Lanes Detection, Traffic signs Detection and Navigation System.



**BCLW**  
Bart van Maarschalkerweerd, Cees Melis, Luke van Es, Wessel Poldervaart

Datum: 5-11-2018

Raspberry Pi Competition

Inhoud

[2 HOLOCAR prototype 2](#_Toc529352606)

[2.1 Proof of concept prototype 2](#_Toc529352607)

[2.2 Hoe het is gebouwd 4](#_Toc529352608)

[3 Overzichten 10](#_Toc529352609)

[3.1 Overzicht van alle software modules 10](#_Toc529352610)

[3.2 Aanvullende hardware 10](#_Toc529352611)

[3.3 Publieke software code 10](#_Toc529352612)

[3.4 Gebruikers Instructies 14](#_Toc529352613)

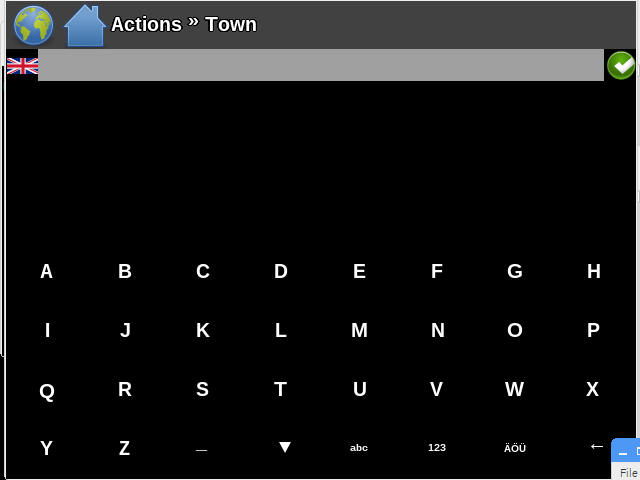
[3.5 Video presentatie 14](#_Toc529352614)

# HOLOCAR prototype

## Proof of concept prototype

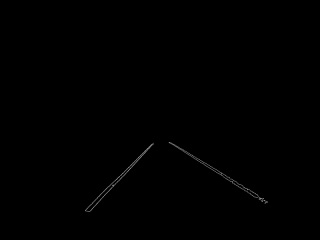
Er is een applicatie gebouwd die zowel borden als witte en gele weglijnen detecteert en deze beide visualiseert op het scherm. Bij de borden geeft de applicatie ook de tekst bij het bord weer en wordt de visualisatiekleur veranderd op basis van of dit een Advies, Verbod of een waarschuwingsbord is. Verder geeft hij linksboven ook het pictogram van het bord aan. Ook heeft deze applicatie een werkend navigatiesysteem die wordt geprojecteerd op een kleiner scherm die redelijk lijkt op een navigatiescherm. Het detecteren weglijnen/borden werkt op de Raspberry Pi maar heeft nu wat dergelijke “performance issues”. Dit kan simpelweg opgelost worden met betere hardware. Het navigatiesysteem werkt helemaal goed op de Raspberry Pi. In zijn huidige staat zijn alle delen van de software opgesplitst. De borden herkenning, lijnen detectie en navigatie kunnen in theorie tegelijkertijd werken maar helaas vanwege performance redenen hebben wij besloten om deze los van elkaar te laten werken. Op deze manier kunnen wij een overtuigender “proof of concept” plaatje laten zien. In de onderstaande voorbeelden is er te zien dat een aantal elementen zoals de lijn detectie en navigatie al uitermate accuraat en al bijna “production ready” zijn.

