

Video bewerking (video processing)

Irob – robotica domotica & industriële automatisering

Inleiding	1
Open CV installeren op de Raspberry Pi.....	1
Het testen van OpenCV	1
Basis manipulaties	2
Background subtraction.....	2
Contour detection	4
Bounding boxes	4
Template matching.....	5

Inleiding

In deze les gaan we leren hoe we video's kunnen bewerken. De bewerking die je op afbeeldingen doet kun je ook doen op video's. Denk aan grijswaarden, blurring, en edge detection. Bij video's gaan we ons focussen op het detecteren van objecten. Object detectie is een veelgebruikte toepassing binnen AI/Machine Learning. Hiervoor zullen we de volgende onderwerpen behandelen: background subtraction, contour detection en template matching. Voor de camera gebruiken we de Raspberry Pi. Het doel is om uiteindelijk object detectie te implementeren op een Raspberry Pi.

Open CV installeren op de Raspberry Pi

Het installeren van OpenCV op de Raspberry Pi is redelijk rechttoe rechtaan. Je kunt er voor kiezen om bij het installeren een environment te maken waarbinnen je OpenCV installeert. Bekijk de volgende links voor installatie.

<https://pypi.org/project/opencv-python/>

<https://raspberrytips.com/troubleshooting-camera-module/>

<https://www.youtube.com/watch?v=QzVYnG-WaM4>

Verder is het voor deze scripts makkelijker om met Idle editor te werken in plaats van Jupyter Notebook.

Het testen van OpenCV

Om de installatie te testen kunnen we een video afspelen met het volgende script op de DLO: [oefening1_playvideo.py](#). Het is belangrijk dat je het bijhorende video bestand ook op je Pi zet. Voor het overzetten van bestanden kun je FileZilla gebruiken. Deze regel laadt de video in:

```
cap = cv2.VideoCapture('data/auto_oscar.mp4')
```

Als je nu de camera wilt gebruiken hoef je alleen de bestandsnaam te vervangen voor een 0. 0 betekent de eerste camera die OpenCV kan vinden. Heb je meerdere camera's dan kun je ook een 1 opgeven voor de volgende camera.

```
cap = cv2.VideoCapture(0)
```

Basis manipulaties

Nu OpenCV en de camera werkt kun je basis operaties uitvoeren op de video's. Een video is eigenlijk niets anders dan een sequentie van afbeeldingen over tijd. Je kunt dezelfde operaties uitvoeren als bij afbeeldingen. Probeer de camera output maar eens een blur te geven met de volgende code:

```
kernel = np.ones((5,5),np.float32)/25
frame = cv2.filter2D(frame,-1,kernel)
```

Ook kunnen we de camera beelden zwart-wit (grijswaarden) maken door het volgende aan je code toe te voegen:

```
frame = cv2.cvtColor(frame, cv.COLOR_RGB2GRAY)
```

Met de volgende code kun je Canny edge detection toepassen. Canny is een meer geavanceerde variant van **Sobel** die we eerder gezien hebben. Zoek op Internet op hoe deze werkt.

```
frame = cv2.Canny(frame, 50, 150)
```

Background subtraction

Een van de meest gebruikte toepassingen in beeldverwerking is object detectie. Stel je hebt een camera boven een snelweg en je wilt het aantal voertuigen detecteren en misschien ook deze voertuigen volgen (en tellen). Een voorbeeld hiervan is te zien in onderstaande afbeelding.



Een methode is om de achtergrond van de voorgrond af te trekken. Maar wat bedoelen we hiermee? Laten we eerst een voorbeeld nemen van twee afbeeldingen.



Je ziet een tafel zonder bal en een tafel met een bal. Je zou de twee afbeeldingen van elkaar kunnen aftrekken. Waar de afbeeldingen gelijk zijn is het verschil 0 (of ongeveer 0). Waar de bal is zie je wel een verschil. De gehele analyse kun je vinden in het volgende script (run dit script op je PC met Jupyter notebook): [achtergrond verwijderen kleur.ipynb](#)

De achtergrond verwijderen in een video gebeurt anders. Hiervoor gebruiken we de functie.

```
cv2.createBackgroundSubtractorMOG2()
```

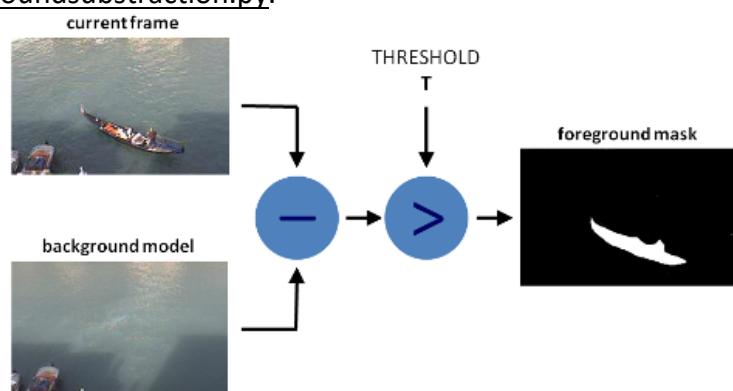
MOG staat voor Mixture of Gaussians. Wat deze functie doet is een pixel waarde meten over tijd. Hierdoor krijg je een gemiddelde of een distributie van een pixel waarde. Er is veel informatie te vinden over background substraction. Bekijk bijvoorbeeld de volgende video:

