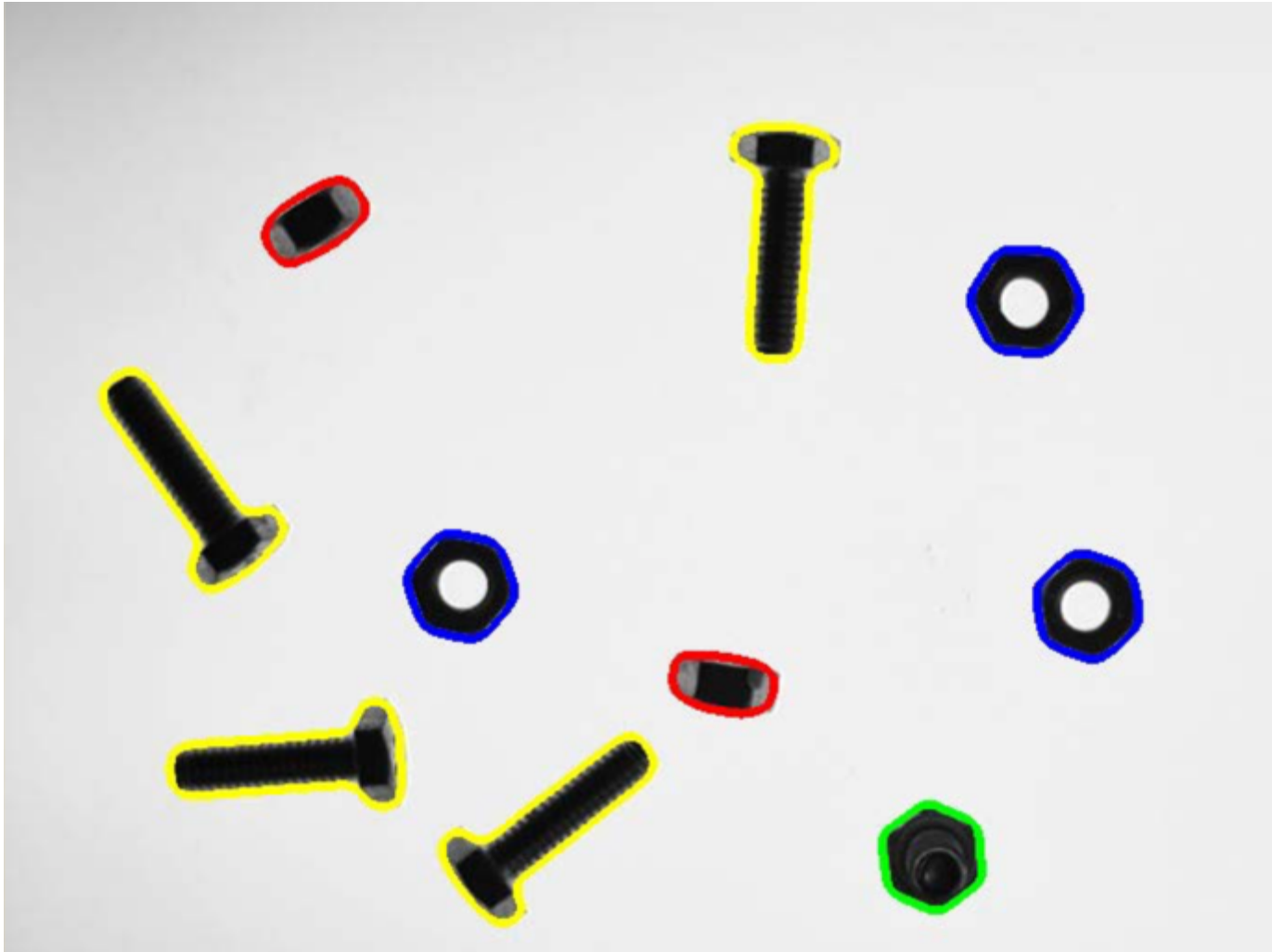
## OPDRACHT 4a: Aanwijzingen bij bouten en moeren

```
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
grayBlur = cv2.GaussianBlur(gray, (25,25), 0)
ret,th=cv2.threshold(grayBlur, 180, 255, cv2.THRESH_BINARY_INV)
contours, hierarchy
    = cv2.findContours(th, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)

for cnt in contours:
    area = cv2.contourArea(cnt)
#   teken alleen contouren met een oppervlakte groter dan 100
    if area > 100:
        cv2.drawContours(img, [cnt], -1, (0,255,255), 3)
```