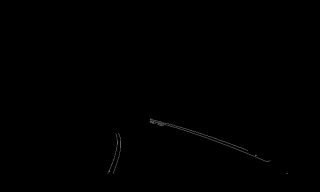
*Voorbeeld: Navigatie kaart*  


*Voorbeeld: Adres invoeren navigatiesysteem*  
  
*Voorbeeld: Advies borden detectie/meerdere tegelijk*  


*Voorbeeld: Meerdere dezelfde verbodsborden*

*Voorbeeld: Adviesbord/Waarschuwingsbord*

*Voorbeeld: Lines detectie (Visualisatie/Detectie)*

*Voorbeeld: Lines detectie scherpe bocht (Visualisatie/Detectie)*

## Hoe het is gebouwd

Ten eerste, wij hebben de **Lines/Borden Detection** op de volgende wijze gemaakt:



Open Source Computer Vision Library. Wij gebruikten Python 2.7 en hadden een Linux Ubuntu virtual machine. Dit was omdat wij dan allemaal aan het project konden werken op onze eigen pc. Met OpenCV deden wij de meeste dingen, alleen waren er een aantal andere dependencies zoals: MatPlotLib voor visualisatie en Numpy voor arraysopslag. Numpy was vooral voor de vertices en threshholds arrays. En hebben een dataset (veel images) van verkeersborden zodat wij deze konden detecteren in een real-time video/op de pi camera.  
  
De **Lines Detection** wordt voornamelijk met Image/video recognition en filtering gedaan.

**OpenCV** helpt hier dus bij, deze biedt vooral krachtige image filtering zodat de image/video recognition accuraat is.  
  
Ons grootste probleem met het huidige HOLO**CAR** prototype is performance. Dit is de hoofdreden waarom wij niet Borden/Lines detectie tegelijkertijd uit kunnen voeren. Voornamelijk het detecteren van de borden met de specifieke methode die wij voor het HOLO**CAR** prototype hebben is zeer performance heavy. En hebben hierom voor elke paar borden een andere video geladen. We kunnen deze wel runnen op de Raspberry Pi alleen is de performance niet optimaal (helemaal bij borden detectie)  
  
**Dit zijn alle filters dij wij hebben gedaan bij de Lines Detection.**  
Eerst doen wij de video **grayscalen**, dit is eigenlijk een basis die altijd bij Image recognition wordt gebruikt omdat deze dan net iets betere duidelijkheid heeft en helpt ook qua performance.

Vervolgens doen wij hier een **GaussianBlur** op simpelweg omdat de kleuren dan meer opvallen en dan ook weer accurater zijn Hiermee is het dus makkelijker om de randen van objecten en elementen te zien en kan de code ze makkelijker herkennen..

Vervolgens zetten wij hier de colorspace naar **HSV** (alternatief voor RGB), dit is de meest precieze colorspace. Dit is vooral voor de drawlayer/thresholds voor de visualisatie/detectie maar het wordt ook wel verwerkt in de filters.

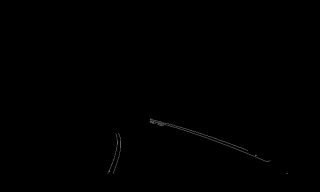
Dan komt de **Region of Interest**, oftewel **ROI**.Dit geeft eigenlijk een kijkhoek naar waar de software echt moet gaan detecteren. Sommige zullen denken: Waarom doe je niet eerst je ROI Definen en doe je daarin het grayscalen / GaussianBlur voor betere performance. Dit is omdat je er geen betere maar juist slechtere performance door krijgt. OpenCV vind het erg fijn om de hele videosource te grayscalen/Bluren i.p.v eerst een check doen of die in deze hele specifieke ROI zit. Door dit te doen hebben wij onze performance verbetert met 7.6%-8%. De ROI is over het algemeen hetzelfde maar wordt dus bij compleet verschillende camera posities wel iets aangepast. ROI is dus een belangrijk deel van ons huidige prototype. In de toekomst zouden wij hiervoor een meer dynamisch systeem maken om automatisch te kunnen bepalen aan de hand van de kijkhoek hoe de ROI moet zijn. Dit is natuurlijk iets wat alleen in het uiteindelijke HOLO**CAR** product zou komen.

*Voorbeeld ROI Bij deze video (rode lijn)*



Nu we dit allemaal hebben doen we nog niet echt een specifiek iets detecteren. De eerste stap naar dit is om onze **Threshholds** te definiëren. Dit zijn dus van die **HSV** waardes opgeslagen in een Numpy array. Wij maken 2 grenswaardes aan. De low\_white (grijsachtig) en de high\_white (volledig wit) Zodat de detection op deze kleuren kan focussen. Wij hebben de grenswaardes zo gedaan dat zij bij zowel dag als nacht een bijna perfecte detection hebben. Om het 99.9% werkend te maken zou dit eventueel op kunnen gesplitst worden in 2 scripts alleen zal dat weer meer performance eïssen. Dan moet je namelijk ook nog detecteren of het nacht is of niet.

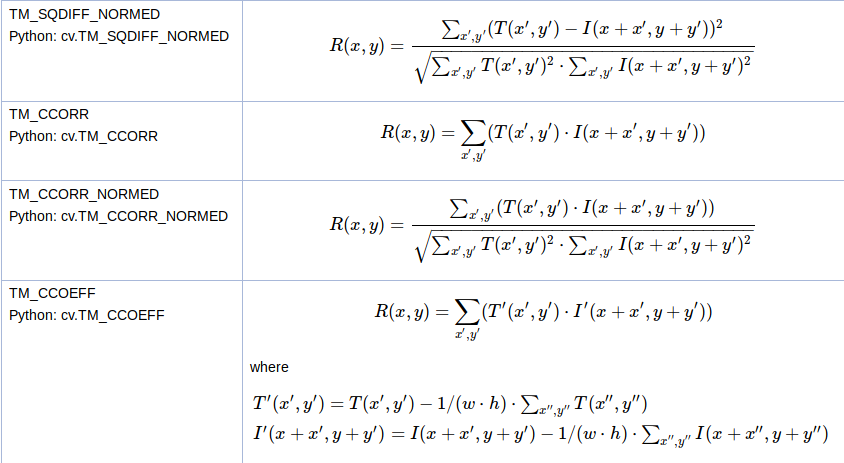
Dan komt de **Canny** methode. Dit is een proces die OpenCV voor ons doet. Deze wordt gedefinieerd met de grenswaarde en de ROI van boven. De grayscale/gausionblur helpen hierbij. Hij laat allen de randen van de gedetecteerde dingen zien en laat de rest compleet zwart. Hiermee kunnen we zien dat de detectie erg accuraat is zelfs onder verschillende weeromstandigheden hij pakt hier de edges van deze randen. Dit is dus redelijk performance heavy. Ook werken deze met snelwegen en tekent hij de wegstrepen individueel. Het berekenen van een row is gewoon te performance heavy om hier te doen. Dit zal dus ook al makkelijk geïmplementeerd worden met betere Hardware.

*Voorbeeld: Output Canny*  


Nu we de detectie van de lijnen accuraat hebben bepaald. Wat overigens een zeer belangrijk aspect is van het HOLO**CAR** prototype omdat dit ons primaire sales punt is.  
  
Hiervoor hebben wij de **HoughLinesP**. Ook al is dit nog een soort van laatste check, het wordt wel echt gezien als een tool om lines toe te voegen op de correcte plaatsen. HoughLinesP Checkt of de lijnen die worden gewrite wel verticaal zijn. (dit is altijd het geval bij lanes) hieraan geven wij aan aantal attributen zoals **MaxLineGap**(hoelang een lijn mag zijn) **Thickness** (Hoe dik deze is). En dan nog het **aantal lines** hij doet. En de **HSV**-waarde daarvan (welke kleur). Hierdoor worden er dus allerlei lijnen getekend over de detectie heen. Hiermee moesten wij simpelweg even kijken hoeveel lijnen we per video konden gebruiken, dit kost namelijk ook weer performance.

