# SIGB Opgave 1: Eyetracker

## Introduktion og problembeskrivelse

Gaze-tracking er en metode hvor man kan identificere hvilket område en bruger ser på, på en skærm. At kunne identificere elementer i øjet ved eye tracking er essentielt for at kunne lave gaze-tracking, specielt identifikation af iris, pupil og glints (afspejlninger af lys i hornhinden) er her interessant. Under udførsel af opgaven nåede vi dog aldrig frem til at lave gaze tracking, men udførte de fleste skridt på vejen dertil, herunder blob detection, Hough transformation, ellipse fitting og k-means clustering. Vi vil undervejs i rapporten forklare teori, implementation og tests af disse elementer, samt diskutere fordele og ulemper ved de enkelte metoder.

Vi har derfor arbejdet med 3 delopgaver:

- Pupil- og glint-identificering
- Identificering af iris
- Segmentering af øje-billedet i clusters, en muligt alternativ fremgangmåde (eller supplement) til de to foregående opgaver.

### Pupil- og glint-identificering

Første delopgave omfatter identificering af pupiller og glints i gråtonebilleder. Til dette har vi overordnet anvendt to metoder; thresholding og blob detection.

#### Thresholding

En simpel løsning til at finde henholdsvis pupil- og glint-pixels i et gråtonebillede er ved brug af thresholding på hele billedet med en enkelt intensitetsværdi. Thresholding forklares i det følgende. Givet et billede *f(x)* vil det resulterende billede efter thresholding, *g(x)*, opnås ved følgende beregning (med antagelse af intensiteterne er 8-bit-værdier, dvs. 256 muligheder):

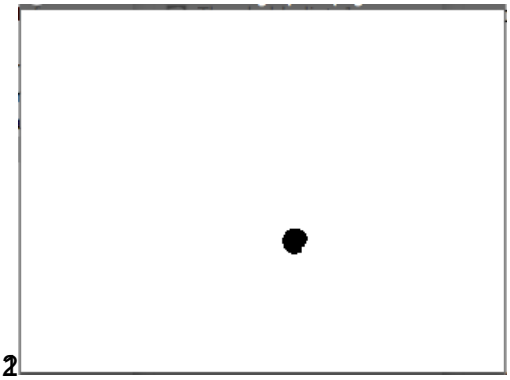
$$g(x) = \begin{cases} 0 \; if \; f(x) \leq T \\ 255 \; otherwise \end{cases}$$ [1]

Her er *T* en værdi i intervallet [0;255]. Dette vil virke hvis vi antager, at pupillens intensitet er den laveste i billedet (med værdien 0, eller tæt på 0) samt at glintens intensitet er den højeste i billedet (tæt på 255).

#### Test af pupil- og glintdetektion med thresholding

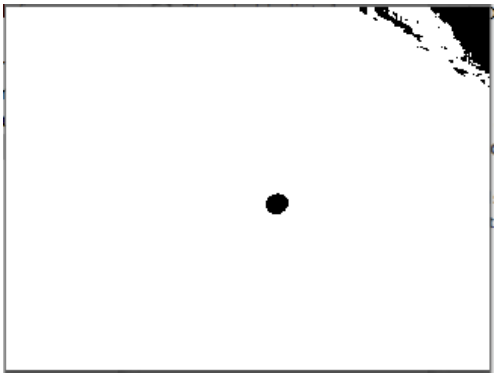
1: Thresholding på billedet (2) med *T=100* (Pupildetektion)
2: Billede af øje

3: Thresholding på billedet (2) med *T=250* (Glintdetektion)
4: Thresholding på et andet billede med *T=100* (Mindre succesfuld pupildetektion)



2

3



4

Ovenfor vises et eksempel på hhv. pupil- og glintdetektion med thresholding. Som det ses fungerer pupildetektion ganske godt med thresholding, idet der ikke ses andet end pupil-pixels i det resulterende billede (1). Dog er dette ikke altid tilfældet. På billede (4) bemærkes, at visse andre mørke elementer er med på billedet, og vi må derfor konkludere at thresholding alene ikke er en optimal løsning til pupildetektion. Under løsning af opgaven kom vi ofte ud for situationer lignende (4), når vi udelukkende brugte thresholding.
Detektion af glints med thresholding fungerer generelt ikke særlig godt på de billedesekvenser vi afprøvede. På billede (3) ses thresholding med *T=250*, hvilket er en ganske høj intensitetsværdi, og alligevel fås ikke rigtig noget brugbart resultat. Ud fra de data vi havde under løsning af opgaverne viste glintdetektion kun ved brug af thresholding sig altså ikke som en effektiv metode.

Fælles for både pupil- og glintdetektion med thresholding er, at resultaterne ikke er særligt konsistente. De er meget afhængige af lysforholdene på det enkelte billede, så det er svært at definere en thresholding-værdi som kan bruges i alle tilfælde; metoden fejler altså, når lysforholdene mellem på hinanden efterfølgende billeder ændrer sig. Thresholding-værdierne er i dette tilfælde fastsat efter et skøn, men de kunne med fordel have været defineret gennem klassificering[2] - det har vi dog ikke gjort.

