|  |  |
| --- | --- |
|  | **2014** |
|  | Hogeschool Utrecht  Bryan Baan – 1625054  Chanan van Ooijen – 1611621  Lars Veenendaal – [input studentnr hier]  Mitchell Werensteijn - 1624291  Yusuf Syakur - 1633276 |

|  |
| --- |
| **[Technisch rapport]** |
| In dit document kunt u onze bevindingen vinden over het gedane experiment. |

Contents

[Wat gaan we testen? 2](#_Toc382114710)

[De te testen algoritmen 2](#_Toc382114711)

[Algoritme 1 2](#_Toc382114712)

[Algoritme 2 3](#_Toc382114713)

[Hoe is de test verlopen + resultaten 4](#_Toc382114714)

# Wat gaan we testen?

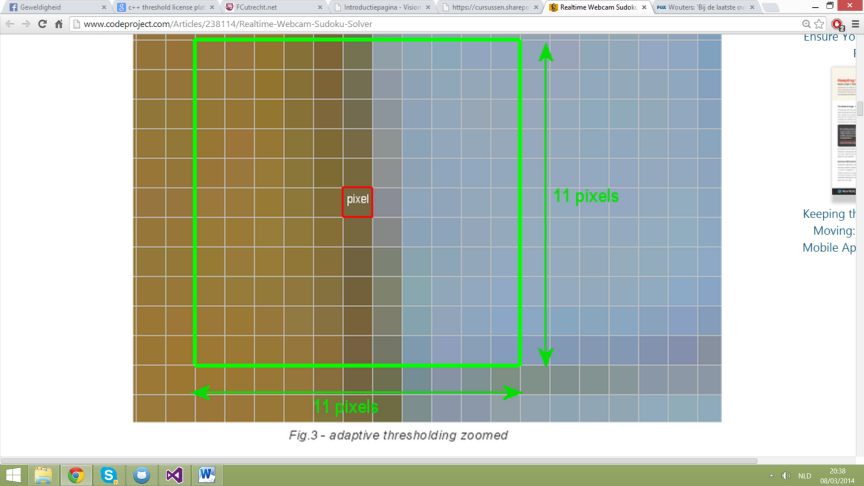
Er moet deze week software gemaakt worden die kentekenplaten op foto’s kan lokaliseren. Om vast te stellen hoe goed de geschreven software werkt zullen de algoritmes aan een test onderworpen moeten worden. Bij de opdracht wordt een zipje geleverd die een aantal foto’s bevat daarop verschillende kentekenplaten. Ook zit er bij het zipje een XML bestand met wat de hoekcoördinaten van de kentekenplaten op de foto’s zijn.

Ons eerste algoritme gaat bepalen wat hij denkt dat de hoekcoördinaten zijn van de kentekenplaat in een foto. Aan de hand van deze foto voert het algoritme een aantal ingewikkelde berekeningen uit en creëert hij als het ware nieuwe plaatjes. Het resultaat zal uiteindelijk 4 hoekpunten moeten opleveren die overeenkomen met de hoekpunten die in het meegeleverde XML file staan.

Het tweede algoritme zoekt de gele kleur in een plaatje. Als hij deze heeft gevonden, worden er weer berekeningen uitgevoerd. Als eindresultaat zullen er, net zoals het vorig besproken algoritme, 4 hoekpunten moeten opleveren die overeenkomen met de hoekpunten die in het meegeleverde XML file staan.

# De te testen algoritmen

## Algoritme 1

Het algoritme dat hier zal worden beschreven werkt door middel van edge detection en binarization. Eerst wordt er een origineel plaatje gekozen. Dit plaatje wordt dan bewerkt door een grayConverter die de kleuren omzet naar grijswaarden. Op het plaatje met grijswaarden laten we een 11x11 mediaan filter los. Als er bijv. salt&pepper-ruis in het plaatje zit zal dit door het mediaan filter worden weg gefilterd. Het resultaat wordt opgeslagen als een nieuw plaatje.

