1 Analyse

1.1 Indledning

Formålet med analysedelen af projektet var at få en indledende indsigt i emnet. Det indebar at få skabt et overblik over hvad der skullealaves, hvorfor det skulleal les, og hvilke krav der er til de ting som skulle lavesal lavdover er det også vigtigt at nævne at det der bliver diskuteret overvejende er koncepter eller teori, med enkelte eksempler som kan øge forståelsen for meningen. Konkret design og implementering vil blive gennemgået senere, i design og implementeringsdelene af rapporten.

1.2 Systemoversigt

Blokdiagram: Userskærm, testskærm, IR-LED, kamera. Blokdiagram2: Overordnet blokdiagram over software. (Figur 2, geometric approach to eyetracking)

Billede af opstilling.

Dette er opstillingen som projektet tager udgangspunkt i. Argumentationen for valget af denne opstilling kan ses i indledningen til projektet (Ref indledning).

1.3 Funktionalitetskrav

1.3.1 Use case eksempel - Start måling

I figur **??** ses et eksempel på en use case for systemet.

1.4 Overvejelser, Funktionelle krav

1.4.1 Real-time eye-tracking

Dette krav er stillet af udbyderen af projektet, og kan betragtes som et af de mest fundamentale krav i projektet. Projektet udspringer fra et tidligere projekt, hvor billedbehandlingen af kameradata foregik offline. Det skal optimeres i en tilstrækkelig grad til at kunne køre i realtime i stedet for, med en framerate på 100 billeder per sekund.

1.4.2 Kalibrering

For at systemet skal fungere korrekt skal der først foretages en kalibrering. Målinger kan foretages uden kalibrering, men det vil ikke være muligt at over-

Figur 1: Use case 3

Sektion	**Kommentar**
Mål	Programmet påbegynder real-time eye-tracking
Initiering	Initieres af aktøren *bruger*
Aktører	Aktøren *bruger* og aktøren *kamera*
Antal samtidige forekomster	1
Startbetingelser	Computerprogrammet skal være opstartet, *kamera* skal være tændt, programmet skal være kalibreret.
Slutresultat – succes	Programmet har påbegyndt real-time eye-tracking
Slutresultat – undtagelse	Programmet alarmerer *bruger* at der ikke er foretaget kalibrering
Normal forløb	1. *Bruger* klikker på knappen "Start". 2. Programmet starter ny måling. 3. Visuel feedback på GUI viser at måling er i gang.
Undtagelsesforløb	Programmet kan ikke starte ny måling. Programmet melder at kalibrering ikke er foretaget.

sætte de resulterende data til et sæt skærmkoordinater. Denne kalibrering er derfor essentiel for systemet.

1.4.3 Output

Resultater fra målinger skal gemmes i en log-fil tilgængelig til brugeren. Dette krav stammer fra et ønske fra udbyder om at have adgang til data efter behandling. I design-delen af rapporten kan ses en protokol, der beskriver hvordan data skal gemmes i log-filen.

1.4.4 Modularisering

essentielt for projektet. Programmet skal skrives som en API (Application Programming Interface), altså et software-interface til anden software. Denne API skal designes således at bruger-interfacet, håndteringen af diverse filer, logning af data, kommunikation med hardware og så videre, kan implementeres som sit eget stykke software. Denne software kan så være basis for algoritmer der overholder API'ens regler. rænseflade, tilstrækkelig funktionalitet til at dække formodede behov, samt et design der gør det let at udvide applikationen på et senere tidspunkt.

De resterende funktionelle krav er blevet uddybet ved hjælp af Use Cases -REF-.

1.5 Overvejelser, Ikke-funktionelle krav

1.5.1 Fejlmargin

1.5.2 Real-time

1.5.3 Kodesprog

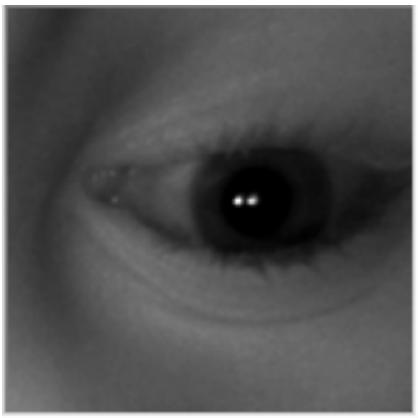
Det er blevet aftalt med projektudbyder at prototypen skal udarbejdes med en C++ backend "Algoritmer"og Python frontend "Grafisk bruger interface". C++ er valgt som backend fordi det er et relativt hurtigt kodesprog, og fordi gruppen har tidligere erfaring med C++. Python blev foreslået til gruppen af projektvejleder, og idet Python har udvidelser der er opbygget i C/C++ "OpenCV og Numpy"var det en oplagt mulighed at anvende Python til front end delen af prototypen.

