

# Rapport

**Bachelorprojektet:  
Real-time eye-tracking  
Projektnummer: 15017**



**29/05/2015**

**Studerende: Søren Vøgg Krabbe Lyster (SVL) 10920,  
Martin Degn Kristensen (MDK) 10441**

**Studieretning: Elektro**

**Vejleder: Preben Kidmose**

---

# Indhold

<b>1</b>	<b>Prolog</b>	<b>4</b>
1.1	Indledning . . . . .	4
1.2	Krav til bachelorprojekt . . . . .	4
1.3	Ordforklaring . . . . .	4
1.4	Projektformulering . . . . .	5
1.5	OpenEyes og Siboska . . . . .	5
<b>2</b>	<b>Metode</b>	<b>6</b>
2.1	Metoder . . . . .	6
2.2	Planlægning . . . . .	8
2.3	Diskussion . . . . .	8
<b>3</b>	<b>Værktøjer</b>	<b>9</b>
3.1	Udviklingsmiljø . . . . .	9
3.2	OpenCV . . . . .	9
3.3	Python . . . . .	9
3.3.1	Python vs. Java og C++ . . . . .	9
3.3.2	Relevante programbiblioteker . . . . .	10
3.4	Diskussion . . . . .	10
<b>4</b>	<b>Kravspecifikation</b>	<b>11</b>
4.1	Problemstillinger . . . . .	11
4.2	Funktionelle krav . . . . .	11
4.2.1	Real-time eye-tracking . . . . .	12
4.2.2	Kalibrering . . . . .	12
4.2.3	Output . . . . .	13
4.2.4	Modularisering . . . . .	13
4.2.5	Use-cases . . . . .	13
4.3	Ikke funktionelle krav . . . . .	13
4.4	Performance-evaluering . . . . .	14
4.5	Diskussion . . . . .	14
<b>5</b>	<b>Analyse</b>	<b>15</b>
5.1	Indledning . . . . .	15
5.2	Systemoversigt . . . . .	15
5.3	Funktionalitetskrav . . . . .	15
5.3.1	Use case eksempel - Start måling . . . . .	15
5.4	Overvejelser, Funktionelle krav . . . . .	17
5.4.1	Real-time eye-tracking . . . . .	17

5.4.2	Kalibrering . . . . .	17
5.4.3	Output . . . . .	17
5.5	Overvejelser, Ikke-funktionelle krav . . . . .	17
5.5.1	Fejlmargin . . . . .	17
5.5.2	Real-time . . . . .	17
5.5.3	Modularisering . . . . .	17
5.5.4	Kodesprog . . . . .	18
5.6	Starburst-algoritmen . . . . .	18
5.7	Algoritme Oversigt . . . . .	23
5.7.1	Calculate pupil and gaze . . . . .	23
5.7.2	Locate corneal reflection . . . . .	23
5.7.3	Starburst pupil contour detection . . . . .	23
5.7.4	Locate Edge Points . . . . .	24
5.7.5	Fit ellipse ransac . . . . .	24
5.8	Fokuspunkter . . . . .	24
5.8.1	Kantdetektion . . . . .	24
5.8.2	Ellipse Tilpasning . . . . .	25
5.8.3	Kegleparametre til Ellipseparametre . . . . .	25
5.8.4	Kamera Input . . . . .	25
5.9	OpenEyes og Siboska . . . . .	25
5.10	Simulering . . . . .	26
5.10.1	Extract gaze vector from video . . . . .	26
5.10.2	Variabel Ændringer . . . . .	26
5.10.3	Resultater . . . . .	26
5.11	Diskussion . . . . .	26
<b>6</b>	<b>Design</b>	<b>27</b>
6.1	Indledning . . . . .	27
6.2	Modularisering . . . . .	27
6.3	Applikationsarkitektur . . . . .	27
6.3.1	UserInterface . . . . .	29
6.3.2	SessionHandler . . . . .	29
6.3.3	LogHandler . . . . .	30
6.3.4	VideoCapture . . . . .	30
6.3.5	EyeTrackingHandler . . . . .	30
6.3.6	Interaktion mellem klasser . . . . .	30
6.4	Algoritme-arkitektur . . . . .	31
6.5	Bruger-interface . . . . .	32
6.6	Diskussion . . . . .	32

<b>7</b>	<b>Implementering</b>	<b>34</b>
7.1	Indledning . . . . .	34
7.2	Bruger-interface og applikation . . . . .	34
7.2.1	UIHandler . . . . .	34
7.2.2	SessionHandler . . . . .	35
7.2.3	LogHandler . . . . .	36
7.2.4	EyeTrackingHandler . . . . .	36
7.2.5	VideoCapture . . . . .	38
7.2.6	Kommunikation imellem applikationens klasser . . . . .	38
7.3	ETAlgorithm . . . . .	38
7.4	Starburst-algoritmen . . . . .	39
7.5	Optimering . . . . .	39
7.5.1	Fremtidige optimeringsmuligheder . . . . .	39
7.6	Diskussion . . . . .	39
<b>8</b>	<b>Test</b>	<b>40</b>
8.1	Indledning . . . . .	40
8.2	Deltest . . . . .	40
8.3	Integrationstest . . . . .	41
8.4	Accepttest . . . . .	41
8.5	Performance-evaluering . . . . .	41
8.6	Diskussion af testresultater . . . . .	41
<b>9</b>	<b>Konklusion</b>	<b>42</b>
<b>A</b>	<b>Appendiks</b>	<b>43</b>

# 1 Prolog

## 1.1 Indledning

## 1.2 Krav til bachelorprojekt

Aarhus Universitets kursuskatalog beskriver følgende krav til bachelorprojekt på Ingeniørhøjskolen:

*Ingeniøruddannelsen afsluttes med et bachelorprojekt, som skal dokumentere den studerendes evne til at anvende ingeniørmæssige teorier og metoder inden for et fagligt afgrænset emne.*