Figuur 1 - Thresholding

Op dit nieuwe plaatje laten we thresholding los. Dit houdt in dat we voor elke pixel in het plaatje (uitgezonderd de buitenste 5 rijen pixels) kijken wat waarden van de 11 bij 11 neighbourhood is (zie figuur 1). Alle omringende pixel waarden worden bij elkaar opgeteld en gedeeld door het aantal pixels binnen de neighbourhood (121 in dit geval). Het getal wat daar uitkomt, is de zogenaamde mean waarde die wij gebruiken als drempelwaarde. Nu wordt gekeken of de waarde van de pixel waar je op dat moment bent (pixel met rood vakje eromheen in figuur 1) hoger of lager is dan de drempelwaarde. Is de waarde van de pixel hoger dan de drempelwaarde dan zet je de drie kleurkanalen van de pixel op 0 en wordt de pixel zwart. Zo niet (de drempelwaarde is lager dan de pixel waarde) dan zet je de drie kleurkanalen van de pixel op 255 en wordt de pixel wit. De uitkomst hiervan wordt opgeslagen als een nieuw plaatje.



Figuur 2 – Na thresholding

Nadat het plaatje zwart wit is gemaakt wordt er een edge detection algoritme toegepast. Deze heet de sobel edge detection. De algoritme kijkt naar de gradiënt. Dit vertegenwoordigt de intensiteitsverandering in het plaatje. Om de verandering te zoeken worden twee maskers toegepast aan het plaatje. Één voor de intensiteitsverandering in de X richting en een tweede voor de Y richting. Deze gradiënten worden bij elkaar opgeteld en zo krijg je de edges.



Figuur 3 – Na sobel edge detection

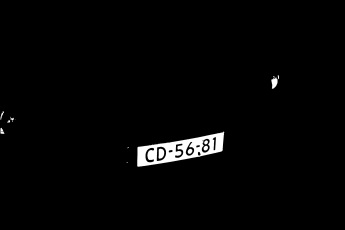
Op basis van het nieuwe plaatje (dat uitsluitend bestaat uit witte en zwarte pixels) worden 2 histogrammen gemaakt. Dit nieuwe plaatje zal, omdat het wordt opgeslagen als .jpg, allerlei andere kleuren en tinten bevatten. Om het eerste histogram te maken wordt het plaatje uitgelezen regel voor regel in horizontale richting. Per regel wordt het aantal veranderingen van wit naar zwart of zwart naar wit bijgehouden. Op basis daarvan wordt een histogram gemaakt (Bij het kenteken zijn er veel veranderingen van zwart naar wit of andersom en in het histogram is op die plek een grote piek te zien). Om het tweede histogram te maken wordt het plaatje uitgelezen regel voor regel in verticale richting. Per regel wordt het aantal veranderingen van wit naar zwart of zwart naar wit bijgehouden. Op basis daarvan wordt een histogram gemaakt (Bij het kenteken zijn er veel veranderingen van zwart naar wit of andersom en in het histogram is op die plek een grote piek te zien).

## Algoritme 2

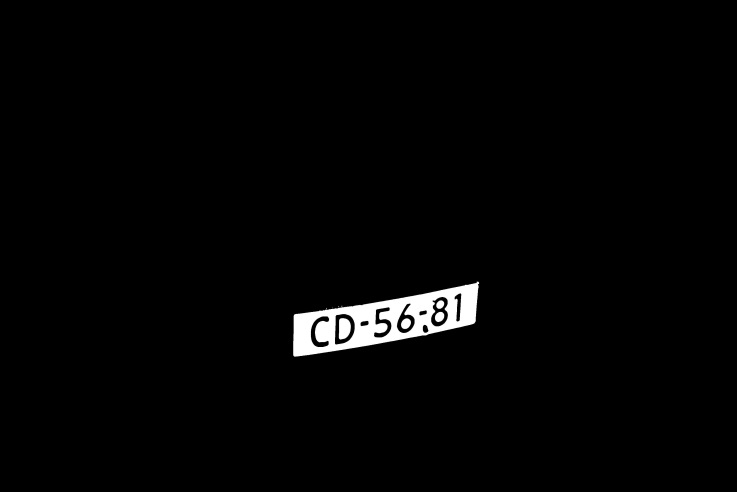
Het tweede algoritme dat is geschreven werkt door middel van het zoeken van de kleur geel in een plaatje. Omdat de opdracht was dat er alleen nederlandse kentekens mee moesten worden opgespoort, bleek ons dit een goede methode om mee te starten. Dit bleek toch iets moeilijker dan verwacht.