1.6 Starburst-algoritmen

I det følgende vil algoritmen som projektet tager udgangspunkt i blive gennemgået. En sammenligning af gruppens prototype og denne algoritme vil

Image.png 

Figur 2: Enkelt billede fra video optagelse

Eye.png 

Figur 3

senere blive vist i implementeringsdelen af rapporten.

Bemærk at der udelukkende beskrives hvad Starburst-algoritmen gør, og at der IKKE bliver beskrevet andre problematikker i følgende afsnit "Eksempelvist lokalisering af øjne, lokalisering af pupilmidtpunkt".

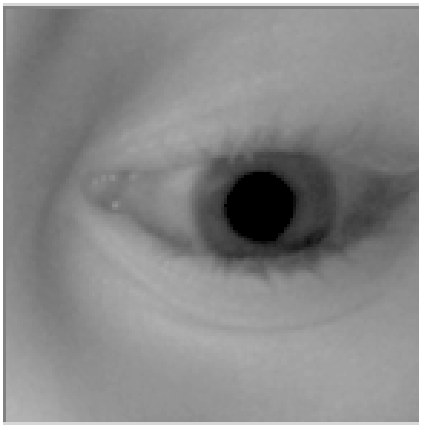
Algoritmen kan opdeles i følgende generelle punkter "ref starburst":

Input: Billede. Output: blik koordinater Procedure: Detekterer hornhinde reflektioner. Lokaliserer hornhinde reflektioner. Fjerner hornhinde reflektioner Iterativ detektion af pupil kantpunkter. RANSAC "Random Sample Consensus"til at finde en ellipse der passer til punkterne. "Kalibrering anvendes til at bruge homografisk mapning"

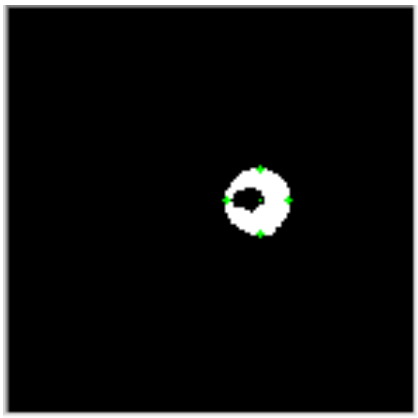
Bemærk at procestiden er opgivet som procentdel af samlet procestid, og at værdierne kun omfatter processen selv, og ikke medregner den tid der bruges på metoder der kaldes undervejs. Procestiden for disse metodekald står ud for de enkelte metoder i stedet.