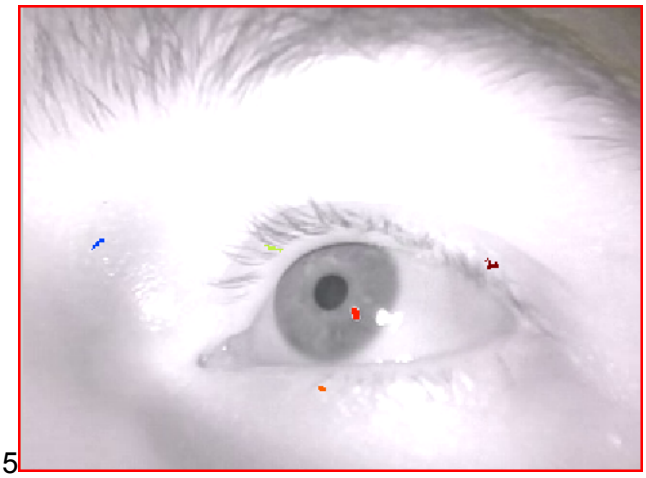
## Blob detection

For at præcisere pupil og iris-områderne grupperede vi de sammenhængende pixels i “blobs”.

Vi lavede derefter nogle antagelser om mængden af pixels der ville være i henholdsvis pupil- og glint-blobs, og eliminerede derefter blobs der enten var for store eller for små til at være potentielle kandidater. Klassifikation af blobs skete ved segmentering af billedet igennem thresholding og connected component analysis. Herefter undersøgte vi arealet af alle de fundne blobs, og eliminerede dem, der lå uden for det fastsatte interval.

Vi fandt ud af at en pupil skulle indeholde mellem 85 og 250 sammenhængende pixels. 250 er sat ret højt, men eliminerede de fleste problematiske blobs der var til stede med hensyn til pupil-detektion efter thresholding. Vi bestemte derefter at en glint skulle indeholde mellem 7 og 40 sammenhængende pixels for at fungere bedst på vores videosekvens.

Dette havde dog stadig sine mangler med hensyn til glints. Den fandt de glints vi forventede, og en masse andre falske kandidater. Se billede (5).



5

Vi gjorde dog den antagelse at glints altid vil ligge inden for en vis afstand af pupillen, og vi kunne relativt let finde midtpunktet af pupillen samt dens størrelse, og kunne derefter eliminere blobs der lå for langt fra fra pupillen. Afstanden var baseret på størrelsen af pupillen; dette kan dog være en farlig fremgangsmåde, hvis størrelsen af pupillen ændrer sig. Dette var dog ikke et stort problem i vores sekvens, men kan være et problem for en helt generel løsning.

Efter at have elimineret usandsynlige glint-kandidater fandt vi også midtpunktet for glints, og det kunne i de fleste tilfælde finde iris og glints. Se billede 6.

6 

Det er dog problemer med denne fremgangsmåde, da hvis man ikke finder pupillen, eller finder en forkert pupil, så finder man heller ikke de rigtige glints. Dette skyldes at vi brugte afstanden fra pupillen til at beregne mulige glints. Når pupillen ikke blev fundet, fejlede vores løsning altså. Derudover havde vi problemer med vores fremgangsmåde, når glints var uventet store eller små, da vores estimat på størrelsen således ikke ville fange de rigtige glints.

**Problemer vi ikke nåede at få løst**
Principielt fungerede pupil- og glint-identifikationen rigtigt godt, men der var dog nogle problemer. Tit ville den kun finde en ud af to gode glint-kandidater, hvilket både kunne være et problem med thresholding eller med blob-størrelsen. Et andet problem var at der kunne findes flere tilstødende blobs med hvert deres centrum, som burde have blevet smeltet sammen. Dette kunne f. eks. gøres ved brug af morphologisk operator, blurring eller noget lignende. Problemet var at man kunne finde op til 4 glints i den videosekvens vi brugte, hvor der højest skulle have være 2. Til visualisering var dette ikke noget specielt problem, men det ville det nok blive hvis vi implementerede gaze-trackeren.
Nogle gange kunne pupillen heller ikke identificeres. Dette kunne igen være på grund af svingende blob-størrelser eller intensiteten af pupil-farven.

## Identificering af iris

Til identificeringen af iris har vi brugt Hough-transformation som er en generel teknik til at identificere elementer i et billede. Et generelt problem med identificering af elementer er at de i mange tilfælde ikke passer perfekt med hvad der ledes efter. Et billede kan være taget fra en anden vinkel, lysforholdene kan være dårlige, og så videre. Med Hough transformation kan visse af disse problemer løses.
En ting man skal huske er at man ikke arbejder med alle pixels i billedet, kun dem med informationer i sig. Det vil sige at hvis man har en stor flade med samme farve er der ikke nogen information i den. Kraftig overgang mellem to farver har information i sig da det viser at der er en kant i billedet og det er den information der ledes efter[3].

Hvis man eksempelvis leder efter en linje, og et lille stykke af linjen mangler, kan dette stoppe identificeringen af den. Hough-transformation prøver at løse dette og andre problemer ved at konvertere punkter indeholdende information i et euklidisk koordinatsystem til kurver i Hough-rummet, defineret som:

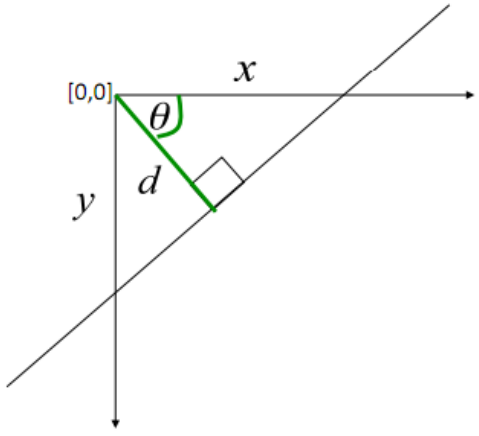
$$Hough\,(m,b)$$

hvor *m* er en linjes hældningskoefficient og *b* er dens skæringspunkt med y-aksen (i et euklidisk koordinatsystem). Hvis der er nok kurver der overlapper i ca. samme punkt, kan en kant identificeres. Den første beskrivelse af Hough transformation involverede kun identifikation af linjer, men er senere blevet udvidet til at virke med andre andre typer elementer f. eks. cirkler.