*Når kurset er afsluttet, forventes den studerende at kunne:*

- *Anvende videnskabelige forskningsresultater og indsamlet teknisk viden til løsning af tekniske problemstillinger*
- *Udvikle nye løsninger*
- *Tilegne sig og vurdere ny viden inden for relevante ingeniørmæssige områder*
- *Udføre ingeniørmæssige rutinearbejde indenfor fagområdet*
- *Kommunikere resultater af et projekt skriftligt til fagfolk såvel som kunder*
- *Præsentere resultater af et projekt mundtligt og ved hjælp af forskellige audiovisuelle kommunikationsværktøjer*
- *Integrere sociale, økonomiske, miljømæssige og arbejdsmiljømæssige konsekvenser i en løsningsmodel.*

## 1.3 Ordforklaring

**Gaze Vector:** Dette term bruges om den vektor som beskriver hvor en person ser hen. Denne vektor er et af de resultater der ønskes fra eye-tracking systemet.

**Session:** Dette term bliver brugt om en real-time eye-tracking måling foretaget af i programmet. Sessionen beskriver det enkelte målingsforløb fra start til slut. Til hver session vil der være tilknyttet separate præference- og logfiler.

**Data-fil:** Dette er den fil der vil blive tilknyttet til hver session. Filen vil forventes at indeholde al relevant data i forbindelse med real-time eye-tracking måling. Filen vil blive kreeret af programmet og vil være tilgængelig til brugeren.

**Testperson:** Det er denne person der foretages real-time eye-tracking på. Personen er sammen med brugeren en del af kalibreringsrutinen. Denne person ses ikke som aktør i systemet.

**Trigger:** For at kunne holde en synkronisering imellem real-time eye-tracking softwaren og andre målinger (EEG), er der givet et trigger-signal. Dette signal består af en ændring af lys-intensitet.

## 1.4 Projektformulering

Fra projektoplægget er følgende beskrivelse givet:

*Eye-tracking is widely used in different research areas as for example in psychology, in analysis of man-computer interactions, and in behavioural studies. Eye-tracking is also used in computer gaming. At ASE eye-tracking is used for research both in the biomedical lab and in the vision lab. The system as it is now is based on off-line Matlab processing of camera data. The purpose of this project is to design and implement a software system for real-time acquisition of the eye-movements. The basic principle of the eye-tracking system is based on reflection from two IR-LED's from the eyes. By identifying the reflections, and doing some geometrical computations, it is possible to determine the users gaze vector. The project will build on an existing hardware setup comprising a camera, IR-sources and a computer screen. The primary objective of the project is to design and build a flexible software system that can process the camera data in real-time, and output the gaze vector. The image processing will build on an existing Matlab implementation. The system must be designed such that there is a flexible interface to the image processing part in order to facilitate new image processing algorithms to be tested in the system. For students with particular interest in image processing, a secondary objective could be to further develop on the image processing algorithms.*

## 1.5 OpenEyes og Siboska

Kort beskrivelse af de to algoritme-implementeringer projektet har taget udgangspunkt i.

## 2 Metode

### 2.1 Metoder

Til dette projekt er der blevet benyttet V-modellen som udviklingsmetode [1], og UML (Unified Modelling Language) [2] til design af softwarearkitekturen. V-modellen tilbyder en overskuelig tilgang til softwareudvikling i et projekt hvor en række krav er fastsat som udgangspunkt for udviklingen. Ved hjælp af UML er de forskellige krav omskrevet til use cases, og derfra er der udviklet design-diagrammer for softwarearkitekturen.

V-modellen følger i dette projekt følgende stadier fra analyse til implementering:

- Opstilling af krav

Ud fra projektoplægget og samtaler med projektudbyderen kan der formuleres en række konkrete krav til systemet. Disse krav kan deles op i funktionelle og ikke funktionelle krav. De funktionelle krav er direkte tilknyttet systems funktionalitet, hvor de ikke funktionelle krav i dette projekt omhandler ting som for eksempel præcision og opdateringshastighed. Dette stadie ender ud i færdiggørelsen af en kravspecifikation.

- Analyse

- Design

Ud fra de opstillede use cases kan der udvikles designdiagrammer til systemet. Disse diagrammer er udarbejdet udfra UML, og indeholder derfor en tydelig overgang fra use case. Sekvensdiagrammer er designet for hver use case, og resulterer i en række ønskede klasser. Klasserne defineres mere specifikt i klassediagrammer. Programmets flow-struktur designes således at der er en fornuftig kommunikationsvej fra det grafiske bruger-interface, til selve eye-tracking algoritmen. Et udkast til det grafiske bruger-interface bliver skitseret således at al funktionalitet fra de opstillede krav kan opfyldes. Et flow-diagram for Starburst algoritmen designes. Dette stadie ender ud i et softwarearkitektur-dokument.

- Implementering

ORDORDORD

Efter implementering af koden følger modellen en række test-stadier:

- Deltest

Deltest tester den individuelle klasses funktionalitet op imod softwarearkitekturen. Herved kan eventuelle fejl og mangler fanges tidligt.