Eye, Post Brightness.png 

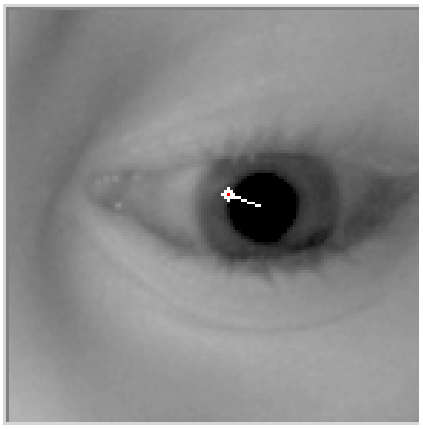
Figur 4

Eye, Reflection removed.png 

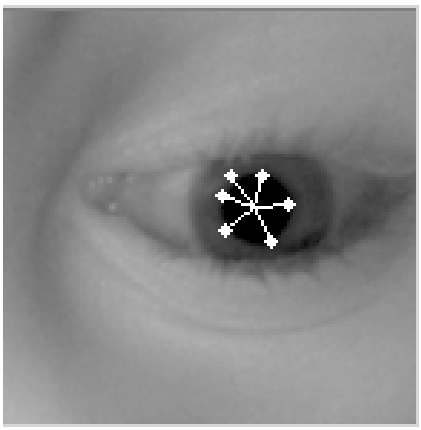
Figur 5



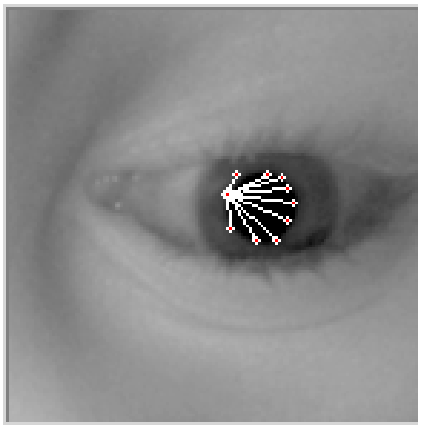
Figur 6

First single.png 

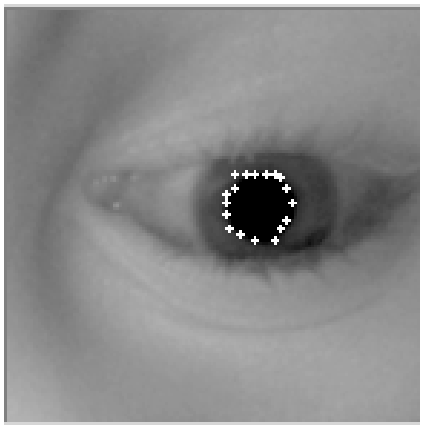
Figur 7

First.png 

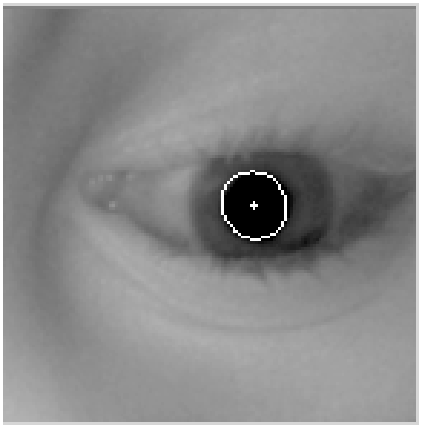
Figur 8

Second single.png 

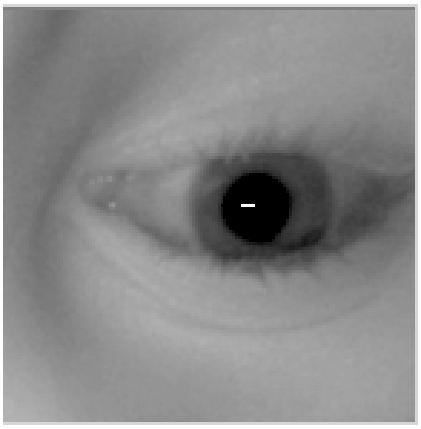
Figur 9

Second.png 

Figur 10

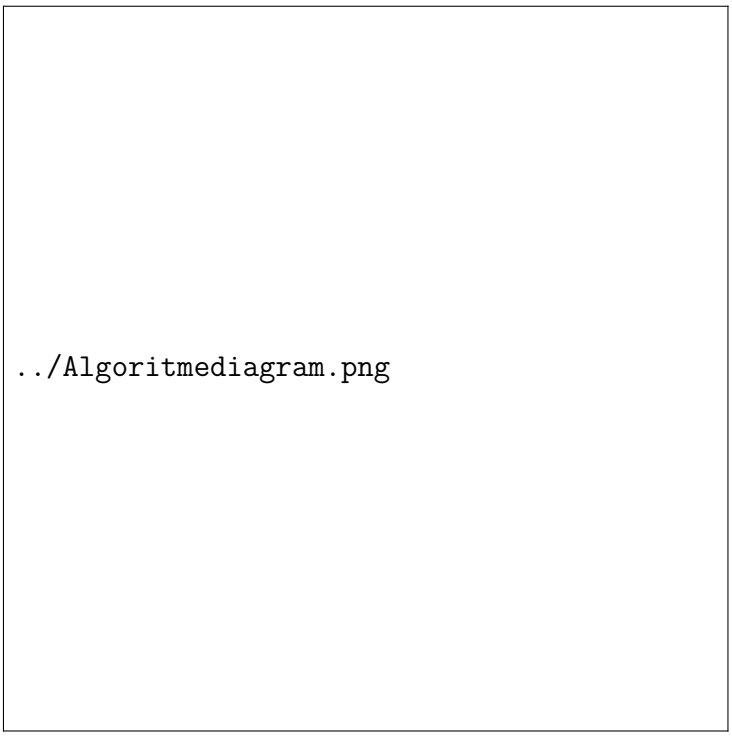
Ellipse.png 

Figur 11

Gaze.png 

Figur 12

1.7 Algoritme Oversigt



Figur 13: Systemdiagram for Real-time eye-tracking

1.7.1 Calculate pupil and gaze

Main funktion - 0.17

1.7.2 Locate corneal reflection

Finder reflektionspunker - 0.3

1.7.3 Starburst pupil contour detection

Starburst algoritme - 0.63

1.7.4 Locate Edge Points

Find pupil kant punkter - 44.71

1.7.5 Fit ellipse ransac

Tilpas en ellipse til punkterne - 11.7

1.8 Fokuspunkter

I det følgende underafsnit vil der kort blive beskrevet hvilke dele af systemet der med fordel kan fokuseres på. Dette er baseret på antal gange rutinen bliver kørt, samt hvor lang tid det tager for processen at blive færdig. Begrundelsen for dette er at det højst sandsynligt er lettere at øge performance med en betydelig del hvis de dele der tager længst tid først bliver optimeret.

1.8.1 Kantdetektion

Den mest tidskrævende del af algoritmen er selve starburst-delen, altså den del hvor pupilkantpunkter findes. Udover at optimere koden ved at køre i C, er der også mulighed for at justere variabler for at få processen til at tage kortere tid. Dette vil dog højst sandsynligt ske på bekostning af resultaternes nøjagtighed/præcision, eller systemets overordnede stabilitet. Der vil altså formentligt være behov for en balancering af performance overfor kvalitet. Denne balancering vil foregå iterativt i løbet af implementeringsfasen af projektet.

1.8.2 Ellipse Tilpasning

Den næstmest tidskrævende del af algoritmen er ellipsetilpasningen. Her vil det igen være muligt at justere på variabler, for at balancere performance og kvalitet. Derudover skal der ses nærmere på undtagelsestilstande hvor algoritmen gentages mange gange, RANSAC iterations = 10000. Hvis disse undtagelsestilstande opsluger meget tid og sker ofte, kunne det have en markant effekt på performance for systemet.