Identificering af en linje er defineret således:

"Given a set of points (x,y), find all (m,b) such that y = mx+b" [4]

Det vil sige, vi konverterer et punkt til en linje via linjens ligning.
Et problem med denne måde er at det ikke kan repræsentere vertikale linjer, så i stedet for denne teknik kan man benytte polære koordinater til at beskrive linjen.



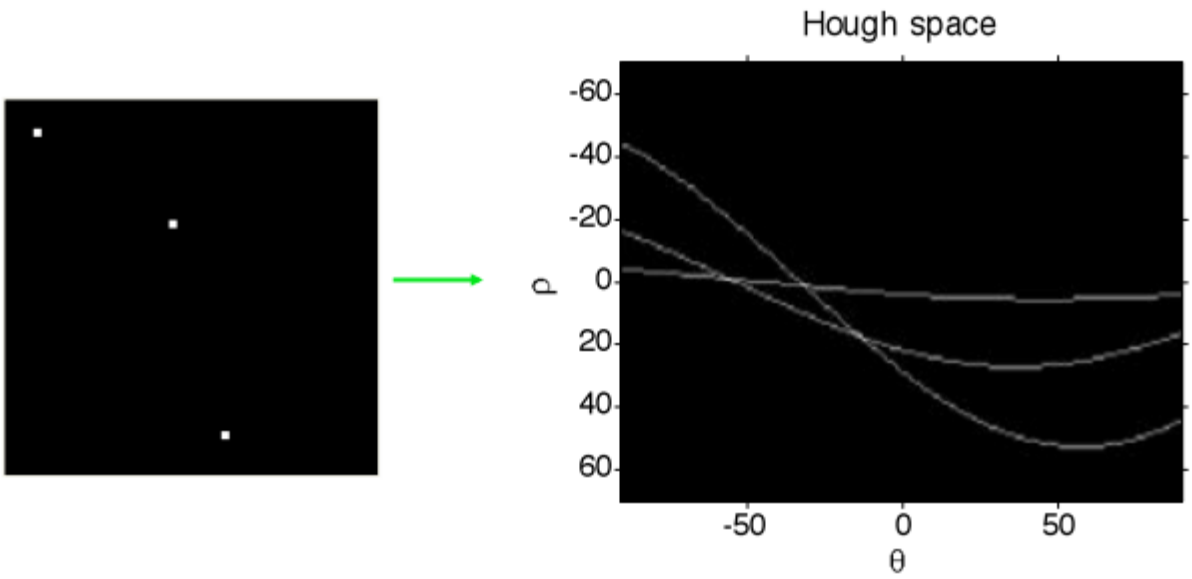

**[4]**

$$xcos(\theta) - ysin(\theta) = d$$

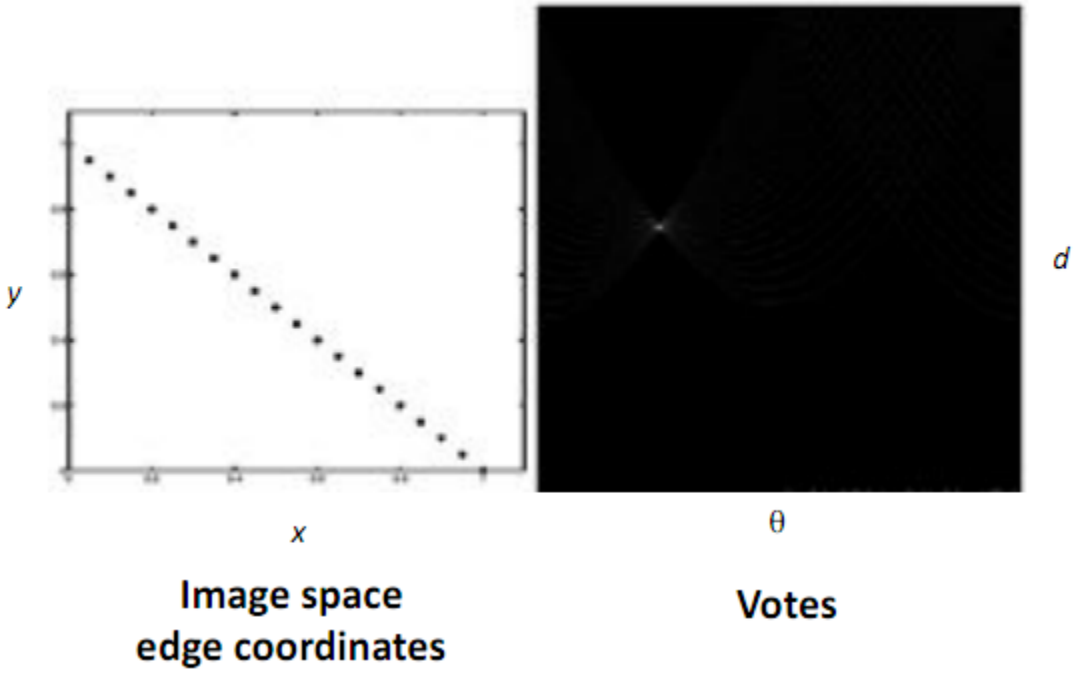
d = længden mellem (0,0) og punktet

$\theta$=vinklen mellem x aksen og normalen fra linjen

Denne metode konverterer direkte fra vores originale billede til det polære rum. Hough-transformation med polære koordinater giver dette resultat hvor der er valgt tre tilfældige punkter.