## OPDRACHT 4b: Dat maakt het verschil tussen bouten en moeren!



## OPDRACHT 4b: Dat maakt het verschil tussen bouten en moeren!

### Aanwijzingen:

- De *vormfactor* van een contour geeft aan hoe rond een contour is
- Een cirkel heeft *vormfactor* 1
- Minder ronde contouren een *vormfactor*  $< 1$
- Hij wordt berekend als  $4 \cdot \pi \cdot \text{opp} / \text{omtrek}^2$
- Hiermee kun je bouten, staande moeren en liggende moeren onderscheiden

## OPDRACHT 4b: Dat maakt het verschil tussen bouten en moeren!

### Aanwijzingen:

- Liggende moeren hebben een groot gat
- Staande bouten niet
- Via de contour hierarchy kun je ze onderscheiden
- Zie: *Contours Hierarchy*
- Dit is een lastig onderdeel!

Below is the result I got, and each  
Next = 1. There is no previous con

```
>>> hierarchy
array([[[[ 1, -1, -1, -1],
         [ 2,  0, -1, -1],
         [ 3,  1, -1, -1],
         [ 4,  2, -1, -1],
         [ 5,  3, -1, -1],
         [ 6,  4, -1, -1],
         [ 7,  5, -1, -1],
         [-1,  6, -1, -1]]],
       [1, 1, 1, 1, 1, 1, 1, 1]])
```

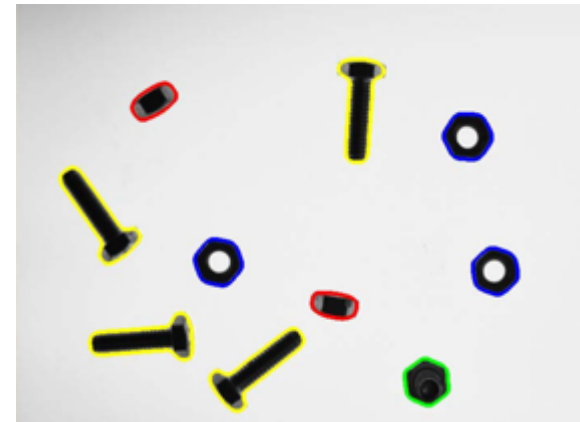
## OPDRACHT 4b: Dat maakt het verschil tussen bouten en moeren!

```
hierarchy = hierarchy[0]
for cnr in range(len(contours)):
    cnt = contours[cnr]
    area = cv2.contourArea(cnt)
    perimeter = cv2.arcLength(cnt, True)
    factor = 4 * math.pi * area / perimeter**2
    holes = 0
    child = hierarchy[cnr][2]
    while child >= 0:
        holes += cv2.contourArea(contours[child])
        child = hierarchy[child][0]
    print area, factor, holes
```

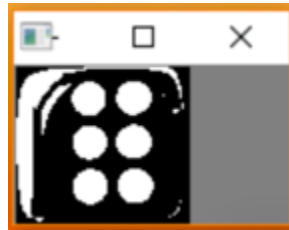
## OPDRACHT 4b: Dat maakt het verschil tussen bouten en moeren!