1.8.3 Kegleparametre til Ellipseparametre

En mindre betydelig, men dog stadig mærkbar del af algoritmen er den del som oversætter kegleparametre til ellipseparametre. Da denne del af algoritmen kun udgør omkring 6 procent af den samlede procestid, er denne blot nævnt som en mulig kandidat for optimering hvis de primære dele begynder at være sammenlignelige i procestid.

1.8.4 Kamera Input

En sidste ting der med fordel kan fokuseres på at forbedre er indlæsning af data fra kameraet. Idet der ikke sker meget kompression bør det meste af arbejdet bestå af overførsler i harddisken, men hvis det viser sig at være for tungt kan vi forsøge at gøre noget.

1.9 OpenEyes og Siboska

Tidligt i projektforløbet blev der udleveret kildekode til gruppen, kildekode som prototypen overvejende er inspireret af "Siboskas kode". Derudover er der også kildekoden som Siboska implementeringen har taget sit udspring i. Væsentlige forskelle og egenskaber vil kort blive gennemgået, og en mere grundig gennemgang kan findes i analysedelen af dokumentationen "Ref".

hvilket også indebærer processeringstid for de enkelte subrutiner. I forlængelse af dette vil begrundelserne for krav der har med algoritmen at gøre også blive givet i en relevant kontekst. I det følgende vil simuleringer i forbindelse med starburst algoritmen blive gennemgået. Disse simuleringer er blevet udført i et forsøg på at danne en bedre indsigt i sammenhængen imellem performance og resultater. Idet en overgang til en anden platform end MATlab formentligt ikke forøger performance tilstrækkeligt, skal der i stedet undersøges hvorvidt det er muligt at tilpasse forskellige variabler til at opnå en kortere processeringstid, alt imens resultaterne forbliver gode.

1.10 Simulering

Simuleringerne er foretaget med det kode og videodata som Daniel Sibozka har udleveret til gruppen. I det første afsnit vises resultaterne, og i andet afsnit beskrives hvilke ændringer af variabler der afprøves, og derefter vises resultaterne af disse ændringer.

1.10.1 Extract gaze vector from video

Processeringstid, CPU = 2.4 GHz

Figur over resultater.

1.10.2 Variabel Ændringer

Antal Rays Antal RANSAC iterationer

1.10.3 Resultater

Figur

1.11 Diskussion