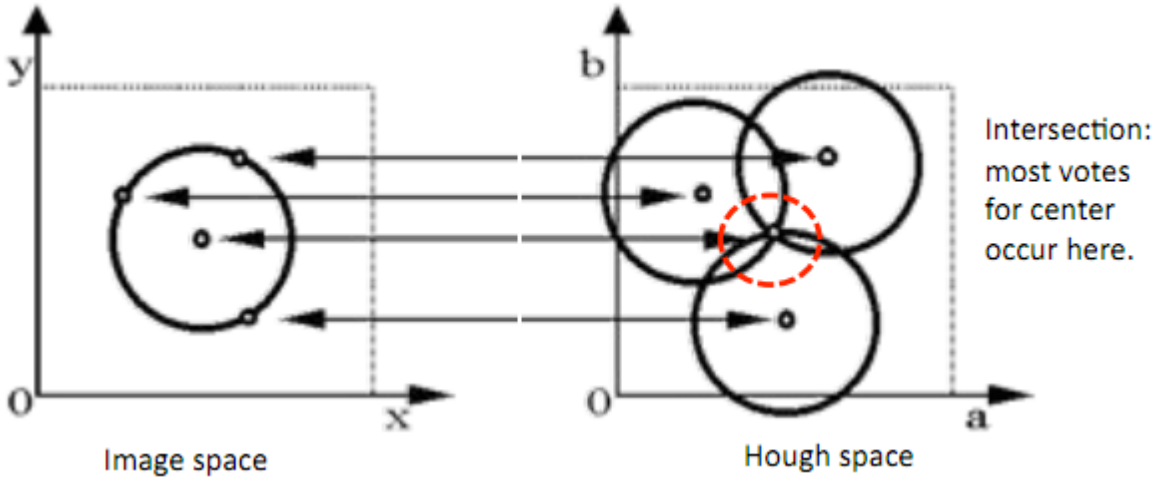

Hvis vi vælger punkter på en lige linje får man et resultat hvor man tydeligt kan se et sted hvor linjerne overlapper; der optræder altså en linje i billedet. Jo mere lyst et punkt er, jo flere linjer overlapper det punkt og jo flere der overlapper et punkt, jo mere sandsynligt er det, at der er en linje.




**Cirkulær Hough-transformation**

Hough-transformation kan også bruges til at finde cirkler med, problemet er bare at vi ikke længere kender formen på det vi leder efter. Det er selvfølgelig en cirkel, men en skiftende

størrelse af cirklen tvinger os til at gentage hele proceduren for hver mulig størrelse af cirkler vi leder efter. Dette skyldes, at vi leder efter de steder hvor der er overlap mellem potentielle cirkler, og hvis der er skiftende størrelser af cirkler skal alle størrelser prøves.
Hvis der er nok cirkler der overlapper i samme punkt kan vi regne med det er centrum af en cirkel i billedet.




[4]

## Implementering af cirkulær Hough transformation

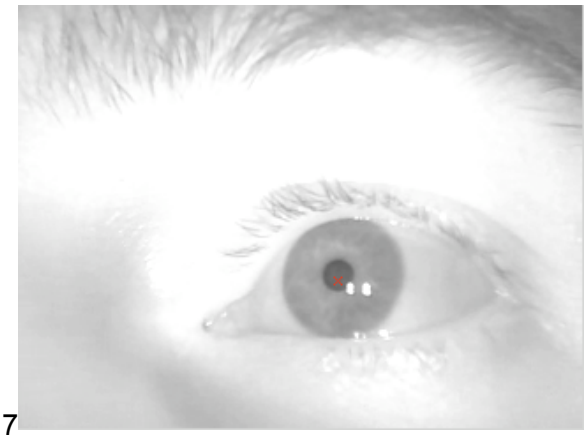
Implementeringen af den cirkulære Hough transformation udføres på et gråtonebillede som en MxN matrix, hvor output består af en tredimensionel akkumulator-matrix, M x N x R, hvor R er det højeste estimat af den iris-radius som der laves cirkulær Hough transformation med. Som input gives således henholdsvis højeste og laveste estimat for radius af iris.

Algoritmen i detaljer:

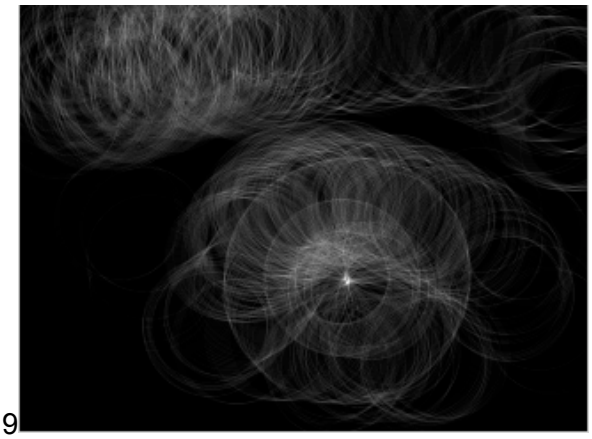
1. Signifikante edge-pixels findes ved udføre convolution på input-billedet med en edge detection-operator; Sobel-operatoren[5] i vores tilfælde.
2. Akkumulator-matrixen initialiseres med nulværdier
3. Der itereres over alle edge-pixels fundet i (1)
    a. For hvert radiusestimat (givet som input) og for hver retning i enhedscirklen (opdelt i intervaller) udregnes et punkt (a,b) i billedet som kunne ligge på cirkelperiferien af en cirkel som har centrum i den nuværende edge-pixel og som har den nuværende radius.
    b. Akkumulator-matrixen tælles 1 op på den position der svarer det udregnede punkt (a,b) samt radius r.