Het algoritme zoekt dus de kleur geel in een plaatje. Als hij de pixels gevonden heeft met een gele waarde, worden alle pixels die deze waarde hebben omgezet naar witte pixels en de rest naar zwarte pixels. Als dit gedaan is wordt er een median filter op het plaatje losgelaten. Dit filter zorgt ervoor dat alle ‘loze’ pixels, pixels die als ruis kunnen worden gezien, verwijderd. Als dit is gebeurt zullen er histogrammen worden gemaakt van dit plaatje. Deze histogrammen zullen het verdere verloop van het proces bepalen.



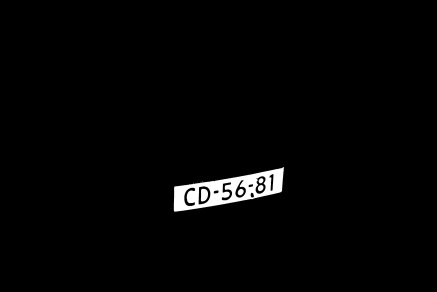
Nadat de histogrammen gemaakt zijn, wordt er een algoritme gebruikt dat ervoor zorgt dat zo goed mogelijk alleen het kenteken wordt gemeten. Wat dit algoritme doet is het zoeken van een top in de histogrammen. Als deze is gevonden (hoogste top is meestal (deel van) een kentekenplaat) zal hij beide kanten oplopen om te kijken op welke X of Y waarden het kenteken stopt. Deze slaat hij op en creëert hier een nieuw plaatje mee. Dan worden de laatste 2 stappen (histogram en histogrambewerking) herhaald en onstaat het resultaat:



Op dit plaatje is duidelij te zien dat het kenteken de letters en nummers “CD-56-81” bevat, net zoals in de plaatjes die hierboven zijn afgebeeld. Ook klopt de positie van het kenteken, wat wil zeggen dat ons algoritme zijn werk goed doet.

# Hoe is de test verlopen + resultaten

De tests van het eerste algoritme zijn minder goed verlopen dan verwacht. De histogrammen die werden gegenereerd waren niet te gebruiken om te kenteken te lokalizeren. We vermoeden dat het ligt aan de sobel edge detection. Omdat de sobel edge detection ook randen detetecteert zoals bossen en dergelijke, is het onmogelijk om de coordinaten van de kenteken te zoeken aan de hand van het aantal randen per rij pixels. Als we gebruik hadden gemaakt van canny edge detection, die echter alleen de sterke randen pakt, dan zou in theorie de bossen niet herkent moeten worden maar het kenteken wel.

De tests over het tweede algoritme zijn goed verlopen, maar niet perfect. De plaatjes die zijn meegestuurd zijn goed verwerkt om tot een goed resultaat te komen, hier volgt een voorbeeld.

Zoals op deze plaatjes te zien is wordt het kenteken leesbaar uit het plaatje gehaald. Ook is hier nog een voorbeeld dat niet zo goed ging.

Zoals te zien is op dit plaatje zijn de middelste letters, FF, niet goed zichtbaar op het verwerkte plaatje. Over het algemeen is het algoritme gelukt voor 8/11 plaatjes wat geen slecht resultaat is. De gevonden coordinaten worden vergeleken met de coordinaten in het xml bestand. De nauwkeurigheid van onze algoritme is hieronder uitgedrukt in percentage: