Schaduw en overbelichting uit een kenteken verwijderen

School:Users:gebruiker:Desktop:Schermafbeelding 2014-04-10 om 10.52.50.png

Hendrik Cornelisse

Jeroen Huisen

Lars Veenendaal

Arthur van der Weiden

# Inhoudsopgave

Inleiding 3

Algoritmen 4

Inleiding 4

Schaduw ontdekken 4

Overbelichting ontdekken 4

Schaduw en overbelichting neutraliseren 5

Controller 5

Meetresultaten 6

# Inleiding

In het gevonden kenteken bevindt zich soms schaduw en/of overbelichting. Omdat dit storend kan zijn voor het herkennen van de karakters binnen het kenteken, is het beter dat de storende belichting geneutraliseerd wordt. Echter omdat niet ieder kenteken schaduw/overbelichting bevat, leek het nuttig om eerst te testen of een van deze 2, of misschien zelfs beide, zich in het kenteken bevatten. Wanneer er namelijk geen van beide in het kenteken voorkomen, is het ook niet nodig om de afbeelding te wijzigen. Dit zorgt er dan vervolgens weer voor dat er tijdswinst is voor het totale programma.

Als het neutraliseren van schaduw en overbelichting niet gedaan zou worden zouden er situaties kunnen ontstaan waarin letters verdwijnen wanneer de kleuren naar binaire waarden geconverteerd worden.

# Algoritmen

## Inleiding

Om goed en snel tot een eindresultaat te komen is er voor gekozen om eerst een check te doen. In deze check wordt gekeken of er schaduw of overbelichting in het gevonden kenteken te ontdekken is. Wanneer dit niet het geval is kan de afbeelding in zijn volledigheid door. Echter wanneer er wel schaduw en/of overbelichting in het kenteken ontdekt is wordt het storende licht eerst geneutraliseerd.

## Schaduw ontdekken

Om te kunnen checken of er schaduw in het gevonden kenteken zit moeten er eerst een aantal waarden bekent zijn. Deze waarden zijn: de hoek die zich linksboven bevindt, de hoek die zich rechtsboven bevindt, de hoek die zich linksonder bevindt en de hoek die zich rechtsonder bevindt. Ook moet er een plaatje bekend zijn. Vervolgens gaat het algoritme één keer door het gevonden kenteken lopen. Dit gebeurt om de donkerste waarde van het kenteken te vinden en op te slaan. Deze donkerste waarde wordt gebruikt als drempelwaarde, zodat de zwarte letters niet als schaduw worden herkent.

Wanneer de donkerste waarde bekend is wordt er nog een keer door het kenteken gelopen. Deze keer wordt, wanneer de grijswaarde van de pixel boven de donkerste waarde en onder de grijswaarde van het geel in het kenteken, een teller, genaamd *ShadowPixels*, verhoogd met 1. Deze teller wordt gebruikt om het aantal procent schaduwpixels te berekenen. Wanneer het kenteken meer dan 2% aan schaduwpixels bevat, wordt het algoritme doorbroken en geeft het de waarde “true” terug. Wanneer het kenteken minder dan 2% schaduwpixels bevat wordt de waarde “false” teruggegeven.

## Overbelichting ontdekken

Om overbelichting te ontdekken worden de x- en y-gradiënt berekend. Deze twee gradiënten worden bij elkaar opgeteld en opgeslagen in een variabele met de naam *sum*. Voordat er wat met *sum* moet eerst zeker gesteld worden dat *sum* binnen de kleuren-grens van 0-255 ligt. Om hier zeker van te zijn wordt een test uitgevoerd: als *sum* boven de 255 komt, wordt *sum* teruggebracht tot 255. Als *sum* onder de 0 komt, wordt *sum* op 0 geforceerd. Wanneer de waarde van *sum* 255 heeft bereikt wordt er gekeken of de pixel als overbelicht wordt beschouwd. Dit gebeurt door te kijken of een pixel een te hoge waarde heeft, tegelijkertijd wordt er een check uitgevoerd op de 100 omliggende pixels. Iedere pixel binnen dit grid wordt gecheckt of deze overbelicht is. Een pixel wordt als overbelicht beschouwt als de blauwe waarde boven de 210 is en de rode/groene waarden boven de 200 zijn. Als een pixel als overbelicht beschouwt is, wordt er een teller verhoogt met 1. Deze teller genaamd *Overexposed\_pixels* wordt gebruikt om het aantal procent overbelichte pixels te berekenen. Wanneer het percentage overbelichte pixels in het kenteken groter is dan 1% wordt de waarde “true” teruggegeven. Als er minder dan 1% overbelichte pixels is gevonden wordt de waarde “false” teruggegeven.

## Schaduw en overbelichting neutraliseren

#### On-geïmplementeerd algoritme

Er wordt in een gebied van 100x100 pixels gezocht naar overbelichte pixels. Wanneer zo’n overbelichte pixel gevonden is kijkt hij in het gebied naar alle gele pixels en neemt het gemiddelde van die kleurwaarden. Het gemiddelde van die gele pixels wordt de nieuwe waarde van de overbelichte pixels binnen het gebied.

#### Geïmplementeerd algoritme

Eerst wordt het gevonden kenteken geconverteerd van RGB (Red, Green, Blue) naar CMYK (Cyan, Magenta, Yellow, Key). Deze conversie wordt gebruikt om de juiste kleur geel te kunnen bepalen. Het “Key” kanaal wordt gebruikt om de zwarte letters te scheiden van de overbelichting. Eerst worden de pixels met K groter dan 0 en Y kleiner dan 0,1 op de kleur zwart gezet. Terwijl de letters extra zwart worden gemaakt wordt ook gezocht naar overbelichte pixels. Een pixel wordt als overbelicht beschouwt als alle kanalen lager zijn dan 0,1. Deze pixels worden omgezet naar de gevonden juiste geelwaarde. De juiste geelwaarde wordt gekozen als de waarde van K lager is dan 0,1 en de waarde van Y hoger is dan 0,1.

## Controller

Om het hele project, en alle klassen, aan elkaar te koppelen is er een “controller” klasse in het leven geroepen. In deze controller klasse worden alle andere klassen geïmplementeerd. Eerst wordt er gecheckt of de afbeelding die ingevoerd is ook echt bestaat/ingelezen kan worden. Wanneer dit niet kan wordt direct het hele programma afgesloten. Om de beurt voeren de verschillende klassen hun algoritmes uit. Wanneer er een exception ontvangen wordt gaat de controller kijken waardoor de exception is ontstaan. Als de controller heeft ontdekt waar het mis is gegaan, dan kiest de controller wat de passende oplossing is. Na iedere klasse wordt de tijd uitgeprint. Die tijd is een weergave van de tijd die de klasse heeft gebruikt, zodat wanneer de 3 seconden overschreden worden, gezien kan worden waar het mis is gegaan.

# Testresultaten