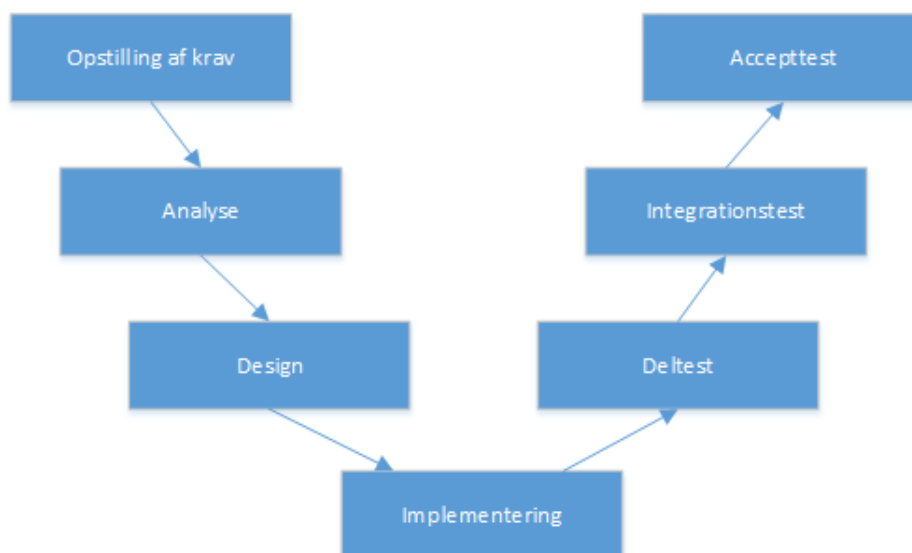
- Integrationstest

Her testes de forskellige klassers interaktion med hinanden. Der undersøges om klasserne sender de rigtige data, og om der bliver gjort de rigtige funktionskald.

- Accepttest

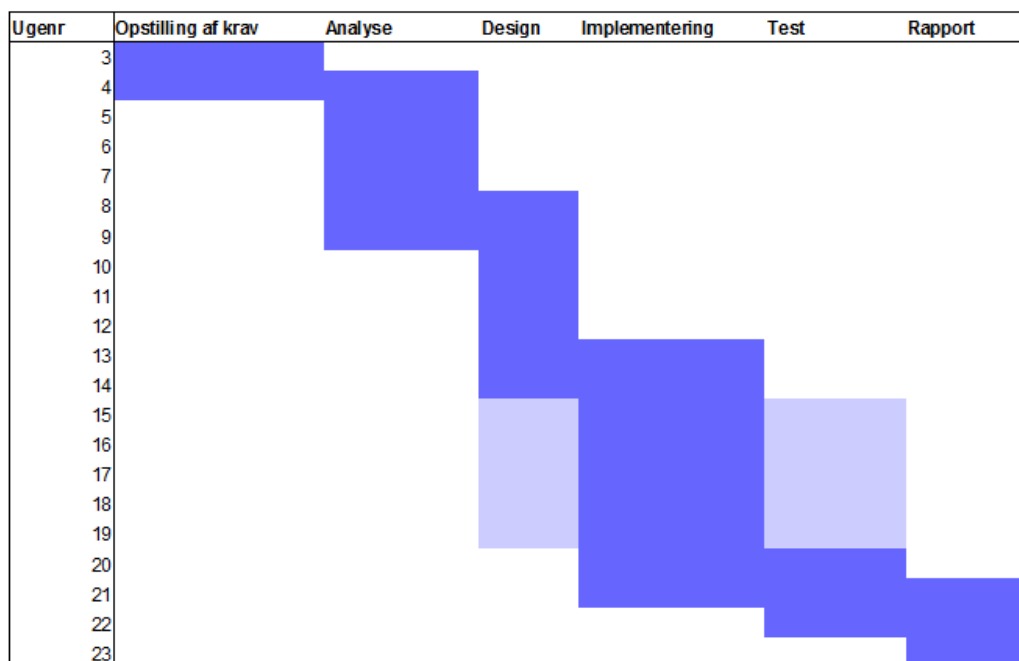
Accepttesten er den endelige test af systemet. Her undersøges om systemet i sin helhed lever op til de forskellige krav formuleret i kravspecifikationen.

V-modellens fleksibilitet tillader at man ved hvert teststadium kan gå tilbage og tilpasse analyse, design og implementering. V-modellen giver derfor mulighed for at opstille en række krav, omskrive dem til softwarearkitektur, implementere arkitekturen, foretage tests på den implementerede kode, og derefter gå tilbage rette hvad der kunne være nødvendigt. I et projekt af denne størrelse, hvor det forventes at der skal tilegnes ny viden, giver denne model derfor god mulighed for tidligt at støde på eventuelle problemer ved implementeringen, og derefter søge ny viden der kan benyttes til at opnå de opstillede krav.



Figur 1: V-modellen





Figur 2: Tidsplan for projektet. Mørkeblå farve viser den konkrete tidsplan. lyseblå farve viser perioder hvor det forventedes at de forskellige stadier ville overlappe hinanden.

## 2.2 Planlægning

Bacheloropgaven er sat til 20 ECTS point, hvilket ifølge Aarhus Universitet svarer til 560 timer. Bachelorprojektets forløb strækker sig fra januar til slut maj, og er anslået til at vare 20 uger. Derudover har der været et forprojekt i december 2014. Der er derfor bestemt et gennemsnitligt arbejdspress på 30 timer om ugen for hele forløbet. I samarbejde med vejleder er der aftalt en times møde hver mandag. Desuden har det været bestræbt at mødes i projektgruppen hver hverdag, med forbehold for sygdom og således. Internt i projektgruppen har det været aftalt af føre personlig logbog, hvori overvejelser og noter kunne føres.

Tidsplan for projektet (se figur 2) er blevet udført i løbet af forprojektet med henblik på projektstruktur som angivet af v-modellen. Denne tidsplan er blevet revideret og opdateret til vejledermødet hver mandag.

## 2.3 Diskussion

## 3 Værktøjer

### 3.1 Udviklingsmiljø

Eye-tracking programmet er blevet udviklet delvist i Pythons IDE Idle, og delvist i Microsofts Visual Studio. Al kode er versionsstyret ved hjælp af versionsstyringssystemet Git. Arbejde med UML, samt design af diverse diagrammer og figurer, er udført i Microsofts Visio.

### 3.2 OpenCV

OpenCV er et programbibliotek frit tilgængeligt under BSD licensen. Programbiblioteket tilbyder optimerede funktioner designet med henblik på hurtige udregninger med fokus på realtids-programmering. Desuden giver OpenCV en række funktioner for kommunikation med video-hardware, samt behandling af videofiler. OpenCV's funktioner er implementeret i kodesproget C++, men understøtter implementering igennem både Python, Java og Matlab, og kører på Windows, Linux, OS X, Android m.m. [3].