## Test af implementering

7: Originalbillede med iris-midtpunkt markeret med rødt kryds
8: Midtpunkt fundet med Hough-transformation (den lille hvide prik til højre for midten)
9: Det resulterende billede efter Hough-transformation. Det hvideste område i midten af de to store cirkler er midtpunktet, altså det samme som i (8)

7 

8 

9 

Ovenstående Hough-transformation er udført på billede (7) med en radiusestimering for iris på 35 pixels. På (7) ses desuden et rødt kryds, som markerer midten af iris. For at finde dette midtpunkt fandt vi punktet i billede (9) med den højeste intensitetsværdi, da dette er punktet som i algoritmen har fået flest stemmer/votes. På billede (8) er al støjen fra (9) fjernet, og kun det fundne midtpunkt er synligt (hvidt). Dette punkt svarer til det røde kryds i (7).

Som udgangspunkt fungerede Hough-transformation ganske godt til at finde iris-lokationen i billedet. Dog oplevede vi, at der opstod problemer med metoden hvis to punkter i billedet havde fået lige mange votes - der havde vi besvær med at bestemme, hvilket punkt der var det korrekte midtpunkt (selvom vi naturligvis kunne se det på billederne). Desuden oplevede vi også problemer med radiusestimatet - det samme estimat fandt ikke effektivt midten for alle billeder. Overordnet set havde vi succes med radiusværdier i intervallet [30;40], men havde altså besvær med at gøre estimatet konsistent for en sekvens af billeder, eksempelvis når øjet trak sig længere væk fra kameraet eller lignende, således at iris' andel af billedet blev mindre eller større.

# Ellipse fitting og gradients

For at foretage ellipse fitting må vi først finde billedets gradienter, så vi har et udgangspunkt.

## Gradienter

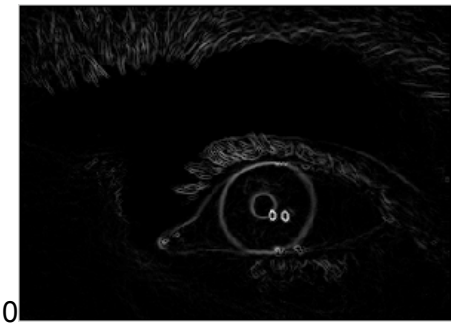
Gradients i billeder er interessante, idet de hjælper os med at finde kanter i et givet billede. En gradient i et billede kan beskrives som en forskel i intensitetsværdi mellem to på hinanden følgende pixels på x-aksen, y-aksen eller både x- og y-aksen. En gradient magnitude beskriver således hvor stor denne forskel er. Ud over en magnitude har en gradient også en retning. Gradient'en af et billede defineres således:

$$\nabla f = \left[\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right]$$ [6]

Et billedes gradient er altså som nævnt udtryk for en differentiering af pixelværdier på henholdsvis x- og y-aksen. Dette hjælper os med at finde forskelle i intensitetsværdier i billedet, men dette fortæller ikke hvilke forskelle der er signifikante - det vil sige de forskelle der indikerer, at der er en kant i billedet. For at finde ud af hvor forskellene er størst, beregner vi da billedets gradient magnitude, som er defineret således:

$$\|\nabla f\| = \sqrt{\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2}$$ [6]

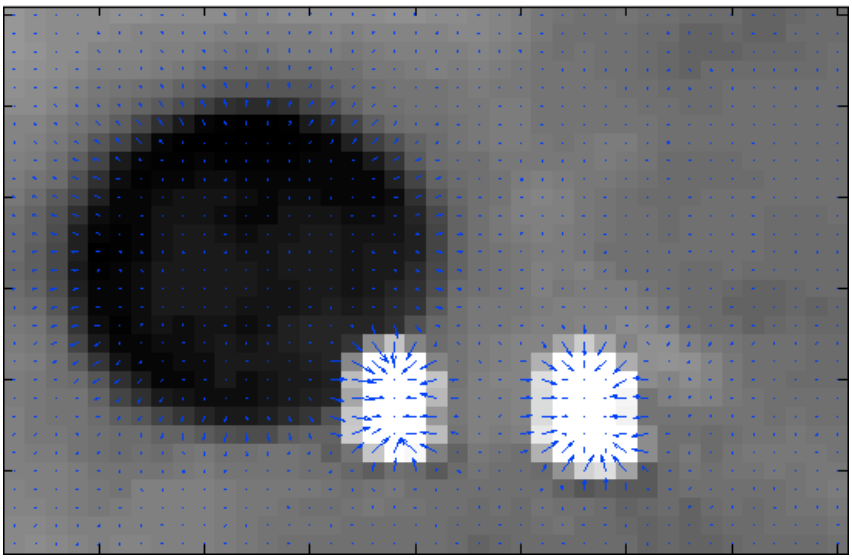
Billedets gradient magnitude beskriver afstanden mellem to gradientværdier som et ikke-negativt tal, der indikerer hvor stærk gradienten er. Jo højere gradient magnitude, des stærkere gradient. Dette betyder at de pixelværdier for hvilke den tilsvarende gradient magnitude er stor sandsynligvis er en del af en kant i billedet. Nedenfor ses et gradient magnitude-billede:

10 

Af billedet (10) kan vi udlede, at de steder hvor intensitetsværdierne er højest er der højst sandsynligt en signifikant kant i billedet; på billedet kan vi se de vigtigste kanter i øjet, hvilket understreger dette.
En gradient har som nævnt også en retning. Denne defineres som følgende:

$$\theta = \tan^{-1}\left(\frac{\partial f}{\partial y} / \frac{\partial f}{\partial x}\right)$$ [6]

En gradient's retning defineres altså som retningen af normalen til kanten, og står derfor vinkelret på kanten. Dette betyder, at gradient'en altid peger i den retning, hvor forskellen i intensitetsværdierne er størst.
Men ovenstående definitioner in mente betragter vi nu følgende billede, som er et close-up af pupil og glints:



11

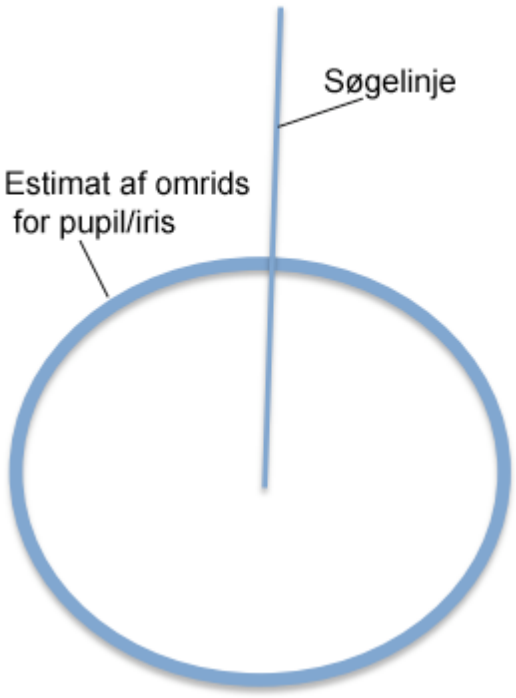
På billedet (11) viser de blå pile de enkelte gradients. Vi ser at gradienterne peger i den retning hvor forskellen i intensitetsværdi er størst, hvilket især er tydeligt omkring glints, hvor alle gradienter peger ind mod disse. Desuden ser vi, at jo større forskel der er mellem de på hinanden følgende pixels, des længere er pilene (eller des mere af deres længde ser vi). Dette stemmer altså fint overens med den ovenstående teori.

## Ellipse fitting

Den overordnede idé i ellipse fitting er at placere en ellipse bedst muligt på en række givne punkter. Selve det at placere ellipsen har dog ikke været en del af vores opgave, da en sådan funktion blev udleveret, så denne del af ellipse fitting vil vi ikke gå i dybden med. Dog vil vi uddybe den indledende del, omfattende det at finde de punkter som ellipsen skal placeres over.

I forhold til eyetracking bruges ellipse fitting til at finde omridset af henholdsvis pupiller og iris i øjebilleder. For at kunne placere en ellipse på enten pupillen eller iris skal vi kende følgende: centrum, radius samt et gradient magnitude-billede af det øje, som ellipsen skal fittes på. Idéen er, at hvis vi kender disse værdier, har vi et estimat for hvor pupillens eller iris' kant er. Ved at trække en søgelinje ud fra disse punkter kan vi ved at tjekke intensitetsværdierne for alle punkterne på linjen finde det punkt, hvor intensitetsværdien er størst. Denne idé er illustreret nedenfor.

Fra ovenstående teori om gradienter ved vi, at når vi trækker denne søgelinje på et gradient magnitude-billede, vil de punkter hvor der er en kant have den største intensitetsværdi. Når vi har et estimat for pupillens eller iris' omrids og trækker søgelinjer ud fra dette estimat, kan vi finde det reelle omrids ved at finde den højeste værdi på hver søgelinje (én for hvert punkt). Dette kræver naturligvis, at søgelinjen er lang nok til, at punktet med den højeste intensitetsværdi ligger på denne linje. Som udgangspunkt fastsættes længden på linjen ud fra et estimat, fx udregnet i forhold til radius af iris.




I det følgende antages det, at centrum af pupil/iris er fundet gennem Hough-transformation eller lignende, at radius er et estimat samt at billedets gradient magnitude er beregnet som beskrevet i afsnittet om gradients. Med disse forudsætninger kan vi finde punkterne som ellipsen skal placeres over ved at følge disse tre trin:

**1. Definér punkter**

Indledningsvis defineres et ønsket antal punkter, fx med MATLABs `linspace`-funktion. Når først man har disse, ganges den givne radius på. Da har vi følgende cirkel (parameteriseret i punktet *t*):

$$(x(t), y(t)) = R*(cos(t),sin(t))$$

Herefter oversættes punkterne således at de har det givne centrum som midtpunkt.