Voor het laatst hebben we nog een overlay oftewell **.addWeighted** gebruikt om overdreven lichte videos een beetje een donkere kleur te geven. Nogmaals dit is gewoon een overlay en veranderd niet de daadwerkelijke lines detectie.

*Voorbeeld hoge addWeighted*  
  
  
  
**Borden detectie oftewel “Template matching”**Template matching is een methode voor het zoeken en het vinden van de locatie van een template image in een grootere image/video. **OpenCV** komt dus met de functie **cv2.matchTemplate()** om dit te doen. Het legt de template image over de input image (de video). Ook kan je op deze een soort van sterkte zetten met detecteren (vergelijkbaar met de grenswaarde techniek) Dit wordt gedaan met een **numpy array if statement** waar dus de “sterkte” hoger is dan een bepaalde waarde (deze waarde is hetzeflde bij elke video) Dan heb je ook nog een template matching Enumerator wat eigenlijk een wiskunde som is. Dit is nogal tricky om dit echt werkend te krijgen met meerdere objecten/ video. Omdat dit vooral een methode is die de voorkeur geeft aan object recognition in een image. Template matching werkt beter met een **grayscale** dus wordt dit gedaan.  
Onder zie je 4 van de 16 verschillende Enumarations. Om heel eerlijk te zijn snap ik zelf niet helemaal wat er gebeurt. Maar wij gebruiken de TM\_CCOEFF. Ik snap de x/y coördinaten maar het wordt nogal intens om het hier stap voor stap weten uit te leggen.



Om duidelijkheid te creëren is hier een klein proces.  
Dus: Je hebt een video:   
  
In die video pak je er een bord uit en maak je er een image van en slaat deze op als template.shape:  


Dit zijn vaak low resolution images. (Beter resultaat als bord uit dezelfde video komt/low res).

Dan dus met de juiste sterkte en filters kan je bijvoorbeeld dit detecteren en hierbij tekst zetten en ook een pictogram laten zien.



Als wij dus met meerdere borden willen doen betekent dus ook dat wij meerdere keren dit moeten doen en ook dus meerdere malen keer het hele template matching proces moet doen. Dit maakt het dus zo performance heavy.

**Het detecteren wordt dus gedaan als hierboven. Hoe doen wij de visualisatie hiervan?**  
  
Ten eerste doen wij eerst een nieuwe overlay maken (anders gaat alles fout met detectie). Daarnaast doen we ook nog het font definen. Als de templatematching function dus een aantal points van detectie teruggeeft kunnen wij hiermee een **rectangle** drawen op de overlay met een custom kleur. Wij houden de kleur Blauw bij Adviesborden Rood bij Verbodsborden en Oranje bij Waarschuwingsborden. Dan vervolgens ook nog **tekst** ernaast plaatsen (met zelfde kleuren). En dan ook nog op basis van welke template image er gedecteerd linksboven een pictogram ervan zetten. Deze wordt ook geresized zodat hij mooi past. Ook als hij meerdere detecteert zet hij deze pictogrammen naast elkaar. Wij vonden dit voldoende voor een prototype waar de main focus was om het correct te detecteren.

**Navigatieoverlay**Natuurlijk is er ook nog het navigatie element. Dit word gedaan met **Navit**. Navit is een **Open-source turn-by-turn navigation software**. Oorspronkelijk was het plan hiervoor een zelfgebouwd systeem voor te maken maar eigenlijk zouden wij dan simpelweg net zo goed Navit kunnen gebruiken aangezien het alle features heeft die wij nodig hebben. Dit is waarom wij voor navit hebben gekozen. Ook is het flexibel met **custom hardware en software implementaties**. Hiervoor hebben wij een implementatie ontwikkeld die als overlay kan draaien. Echter zouden wij in productie een custom skin voor navit ontwikkelen en dit verder integreren in het systeem.