### 3.3 Python

Python er et høj-niveau kodesprog der understøtter objekt-orienteret programmering, miksede høj-niveau datatyper, automatisk memory management, og et ekstensivt programbibliotek. Python understøtter derfor hurtig implementering i forhold til både Java og C++. Desuden understøtter Python også wrapping med C/C++. Dette er basis for arbejde med OpenCV.

#### 3.3.1 Python vs. Java og C++

Fordele [4]:

- 3-5 gange kortere end Java og 5-10 gange kortere end C++.
- Automatisk garbage collector. Ingen fokus på frigørelse af resourcer.
- Miksede høj-niveau datatyper.

Ulemper:

- Langsommere end både Java og C++, da datatyper skal ikke nødvendigvis er defineret før runtime.

### 3.3.2 Relevante programbiblioteker

Nogle af Python's benyttede programbiblioteker har haft stor indflydelse på projektet, og er derfor relevante at nævne:

**Numpy** er et højt optimeret programbibliotek til arbejde med numeriske operationer og tilbyder derfor en god understøttelse af array- og matricematematik, samt en ekstensiv række høj-niveau matematiske funktioner [5]. Numpy er derfor anvendt af OpenCV i Python.

**Tkinter** er en Python-binding til GUI-værktøjet Tk. Dette interface giver mulighed for en nem implementering af det grafiske bruger-interface, og er frit tilgængeligt under en Python-licens [6].

## 3.4 Diskussion

Det valgte udviklingsmiljø har givet et sæt stærke værktøjer til design, implementering, debugging og profiling. Versionsstyring har været de naturlige valg når det kommer til programmering. Brugen af Numpy har lagt grund for en nem omskrivning af Siboskas MATLAB-kode, da de fleste matematiske funktioner også findes i Numpy.

## 4 Kravspecifikation

### 4.1 Problemstillinger

Der ønskes udviklet et system som kan indsamle videodata fra et kamera og derefter anvende dataen til at bestemme hvor en forsøgsperson kigger hen på en specifik skærm. Systemet skal derudover videregive denne information til brugeren via koordinater samt en graf der repræsenterer den skærm forsøgspersonen ser på.

Før dataopsamling skal en indledende kalibrering af systemet gennemføres. Dette gøres ved at et gitter med specifikke punkter indlæses på forsøgspersonsskærmen. Derefter bedes forsøgspersonen fiksere på specifikke punkter på skærmen, og sammenhængen imellem de målte punkter og de kendte punkter kan anvendes til at finde en homografisk mapning. Efter denne kalibrering kan systemet anvendes.

Systemet udvikles med henblik på en standard anvendelsesmåde, med mulighed for brugerdefinerede anvendelsesmåder. Standardanvendelsen omhandler at vælge en sti og et filnavn, hvorefter dataopsamling umiddelbart begynder. Under dataopsamlingen vil gazevectoren løbende blive præsenteret for brugeren på brugerskærmen. Når brugeren er færdig kan opsamlingen stoppes, og dataopsamlingen gemmes i den tidligere valgte fil. Bemærk at den algoritme der anvendes til behandling af data her er forudbestemt. (Hvis brugeren ønsker at bruge en anden algoritme kan denne indlæses. Den kan også indskrives direkte i GUI'en, og derefter gemmes. Formålet med dette er at kunne indrette systemet efter specifikke behov, og hurtigt indhente de opsætninger til fremtidig brug. Eventuelt kan andre variabler indtastes ved systemstart)

I de følgende afsnit fremgår det hvorledes det udviklede system indgår i det samlede system.

### 4.2 Funktionelle krav

Følgende funktionelle krav for systemet er blevet stillet:

1. **Real-time eye-tracking:** Systemet skal kunne foretage real-time eye-tracking.
2. **Kalibrering:** Før måling skal programmet kunne kalibreres.
3. **Output:** Resultater af måling skal ende i en log-fil tilgængelig til brugeren.

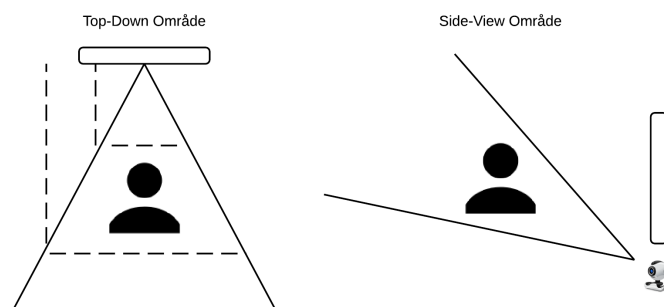
4. **Modularisering:** Det skal være muligt for bruger at ændre/erstatte algoritmen.
5. **Brugertilgang:** Ved hjælp af use case teknikken vil en yderlige række krav blive stillet. Disse vil lægge grundlag for bruger-program-interaktioner. Use-case-kravene er opstillet i afsnit 4.2.5.

#### 4.2.1 Real-time eye-tracking

Systemet skal kunne foretage real-time eye-tracking ved hjælp af en implementeret udgave af Starburst-algoritmen som givet af Siboska og OpenEyes.

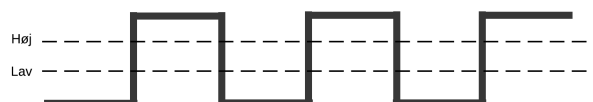
#### 4.2.2 Kalibrering

Specifikke ukendte variabler skal kunne kalibreres ved hjælp af interpolation. Herved skal programmet kunne tilpasses testpersonens fysiske forhold til kameraet.



Figur 3: Kameraets position i forhold til testperson

Derudover skal programmet kunne kalibreres således at der kan findes tærskler (threshold-values) for trigger-niveauet: En værdi når trigger-niveauet går højt, og en værdi når trigger-niveauet går lavt.