### Aanwijzingen:

- Maak een tekening van de hierarchy
- Leg voor jezelf uit hoe de berekening van “holes” werkt
- Zorg dat je alle typen van een andere kleur contour voorziet



## OPDRACHT 5: Dobbelstenen



[3, 3, 3, 3, 4, 4, 5, 5, 6, 6, 6, 6]

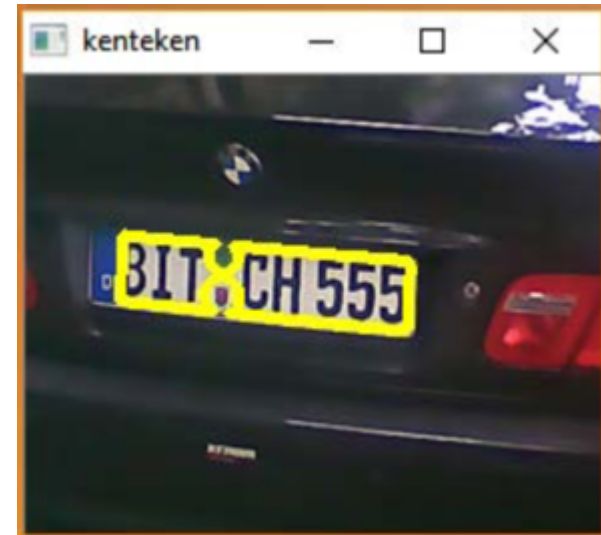
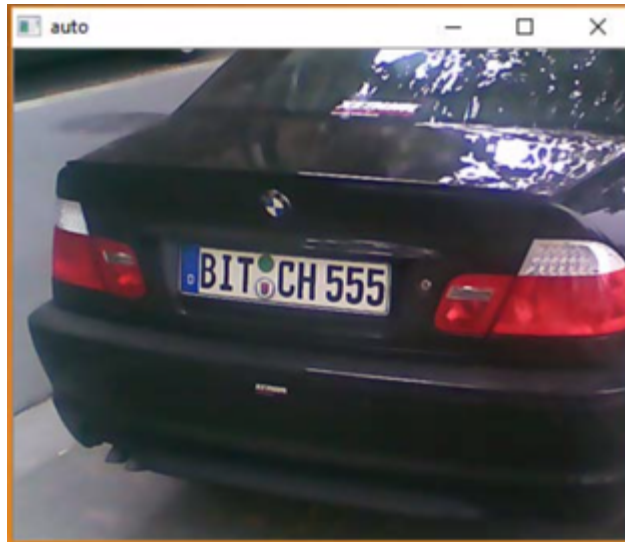
## OPDRACHT 5: Dobbelstenen

### Aanwijzingen:

- Zoek eerst de dobbelstenen via grayscale en threshold
- Doorloop de contouren en bereken *ROI*

```
for cnr in range(len(contours)):  
    x,y,w,h = cv2.boundingRect(contours[cnr])  
    die = gray[y:y+h, x:x+w]
```
- Voor elke “die” ga je thresholden en contouren berekenen. De ronde contouren zijn de “ogen”.
- Denk aan de [vormfactor](#)
- Zie: Core Operations -> Basic Operations

## OPDRACHT 6: Kenteken (optioneel)



### Aanwijzingen:

- Neem een region of interest (ROI)
- Maak het blauwe en rode kanaal 0
- Teken de grootste contour
- Zie: Core Operations -> Basic Operations



# COMPUTER VISION

Bedankt voor de  
aandacht

## Verschillen OpenCV 3.\* / 2.\*

OpenCV 3.\*:

```
image, contours, hierarchy
    = cv2.findContours(th,cv2.RETR_TREE,cv2.CHAIN_APPROX_SIMPLE)
img = cv2.drawContours(img, contours, -1, (0,255,0), 3)
img = cv2.circle(img,(cx,cy), 5, (0,255,255), -1)
```

OpenCV 2.\*, maar ook 3.4.7!:

```
contours, hierarchy
    = cv2.findContours(th,cv2.RETR_TREE,cv2.CHAIN_APPROX_SIMPLE)
cv2.drawContours(img, contours, -1, (0,255,0), 3)
cv2.circle(img,(cx,cy), 5, (0,255,255), -1)
```