Met behulp van **GPSD** word via een seriële verbinding met de **Arduino** de gegevens van de GPS module verstuurd naar de Raspberry Pi. Deze gegevens zijn in de vorm van **NMEA** sentences. Dit is een speciaal format voor GPS gegevens. Deze gegevens worden vervolgens gelezen door Navit en dit gebeurt 10 keer per seconden (10hz update rate). Op deze wijze is het mogelijk om een uiterst accurate positie te bepalen. Ook kunnen dingen zoals snelheid en kilometer rijafstand gemakkelijk kunnen worden berekend.

# Overzichten

## Overzicht van alle software modules

* OpenCV 2.4.13.6
  + NumPy
  + MatPlotLib
* Navit {WESSEL}

## Aanvullende hardware

camera-module

Een simpele camera module maar dan met een groothoek lens. Het is belangrijk voor ons om een groothoek lens te hebben om een wijdere blik te hebben op de weg.

Adafruit Ultimate GPS Breakout - 66 channel w/10 Hz updates

Voor exacte locatie tracking is het belangrijk voor ons om een goede GPS module te hebben. Dit is waarom wij gebruik willen maken van een module met 10Hz updates. Tijdens het rijden moeten er genoeg updates zijn.

Raspberry pi Touch display

Omdat het projecteren erg lastig gaat worden is het voor het prototype erg handig om een alternatief te hebben. Ook omdat de kosten van de hardware flink omhoog zouden gaan met een projector.

LCD display  
scherm voor dingen  
  
Arduino  
boi

## Publieke software code

Dit staat allemaal op onze Github. https://github.com/WeeJeePee/RaspberryPi\_bclw ik raadt u aan om onze publieke code zo te bekijken.

1. Lane-spanje1.py – Lines detection  
     
   #Imports
2. import cv2
3. import numpy as np
4. import matplotlib
6. #Videocapture
7. video = cv2.VideoCapture("../../Video/template-matching/spanje/weg-spanje-1.mp4")
8. i = 1
9. #Region of interest
10. def roi(video, vertices):
11. maskz = np.zeros\_like(video)
12. cv2.fillPoly(maskz, vertices, 255)
13. masked = cv2.bitwise\_and(video, maskz)
14. return masked
16. while True:
17. #Read
18. ret, orig\_frame = video.read()
20. #Nieuwe laag om op te drawen
21. line\_image = np.copy(orig\_frame) \* 0
22. if not ret:
23. video = cv2.VideoCapture("../../../../video2.mp4")
24. continue
26. frame = cv2.GaussianBlur(orig\_frame, (5, 5), 0)
27. #HSV
28. hsv = cv2.cvtColor(frame, cv2.COLOR\_BGR2HSV)
29. #Kleur geel
30. low\_yellow = np.array([18, 94, 140], dtype=np.uint8)
31. up\_yellow = np.array([48, 255, 255], dtype=np.uint8)
32. #Kleur wit
33. low\_white = np.array([0,0,165], dtype=np.uint8)
34. up\_white = np.array([255,255,255], dtype=np.uint8)
35. #Inrages
36. mask1 = cv2.inRange(hsv, low\_yellow, up\_yellow)
37. mask2 = cv2.inRange(hsv, low\_white, up\_white)
38. #Merge
39. mask = cv2.bitwise\_or(mask1, mask2)
40. #Check
41. target = cv2.bitwise\_and(frame,frame, mask=mask)
42. #Canny
43. edges = cv2.Canny(target, 75, 150)
44. vertices = np.array([[325,650],[450,500],[600,500],[600,425],[650,425],[1200,650],
45. ], np.int32)
46. edges = roi(edges,[vertices])
47. lines = cv2.HoughLinesP(edges, 1, np.pi/180, 50, maxLineGap=10)
48. #cv2.polylines(frame,[vertices],True,(0,255,255))
50. if lines is not None:
51. for line in lines:
52. #AANTAL LINES
53. x1, y1, x2, y2 = line[1]
54. cv2.line(line\_image, (x1, y1), (x2, y2), (48, 255, 255), 20)
55. x1, y1, x2, y2 = line[2]
56. cv2.line(line\_image, (x1, y1), (x2, y2), (48, 255, 255), 20)
57. x1, y1, x2, y2 = line[3]
58. cv2.line(line\_image, (x1, y1), (x2, y2), (48, 255, 255), 20)
59. x1, y1, x2, y2 = line[4]
60. cv2.line(line\_image, (x1, y1), (x2, y2), (48, 255, 255), 20)