Figur 4: Eksempel på tærskelværdier for trigger-signalet

### 4.2.3 Output

For hver igangsat session skal programmet generere en log-fil med følgende data:

- Kommasepareret målingsdata med følgende format:  
*timestamp, x-koordinat, y-koordinat, trigger, fejlmeddelelse*

### 4.2.4 Modularisering

Systemet skal være modulariseret, således at bruger kan foretage ændringer i algoritmen, eller tilføje en hel ny algoritme. Det er derfor nødvendigt med en opdeling af programmet, samt klare regler for input-argumenter og output-værdier i forhold til algoritme-delen. Dette system skal implementeres med Starburst-algoritmen som udgangspunkt.

### 4.2.5 Use-cases

1. Opret session:  
Opretter en session i en fil-sti med nødvendige data-filer.
2. Kalibrering:  
Initierer en række kalibreringer før brug.
3. Start måling:  
Igangsætter måling.
4. Stop måling:  
Afslutter måling.
5. Gem indstillinger:  
Gemmer en fil med brugerens nuværende indstillinger.
6. Indlæs indstillinger:  
Indlæser indstillinger fra gemt fil.
7. Indlæs rå data:  
Indlæser rå data fra tidligere session.

## 4.3 Ikke funktionelle krav

Real-time eye-tracking systemet skal leve op til en række ikke funktionelle krav. Disse krav skal garantere et robust system, der med en hvis præcision skal kunne levere de ønskede data.

- **Fejlmargin:** Systemet skal kunne angive XY-koordinater for øjets fokuspunkt. Disse koordinater må have en afvigelse på  $<2^\circ$  [7].
- **Real-time:** Systemet skal kunne angive XY-koordinater med en frekvens bestemt af kameraets frame-rate.
- **Kamera:** Skal kunne levere video-data real-time til en computer. Systemet bliver udviklet til kamera af typen Basler ACA640-100gc GigE med opløsningen 658 x 492 pixels og maksimum framerate på 100Hz.
- Følgende krav er ikke krav til systemet, men krav til testpersonens fysiske forhold til kameraet. Dette er relevant når der foretages eye-tracking.  
**Afstand og skærm:** Systemet bliver udviklet med en afstand fra kamera til testperson på 60cm. På samme afstand fra testpersonen er der placeret en skærm. Denne skærm har størrelsen 26 tommer.

Typen af kamera og afstand til testperson kommer fra tidligere forsøg [8].

## 4.4 Performance-evaluering

Systemet kan efter implementering og test evalueres efter performance. Følgende punkter ønskes evalueret:

- **Miljø:** Algoritmens evne til at tilpasse sig forskellige afstande til forsøgspersonen, forsøgspersonens bevægelser, samt forskellige lysstyrker.
- **Robusthed:** Systemets evne til at håndtere fejl under måling.
- **Brugervenlighed:** Systemets evne til at håndtere fejl i det grafiske bruger-interface. Simplicitet af det grafiske bruger-interface.

## 4.5 Diskussion

De formulerede krav medfører en afgrænsning af projektet, og ligger basis for arbejdet med v-modellen.

## 5 Analyse

### 5.1 Indledning

Formålet med analysedelen af projektet var at få en indledende indsigt i emnet. Det indebar at få skabt et overblik over hvad der skulle laves, hvorfor det skulle laves, og hvilke krav der er til de ting som skulle laves. Størstedelen af tiden i denne fase af projektet blev brugt på at forstå den udleverede kode. En væsentlig del af tiden blev også brugt på at undersøge hvordan ændringer kunne påvirke resultater og performance, samt hvilke steder disse ændringer bedst kunne foretages. Det er også vigtigt at nævne at det der bliver gennemgået i dette afsnit overvejende er koncepter eller teori, med tilhørende eksempler rettet mod at fremme forståelsen. Konkret design og implementering vil blive gennemgået senere, i design og implementeringsdelene af rapporten.

### 5.2 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).

### 5.3 Funktionalitetskrav

I løbet af analysen blev der løbende fremstillet en kravspecifikation, i samarbejde med projektudbyderen. En del af denne kravspecifikation bestod af Use Cases, en række situationer man kunne forestille sig ville opstå, når man anvender systemet. Formålet med dem er at dække en række situationer der kunne opstå, hvad enten systemet gør som tiltænkt eller hvis det af forskellige årsager fejler. Dette gør det lettere at designe et konkret system, idet man har en lang række hændelser som systemet skal kunne håndtere. I rapporten er der valgt kun at vise en enkelt Use Case, de resterende Use Cases kan ses i dokumentationen "Ref".

#### 5.3.1 Use case eksempel - Start måling

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



Figur 5: Use case 3

Sektion	Kommentar
Mål	Programmet påbegynder real-time eye-tracking
Initiering	Initieres af aktøren <i>bruger</i>
Aktører	Aktøren <i>bruger</i> og aktøren <i>kamera</i>
Antal samtidige forekomster	1
Startbetingelser	Computerprogrammet skal være opstartet, <i>kamera</i> skal være tændt, programmet skal være kalibreret.
Slutresultat – succes	Programmet har påbegyndt real-time eye-tracking
Slutresultat – undtagelse	Programmet alarmerer <i>bruger</i> at der ikke er foretaget kalibrering
Normal forløb	<ol style="list-style-type: none"> <li>1. <i>Bruger</i> klikker på knappen "Start".</li> <li>2. Programmet starter ny måling.</li> <li>3. Visuel feedback på GUI viser at måling er i gang.</li> </ol>
Undtagelsesforløb	Programmet kan ikke starte ny måling. Programmet melder at kalibrering ikke er foretaget.