**2. Find normaler**

Når vi kender punkterne, skal vi kende normalen for hvert punkt til den cirkel som punkterne danner. Normalerne findes på følgende vis (med cirkel parameteriseret i *t*):

$$(x'(t), y'(t)) = R * (-sin(t), cos(t))$$

Normalen til cirklen i hvert punkt skal vi bruge til at bestemme søgelinjens retning. Når vi kender denne retning, kan vi gange den ønskede længe for søgelinjen på denne; lad os kalde denne værdi (retning gange længde) for *s*. Herefter kan vi finde søgelinjens endepunkter ved henholdsvis at subtrahere og addere *s* fra udgangspunktet *t*.

**3. Find højeste intensitetsværdi**

Når vi ved hvor søgelinjen befinder sig, samt hvilken retning den har, kan vi kopiere de pixels ud af billedet som ligger på søgelinjen, fx ved brug af MATLABs `improfile`-funktion. Ud fra denne mængde pixels kan vi finde den pixel som har den største intensitetsværdi. Når vi ved hvilken pixel det drejer sig om, gemmer vi dennes position i billedet - for denne skal inkluderes i den mængde af punkter, som den endelige ellipse skal placeres over.
Dette gøres for hvert punkt/søgelinje, indtil man har hele mængden af punkter som ellipsen skal placeres over.

## Implementering af ellipse fitting

Vi kunne desværre ikke få MATLAB til at makke ret da vi skulle implementere ellipse fitting, og vi har derfor ingen konkret implementering af dette. Vi har dog en delvis implementering som følger de tre ovenfor beskrevne trin. Af samme grund har vi heller intet testdata som vi kan vise frem.

## Refleksion over ellipse fitting

På trods af de manglende testdata, er der alligevel nogle relevante pointer, som er værd at reflektere over. Idet at sammenhængen mellem kanter og intensitetsværdier er så stor en del af princippet bag ellipse fitting, kan man forestille sig at dette også kan føre til visse fejl. I situationer hvor man skal finde omridset af iris, og en eller flere søgelinjer indeholder dele af en glint, kan man komme til at fejltolke glintens (formodentligt) høje intensitetsværdi som en mulig kant, selvom dette slet ikke er tilfældet. Dette kunne dog muligvis omgås ved simpelthen at ekskludere sådanne søgelinjer fra datasættet.
Et andet problem med ellipse fitting kan tænkes at opstå ved detektion af pupil eller iris på billeder, hvor øjet eksempelvis er halvt lukket, så disse kun er delvist synlige. Da vil det være stort set umuligt at få brugbar data ud af den halvdel af øjet der er lukket, og man vil derfor kun have data til en halv ellipse. Dette kan tænkes at skabe problemer, især hvis med medregner det første problem med glints, i og med at glints ofte er at finde i den nederste halvdel af øjet. Hvis data omkring glinten også ekskluderes, er datasættet pludselig så lille, at det kan være svært at fitte en ellipse til dette.
Overordnet set virker ellipse fitting dog til at være et effektivt, og relativt simpelt, værktøj til at identificere iris og pupil, som vil virke i de fleste tilfælde, modregnet de scenarier som lige blev beskrevet.

# Segmentering af øje-billedet i clusters

Evnen til at kunne opdele et billede i forskellige segmenter blev allerede brugt under pupil og glint-identificering ved hjælp af matlab funktionen `bwlabels`. Der er dog andre fremgangsmåder man kan benytte, og en af dem er k-means clustering.
K-means clustering kræver at man vælger at lave k antal clusters i et gråtone-billede, og algoritmen grupperer derefter pixels der ligner hinanden mest i disse 1 til k clusters.

Mere specifikt, så gør algoritmen det følgende:

1. Initialisér k tilfældige cluster-midtpunkter
2. Ud fra de valgte cluster-midtpunkter, bestem hvilke punkter der skal være i hvert cluster
    - For hvert punkt, find og grupper punktet med det nærmeste cluster-midtpunkt
3. Sæt cluster i-midtpunktet til at være gennemsnittet af alle punkter i cluster i
4. Hvis cluster i-midtpunktet ændrer sig, gå til trin 2.

K-means clustering har den fordel at den altid vil finde en løsning, og at den er relativt hurtigt. Den vil dog ikke altid give det samme resultat, pga. de tilfældigt udvalgte start-midtpunkter.

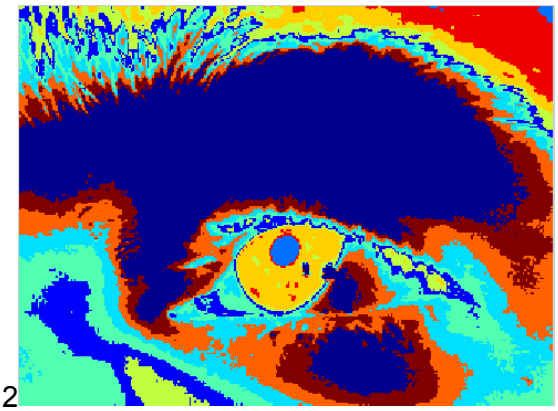
En ting der er essentielt ved brug af k-means clustering er at vælge et godt antal clusters. Vælger man for mange clusters mister man fordelen ved grupperingen, nemlig at billedet bliver simplificeret, og vælger man for få bliver billedet billedet forvredet og ubrugeligt (og vil indeholde for lidt information).

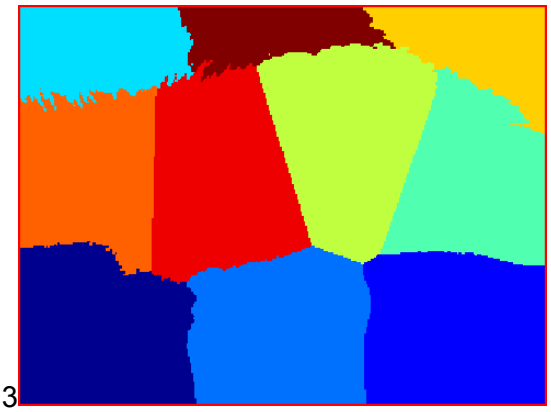
Vores mål med k-means clustering var at få en god opdelingen af billedet, der stadig indeholdt information om iris, pupil samt glint.

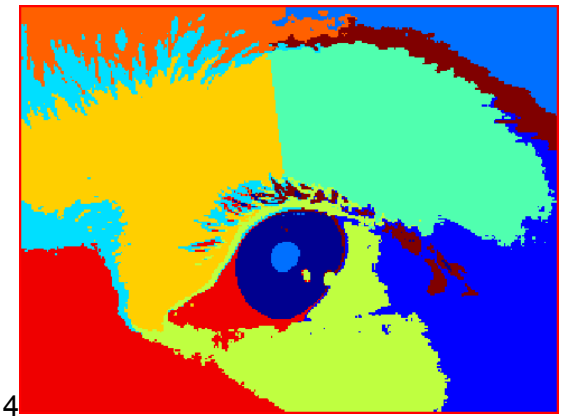
Vi lavede først en MATLAB-funktion der kun benyttede intensiteten på billedet til at dele billedet op i k klusters. Ved kun at se på intensiteten går alle de samme intensiteter igen i rigtig mange dele af billedet, da det eneste kriterie er intensiteten. Se billede (12).

For at løse dette problem, og sørge for at clusters bliver grovere opdelt, kan man bruge pixel-positionen som en vægt på intensiteten. Dette får mindre små dele af en gruppering til at foretrække at blive en del af en anden mere nærtliggende gruppering. Dette har dog en ret fatal konsekvens, nemlig at pixel-positionen kan få en alt for høj vægt, og selve intensiteten bliver irrelevant. Resultatet af dette kan ses i billede (13).

Løsningen til dette er ret simpel, nemlig bare at sørge for at pixel-positionens indflydelse har en lavere vægt end intensiteten. Dette opnås ved at dividere x- og y-værdien med en faktor. At vælge en ordentlig faktor kan være svært, men vi fik gode resultater på vores video-sekvens ved at dividere med 4. Se billede (14) for resultatet.

12 

13 



14

Efter at have prøvet flere forskellige antal clusters, og forskellige antal faktorer for punktets position i forhold til intensiteten kom vi frem til at det følgende virkede bedst på vores videosekvens:

- Clusters: 10
- Faktor: 4

# Konklusion

At kunne udtrække information fra billeder kan være utrolig brugbart og kan bruges til automatisering af et utal af opgaver som f. eks. maskinlæring i sammenhæng med medicinske billeder, til robotter der skal kunne reagere på deres miljø og meget mere.

I denne opgave lavede vi en eye-tracker der fungerede godt efter hensigten.
Man kan identificere elementer i øjet på mange måder, og vi har f. eks. benyttet en kombination af thresholding og segmentering, Hough-transformation og k-means clustering alle med gode resultater, men dog også alle med deres mangler.

**Problemer vi ikke fik løst**

Vi nåede ikke at komme til et punkt hvor iris, pupil og glint rent faktiskt blev isoleret fra billedet ved brug af clustering, men vi mener at i sammenhæng med de tidligere metoder at dette burde være overkommeligt.
Ved at give etiketter til alle sammenbundne pixels kan de adskilte clusters med samme

intensitet yderligere blive skilt ad. Ved derefter at benytte thresholding til at finde pupillen og dens midtpunkt, burde vi derefter kunne identificere den i vores cluster-billede, og finde det omkring-liggende iris. Efter at have fundet iris burde det være simpelt at identificere glints da de skal ligge indenfor en hvis afstand af iris, og man kan også sige at de skal være indenfor området.
Kvaliteten af vores cluster-billede kunne højst sandsynligt også forbedres ved brug af diverse teknikker som smoothing (ved brug af fx gaussian blur), morphologisk transformation af billedet ved hjælp af metoder som opening eller closing, histogramudligning og muligvis også ved brug af thresholding.

Generelt har vi også mange antagelser om størrelsen af øjet og dets komponenter, samt lysforhold, og de er alle blevet indstillet til en bestemt videosekvens. Hvis vores implementation skal kunne fungere på flere forskellige billedesekvenser, skal disse værdier højst sandsynligt estimeres ud fra den gennemsnitlige lysstyrke i billedet og de forventede størrelser burde nok både være afhængige af hinanden, og potentielt også af lysstyrken (da fx pupiller kan ændre størrelse alt efter lysforhold).

# Referenceliste


1. Thomas B Moeslund. *Image and video processing*, 2nd ed. Aalborg University press, 2009
2. Klassificering: Rasmus R. Paulsen and Thomas B. Moeslund. *Introduction to Medical Image Analysis.* DTU Informatice, 2011
3. Rafael C. Gonzalez and Richard E. Woods. *Digital Image Processing.* Addison-Wesley Longman, Publishing Co., Inc., Boston, MA, USA, 2nd edition, 1992
4. http://www.itu.dk/courses/SIGB/F2011/Slides/Segmentation.pdf
5. http://en.wikipedia.org/wiki/Sobel_operator
6. http://www.itu.dk/courses/SIGB/F2011/Slides/Binary&Edges.pdf