63. #Overlay
64. lines\_edges = cv2.addWeighted(frame, 1, line\_image, 1, 0)
66. #voor de window
67. cv2.namedWindow("frameOutput", cv2.WINDOW\_NORMAL)
68. cv2.namedWindow("edgesOutput", cv2.WINDOW\_NORMAL)
69. #Resize window
70. frameR = cv2.resize(lines\_edges, (800,480))
71. edgesR = cv2.resize(edges, (800,480))
73. #Zowel Frame (video) als edges laten zien
74. cv2.imshow("frameOutput", frameR)
75. cv2.imshow("edgesOutput", edgesR)
77. #Stop als Q wordt ingedrukt.
78. if cv2.waitKey(1) & 0xFF == ord('q'):
79. break
81. #Sluit alles af
82. video.release()
83. cv2.destroyAllWindows()

sign-spanje4.py – Borden detectieimport cv2

1. import numpy as np
2. #Hoofd video
3. cap = cv2.VideoCapture("../../Video/template-matching/spanje/borden-spanje\_Trim.mp4")
4. #Alle templates
5. template = cv2.imread("../../Video/template-matching/spanje/image143.png", cv2.IMREAD\_GRAYSCALE)
6. template2 = cv2.imread("../../Video/template-matching/spanje/image147.png", cv2.IMREAD\_GRAYSCALE)
7. template3 = cv2.imread("../../Video/template-matching/spanje/image140.png", cv2.IMREAD\_GRAYSCALE)
8. template4 = cv2.imread("../../Video/template-matching/spanje/image146.png", cv2.IMREAD\_GRAYSCALE)
9. #Naar shapes
10. w, h = template.shape[::-1]
11. w2, h2 = template.shape[::-1]
12. w3, h3 = template.shape[::-1]
13. w4, h4 = template.shape[::-1]
15. number1 = False
17. while True:
18. \_, frame = cap.read()
19. #Grayscale
20. gray\_frame = cv2.cvtColor(frame, cv2.COLOR\_BGR2GRAY)
21. #Font
22. font = cv2.FONT\_HERSHEY\_SIMPLEX
23. number1 = 0
24. number2 = 0
25. #TEMPLATE MATCHING (1 process)
26. res = cv2.matchTemplate(gray\_frame, template, cv2.TM\_CCOEFF\_NORMED)
27. #Numpy where
28. loc = np.where(res >= 0.7)
29. #Selector
30. for pt in zip(\*loc[::-1]):
31. #Draw rectangle
32. cv2.rectangle(frame, pt, (pt[0] + w, pt[1] + h), (0, 0, 255), 3)
33. number1 = 1;
34. #Pictogram
35. overlaySource = cv2.imread("../../Afbeeldingen/TrainingData/50km.png")
36. #Resize hiervan
37. overlay = cv2.resize(overlaySource, (100,100))
38. #Shapes
39. rows,cols,channels = overlay.shape
40. #Transparant
41. overlay=cv2.addWeighted(frame[250:250+rows, 250:250+cols],0.5,overlay,0.5,0)
42. #if else of bord ervoor is
43. if number1 == 1 and number2 == 0:
44. frame[20:20+rows, 20:20+cols ] = overlay
45. else:
46. frame[40:40+rows, 20:20+cols ] = overlay
48. #Text ernaast
49. cv2.putText(frame,'50KM',(pt[0] + 10 + w, pt[1] + h), font, 1, (0,0,255), 1)
51. #Nu komt het zelfde als hiervoor maar dan met een andere template.
52. #Een ander pictogram
53. #Voor de rest hetzelfde