## 5.4 Overvejelser, Funktionelle krav

### 5.4.1 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.

### 5.4.2 Kalibrering

For at systemet kan fungere korrekt skal der først foretages en kalibrering. Målinger kan foretages uden kalibrering, men det vil ikke være muligt at oversætte de resulterende data til et sæt skærmskoordinater. Denne kalibrering er derfor essentiel for systemet.

### 5.4.3 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.

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

## 5.5 Overvejelser, Ikke-funktionelle krav

### 5.5.1 Fejlmargin

### 5.5.2 Real-time

### 5.5.3 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.



Figur 6: Enkelt billede fra video optagelse

#### 5.5.4 Kodesprog

Det er blevet aftalt med projektudbyder at prototypen skal udarbejdes med en C++ backend "Algoritmer" og Python frontend "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.

### 5.6 Starburst-algoritmen

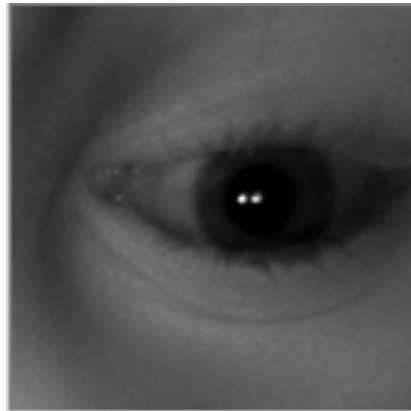
I det følgende vil algoritmen som projektet tager udgangspunkt i blive gennemgået.

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".

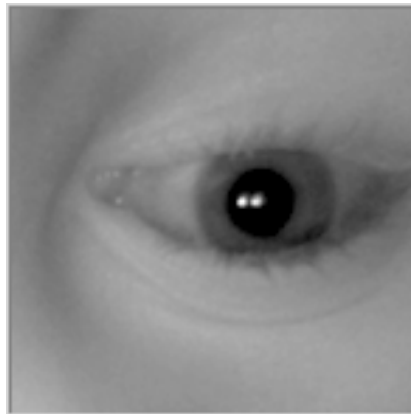
Algoritmen kan opdeles i følgende generelle punkter "ref starburst":

Input: Billede. Output: Blik koordinater  
 Procedure: Detekter hornhinde refleksioner. Lokaliser hornhinde refleksioner. Fjern hornhinde refleksioner  
 Iterativ detektion af pupil kantpunkter. RANSAC "Random Sample Consensus" for at finde en passende ellipse. Anvend kalibrering til at omsætte vektoren imellem midten af hornhinde refleksionerne og midten af pupillen, til skærmskoordinater i pixels.

Gennemgangen tager udgangspunkt i det ovenstående billede, som er det første billede i videoen "Derp". Det første der bliver gjort er at få indsnævret billedet til det der er relevant for algoritmen, altså øjnene. Der findes flere forskellige metoder til dette, eksempelvis kunne man manuelt markere et område som algoritmen skal fokusere på.



Figur 7: Udsnit af øje



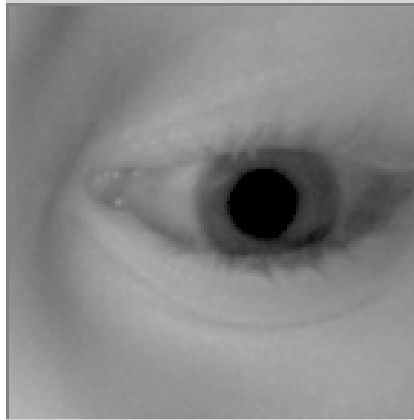
Figur 8: Efter øgning af kontrast og lysstyrke

Når det relevante område af billedet er fundet, kan Starburst algoritmen anvendes. Man kan dog med fordel forsøge at øge kontrasten i billedet, således at differensen imellem pupil og iris øges. Dette gør algoritmen mere robust, da der vil være en mere klar adskillelse af overgangen fra pupil til iris.

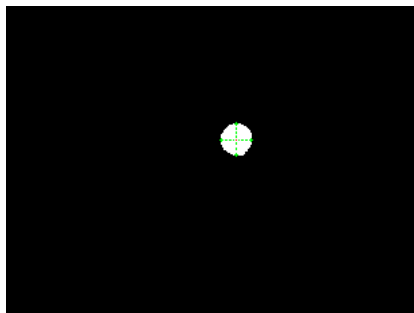
Efterfølgende skal refleksionerne i øjet lokaliseres og fjernes. Alternativt kan man også blot gemme områderne for refleksionerne, og tage højde for dem i sin algoritme i stedet.

Efter fjernelsen af refleksionerne, er det tid til den del af algoritmen som den har fået sit navn fra. Fra et estimeret midtpunkt af pupillen bliver en række "stråler" sendt ud imod kanten af billedet, og når de rammer en tilpas ændring i intensitet stopper de, og det antages at de har ramt overgangen fra pupil til iris.

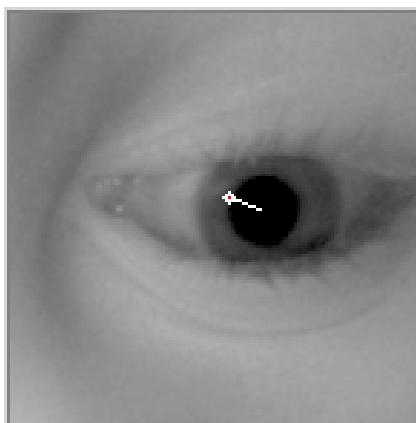
Resultatet af dette er en række punkter som, ifølge hensigten, ligger på kanten af pupillen. Idet der er en risiko for at algoritmen finder punkter som



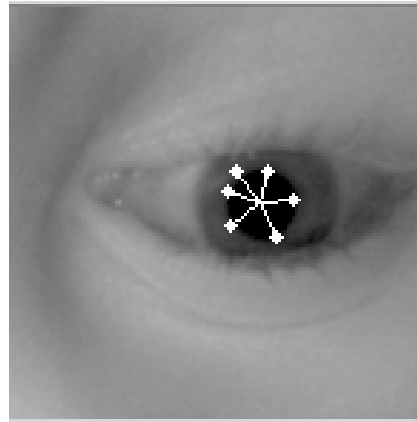
Figur 9: Reflektioner fjernet



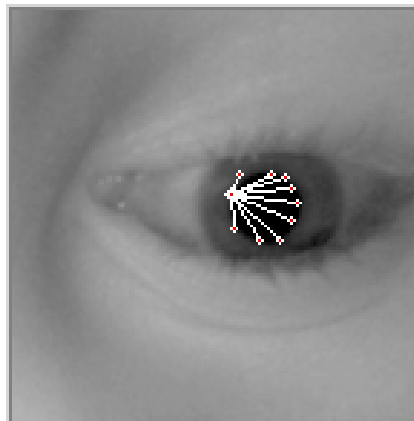
Figur 10: Estimering af pupilmidtpunkt



Figur 11: Starburst, første iteration (Enkelt stråle)



Figur 12: Starburst, første iteration (Resterende stråler)

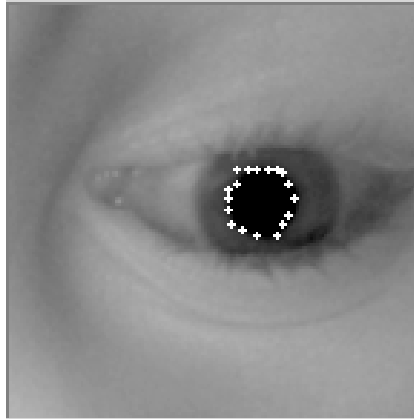


Figur 13: Starburst, anden iteration (Et startpunkt)

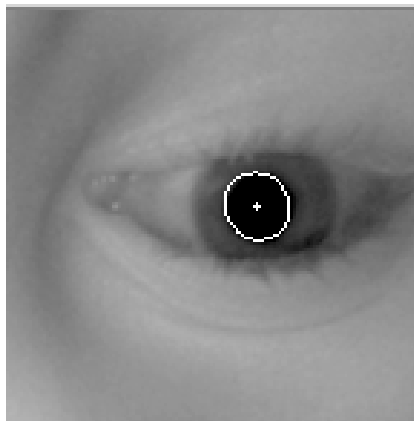
ikke ligger på pupilkanten, gentages Starburst, med hvert fundet punkt som et nyt midtpunkt. Derudover tilpasses vinklerne som strålerne kan sendes ud i, sådan at de søger ind imod det oprindelige startpunkt, + / - et forskelligt antal grader. Dermed får man en række stråler som igen rammer pupilkanten.

Ovenstående billede til venstre illustrerer hvordan strålerne sendes ud fra et tidligere punkt, og billedet til højre viser det endelige resultat af Starburst algoritmen. Ud fra disse punkter kan en ellipse så tilpasses, ved hjælp af RANSAC (Random Sample Consensus). RANSAC fungerer ved at udvælge en vis mængde af de fundne punkter (I vores tilfælde fem), og derefter undersøge hvor godt punkterne passer ind i en model (En model for en ellipse i denne algoritme).

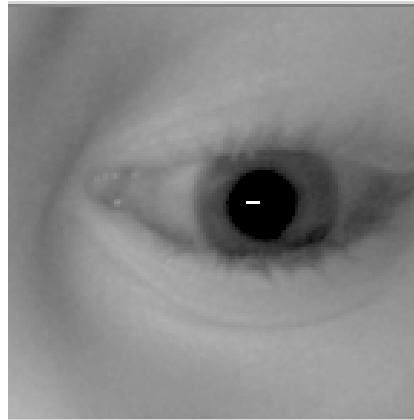
Efter RANSAC har kørt, ender man med et sæt punkter som kan danne en ellipse. Midten af denne ellipse skulle så gerne være midten af pupillen, og



Figur 14: Starburst, anden iteration (Alle Punkter)



Figur 15: Resulterende ellipse



Figur 16: Resulterende gaze vektor

med det punkt, samt midtpunktet imellem de to reflektioner, kan man finde gaze vektoren.

Bemærk at gaze vektoren i sig selv ikke kan fortælle nøjagtigt hvor en person ser hen. Dertil skal man have kalibreret systemet først, ved at gennemgå kalibreringsrutinen som nævnt i "ref". Udfra dette kan man så lave en homografisk mapning ved hjælp af interpolering, som i sidste ende kan bruges til at omsætte gaze vektoren til et sæt skærmkoordinater.

———OLD

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.