<https://www.youtube.com/watch?v=fn07iwCrvqQ>

Background subtraction klinkt simpel maar het heeft haken en ogen. Ruis kan komen door een aantal zaken:

- De camera staat niet perfect stil
- Interne ruis van de camera
- Belichting is anders (bijvoorbeeld de zon komt tevoorschijn achter de wolken)

Dit noemen we de achtergrond (gemeten over tijd). Alleen een fractie van de tijd zal een pixel een andere waarde hebben als er een object voorbij komt [1]. Run het volgende script: [oefening2_backgroundsubtraction.py](#).



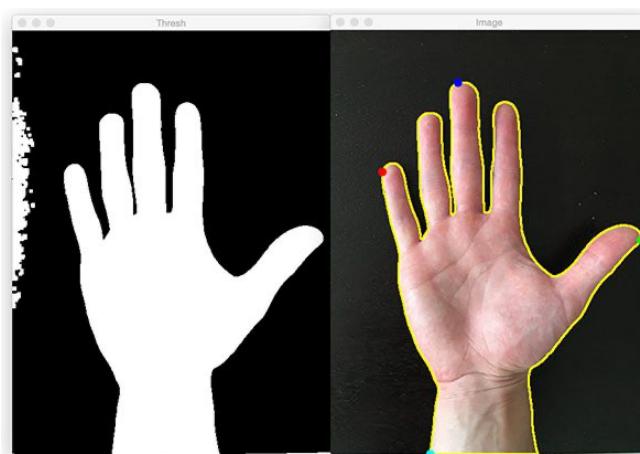
Je zult merken dat als een object langer stilstaat op een bepaalde plek dat deze zwart wordt. Het object gaat in op de achtergrond. Je kunt dit aanpassen door de tijd waarover de achtergrond wordt berekend te veranderen met het history argument (in milliseconden).

```
cv2.createBackgroundSubtractorMOG2(history=1000)
```

Probeer te spelen met dit argument en kijk naar het effect ervan. Neem de tijd om te kijken hoe background subtraction werkt en de invloed van ruis.

Contour detection

Als we de achtergrond hebben verwijderd kunnen we de objecten (de witte vlekken) detecten en daaromheen een contour tekenen.



Dat doen we met de volgende code:

```
contours, hierarchy = cv2.findContours(fgmask,... ,...)
```

Je kunt elk object wat gevonden wordt bekijken op grootte. Alleen de grootste waarden wil je dan plotten met de volgende code (de kleine waarden zijn ruis):

```
for cnt in contours:  
    area = cv2.contourArea(cnt)  
    if area>100:  
        image = cv2.drawContours(frame, [cnt], -1, (0, 255, 0), 2)
```

Run nu het volgende script om dit in actie te zien: [oefening3_contours.py](#)

Bounding boxes

Als je eenmaal de contours hebt gevonden kun je ook een bounding box tekenen.



Hiervoor hoef je slechts een aantal regels aan de code toe te voegen.

```
x,y,w,h = cv2.boundingRect(cnt)
box = cv2.rectangle(frame, (x,y), (x+w,y+h), (255,0,0), 2)
```

Run het volgende script om dit om te zien hoe dit werkt: [oefening4_boundingboxes.py](#)

Template matching

Als laatste zullen we template matching behandelen. Template matching is een eenvoudige manier om objectie detectie te doen en een object ook te volgen. Toch is het is sommige gevallen effectief en hoeven we geen complexe Deep Learning algoritmen te gebruiken die we in de volgende les behandelen.



Wat template matching doet is hierboven afgebeeld. Een object, een vogel in dit voorbeeld, uit de video wordt uitgeknippt en dit is ons template. We kunnen kijken waar we dit template kunnen vinden (matchen) in de afbeelding. Als we dit vaker in een video doen kunnen we ook het object volgen. Bekijk eerst de volgende video over template matching:

<https://www.youtube.com/watch?v=2wwVogo88aY&t=184s>

Run daarna het volgende script: `playvideo_templatematching.py`

Updating template

Het template verandert niet gedurende de match. Als het object verandert (door draaiing bijvoorbeeld) kan de matching niet goed zijn. Dit probleem kun je verhelpen door het template een update te geven per frame. Aangezien elke frame niet veel verschilt van de vorige frame kun je de template aanpassen aan het object. Hiermee word je template matching meer robust.

In het eerdere script gebeurt de template update in deze code:

```
template =  
frame[top_left[1]:top_left[1]+h,top_left[0]:top_left[0]+w]
```

Multiple template matching

Met template matching kun je ook objecten tellen in een afbeelding. Bijvoorbeeld het aantal appels op de grond. Een voorbeeld hiervan is gegeven door het aantal munten in super mario: [template_matching_multiple_object.py](#)

[1] <https://www.youtube.com/watch?v=0nz8JMyFF14>

[2] <https://docs.opencv.org/4.x/>