1. Oog Lokalisatie

1.1. Namen en datum

Sander Kolman
Alexander Freeman
4-5-2016

1.2. Doel

Voor het laatste deel van het lokaliseren van het gezicht moeten de ogen accurater gevonden worden. De implementatie krijgt een gedeelte van het gezicht waar de ogen zich moeten bevinden en zal daaruit de exacte locatie van de ogen kunnen bepalen.

1.3. Methoden

De ooglocalisatie bestaat uit verschillende stappen die op verschillende manieren gedaan kunnen worden.
Allereerst willen we de globale positie van de ogen vinden. De eerste manier hiervoor is door de positie van de neus te vinden (die we al meekrijgen) en dan een gebied naar boven pakken met een hoogte met de ratio van het hoofd, bijvoorbeeld ¼. Het nadeel hiervan is, dat het niet altijd zo hoeft te zijn dat de ogen in dat gebied liggen. Hierdoor kan de ooglocalisatie mislukken.

Een andere mogelijkheid is door bekende posities van het hoofd te gebruiken. Namelijk de:

1. Linkerkant van het hoofd
2. Rechterkant van het hoofd
3. De neus
4. De bovenkant van het hoofd

Door de x en y posities van deze locaties te nemen, kunnen we een bounding box creeëren rondom het hoofd. Dit neemt echter veel extra ruimte mee en ook het voorhoofd. Het haar zal het lastiger maken om de ogen te detecteren. Dit is zo, omdat het haar op histogrammen veel kleine pieken veroorzaken.

Bij beide methoden kunnen we de linkerkant en de rechterkant van het hoofd er nog ‘afhalen’. Dit doen we door bijvoorbeeld 1/16 van de breedte van het hoofd van beide kanten van het plaatje af te halen. Dit zorgt ervoor dat we minder onnodige data hebben.

De bounding box van de ogen uit dit gebied halen is nog wat lastiger. We denken dat we de ogen kunnen localiseren door middel van meerdere histogrammen. We beginnen bovenaan het globale de ogen en noteren op welke y positie twee grote pieken optreden in het midden en wanneer deze verdwijnen. Zo weten we de hoogte van de box waarin de ogen liggen.

Door te meten wat de meeste linkse positie en de meest rechtse positie van de pieken zijn, weten de breedte. Aangezien we dan ook de positie weten, hebben de box te pakken.

1.4. Keuze

Wij gaan werken met de meest makkelijke manier. Om de globale plek te vinden, creëren we een bounding box vanaf de neus en dan ¼ van het hoofd naar boven. Hier hebben we voor gekozen, omdat het alleen faalt in extreem rare situaties en het het meest makkelijk te implementeren is.
Aangezien we maar 1 methode voor de ooglocalisatie hebben, gebruiken we die.

Om de onderkant van de ogen te vinden kijken we vanaf de neus waar de eerstvolgende grote piek begint door een histogram te maken van de neus omhoog. Op een zelfde manier wordt ook de bovenkant gevonden. Alleen gaat het dan om het dal wat na de piek komt van het begin van de ogen. We blijven vanaf de onderkant kijken omdat we dan geen verwarring krijgen door wenkbrauwen die samen lopen met haar of wenkbrauwen die niet te zien zijn. Van te voren zijn de zijkanten van het hoofd er ook al afgehaald zodat hier eveneens geen verwarring mee ontstaat in de pieken.
Als dit nu wordt uitgesneden houden we een afbeelding over waar in de hoogte alleen de ogen zichtbaar zijn. Nu hoeven we dus alleen nog te kijken in de breedte waar de ogen beginnen en waar ze eindigen. Dit kan weer gedaan worden met pieken detecteren. Omdat de zijkanten al verwijderd waren, kan simpelweg worden gekeken waar het zwart begint en dat als beginpunt van het oog gebruiken. Het eindpunt spreekt nu ook voor zich, dat is waar de afbeelding weer voornamelijk wit bevat. We zullen hier nog ratio's voor gebruiken om zeker te weten dat we in het goede gebied kijken.

1.5. Implementatie

We schrijven een StudentKernel klasse voor het implementeren van bewerkingen op afbeeldingen door middel van kernels. De kernel krijgt een afbeelding en voert zijn bewerking daarop uit. Deze voert bijvoorbeeld de dilution kernel uit, die in de opdracht beschreven staat. De StudentKernel slaat een kernel op door middel van een 2D array van doubles. Ook krijgt hij een factor mee om de uitkomsten mee te delen en een shift, die erbij op wordt geteld. Dit is nodig, omdat sommige operaties anders negatieve waarden opleveren. Een voorbeeld hiervan is de edge detection kernel.

Daarnaast schrijven we een StudentHistogram klasse, die operaties kan doen op histogrammen, zoals bijvoorbeeld het vinden van pieken.
Verder hebben we ImageUtils class die histogrammen uit plaatjes haalt en andere 'utility' methoden bevat.

Met de building blocks die hierboven staan, gaan we ons algoritme implementeren in de StudentLocalisation class.

1.6. Evaluatie

We testen ons algoritme op basis van de testsets. Ons algoritme moet voldoende en correcte informatie kunnen geven aan de extractie algoritmen en zodat extractie stap 1 succesvol de ogen kan vinden. Daarnaast testen we de performance in vergelijking met het standaard algoritme.