54. result2 = cv2.matchTemplate(gray\_frame, template2, cv2.TM\_CCOEFF\_NORMED)
55. loc2 = np.where(result2 >= 0.7)
57. for pt in zip(\*loc2[::-1]):
58. cv2.rectangle(frame, pt, (pt[0] + w2, pt[1] + h2), (0, 0, 255), 3)
59. number1 = 1;
60. overlaySource = cv2.imread("../../Afbeeldingen/TrainingData/50km.png")
61. overlay = cv2.resize(overlaySource, (100,100))
62. rows,cols,channels = overlay.shape
64. overlay=cv2.addWeighted(frame[250:250+rows, 250:250+cols],0.5,overlay,0.5,0)
65. if number1 == 1 and number2 == 0:
66. frame[20:20+rows, 20:20+cols ] = overlay
67. else:
68. frame[40:40+rows, 20:20+cols ] = overlay
69. cv2.putText(frame,'50KM',(pt[0] - 50 - w2, pt[1] + h2), font, 1, (0,0,255), 1)
71. result3 = cv2.matchTemplate(gray\_frame, template3, cv2.TM\_CCOEFF\_NORMED)
72. loc3 = np.where(result3 >= 0.7)
74. for pt in zip(\*loc3[::-1]):
75. cv2.rectangle(frame, pt, (pt[0] + w3, pt[1] + h3), (0, 60, 255), 3)
76. number2 = 1
77. overlaySource = cv2.imread("../../Afbeeldingen/TrainingData/kromme\_bocht\_links.png")
78. overlay = cv2.resize(overlaySource, (100,100))
79. rows,cols,channels = overlay.shape
81. overlay=cv2.addWeighted(frame[250:250+rows, 250:250+cols],0.5,overlay,0.5,0)
82. frame[20:20+rows, 20:20+cols ] = overlay
83. cv2.putText(frame,'Sligt turn left',(pt[0] - 200 - w2, pt[1] + h2), font, 1, (0, 60, 255), 1)
85. result4 = cv2.matchTemplate(gray\_frame, template4, cv2.TM\_CCOEFF\_NORMED)
86. loc4 = np.where(result4 >= 0.7)
88. for pt in zip(\*loc4[::-1]):
89. cv2.rectangle(frame, pt, (pt[0] + w4, pt[1] + h4), (0, 60, 255), 3)
90. number2 = 1
91. overlaySource = cv2.imread("../../Afbeeldingen/TrainingData/kromme\_bocht\_links.png")
92. overlay = cv2.resize(overlaySource, (100,100))
93. rows,cols,channels = overlay.shape
95. overlay=cv2.addWeighted(frame[250:250+rows, 250:250+cols],0.5,overlay,0.5,0)
96. frame[20:20+rows, 20:20+cols ] = overlay
97. cv2.putText(frame,'Slight turn left',(pt[0] - 200 - w2, pt[1] + h2), font, 1, (0, 60, 255), 1)

100. #Resize de frame
101. frameR = cv2.resize(frame, (800,480))
102. #Laat zien
103. cv2.imshow("Frame", frameR)
104. #Debug print voor als er een bord is
105. print number1
107. key = cv2.waitKey(1)
108. #Press q close all
109. if cv2.waitKey(1) & 0xFF == ord('q'):
110. break
111. #alles uitdoen
112. cap.release()
113. cv2.destroyAllWindows()

## Gebruikers Instructies

Nogmaals, dit is nog niet te gebruiken voor de gebruiker, wij moeten dit nog verder ontwikkelen en hebben betere hardware nodig.

## Video presentatie

Dit wordt gedaan in de klas.