## 5.7 Algoritme Oversigt

Det meste af tiden i løbet af analysefasen i projektet blev brugt

### 5.7.1 Calculate pupil and gaze

Main funktion - 0.17

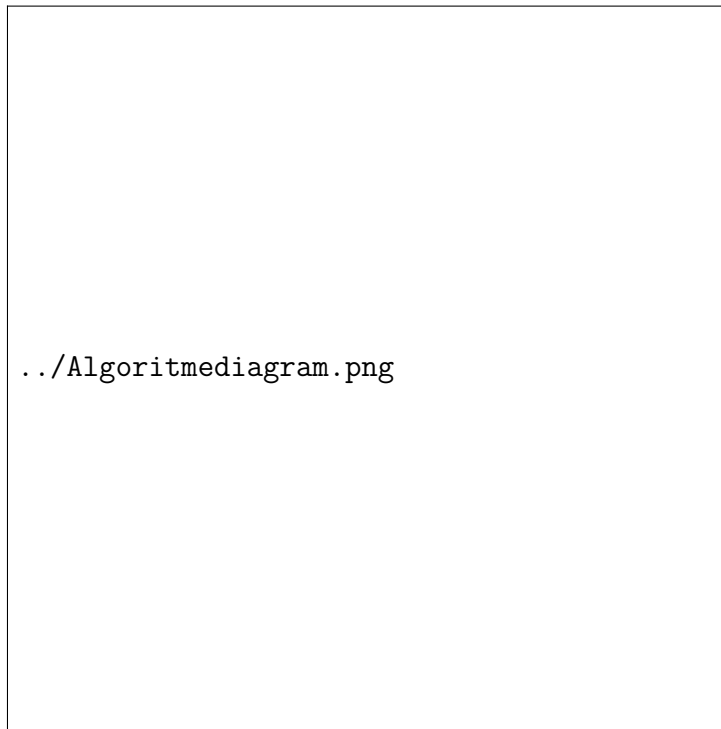
### 5.7.2 Locate corneal reflection

Finder reflektionspunkter - 0.3

### 5.7.3 Starburst pupil contour detection

Starburst algoritme - 0.63





Figur 17: Systemdiagram for Real-time eye-tracking

#### 5.7.4 Locate Edge Points

Find pupil kant punkter - 44.71

#### 5.7.5 Fit ellipse ransac

Tilpas en ellipse til punkterne - 11.7

### 5.8 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.

#### 5.8.1 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.

### 5.8.2 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.

### 5.8.3 Keglparametre 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.

### 5.8.4 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.

## 5.9 OpenEyes og Siboska

Tidligt i projektforsløbet 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. Et eksempel på væsentlige forskelle og egenskaber vil kort blive gennemgået, og resterende eksempler kan findes i analyseafsnittet i dokumentationen "Ref". En sammenligning af gruppens implementering og disse implementeringer vil blive lavet i implementeringsdelen af rapporten.

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.

## 5.10 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.

### 5.10.1 Extract gaze vector from video

Processeringstid, CPU = 2.4 GHz

Figur over resultater.

### 5.10.2 Variabel Ændringer

Antal Rays Antal RANSAC iterationer

### 5.10.3 Resultater

Figur

## 5.11 Diskussion

## 6 Design

### 6.1 Indledning

For at oprette en software arkitektur der overholder de krav stillet i kravspecifikationen, er der gjort brug af UML (Unified Modelling Language) [2]. UML tillader en tilnærmelsesvis direkte omskrivning af krav opstillet som use cases, til UML-diagrammer der skitserer en software arkitektur. Da der i kravspecifikationen er gjort brug af UML til udarbejdelse af de forskellige use cases, har det derfor været muligt af skrive sekvensdiagrammer ud fra hver enkelte use case. Sekvensdiagrammerne giver et klart billede af hvilke funktioner der skal implementeres for at overholde de stillede krav.

Sideløbende med udviklingen af sekvensdiagrammerne er de forskellige klasser blevet forfattet. Følgende afsnit beskriver grundlæggende tanker og argumentation for valget af klasser.

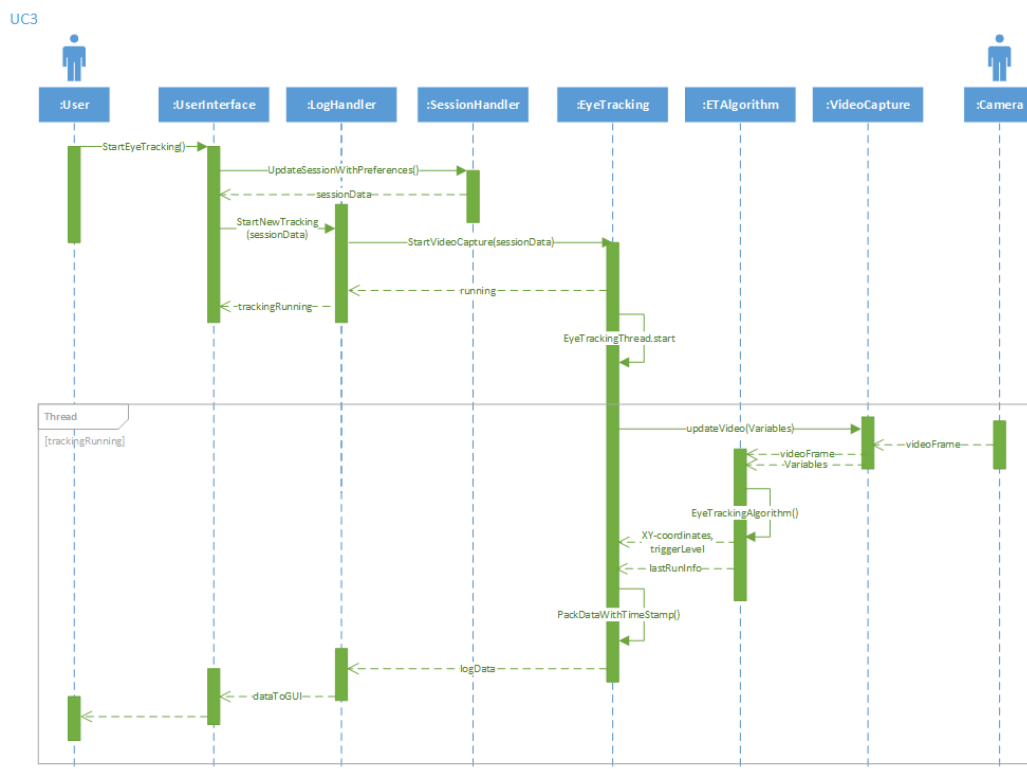
### 6.2 Modularisering

Programmet er ønsket opbygget, så der af en udestående person, er mulighed for fremtidig ændringer i algoritmen. Programmet er derfor struktureret op omkring at have et fastlagt interface med konsistente input/outputs med en selvstændig algoritme-klasse.

### 6.3 Applikationsarkitektur

Ved at benytte UML-tilgangen har de fastlagte use cases kunne omskrives til sekvensdiagrammer. De forskellige klasser er blevet forfattet med henblik på håndtering af de forskellige opgaver (bruger-interface, fil-håndtering, hardware-kommunikation, håndtering af algoritmen, og selve algoritmen), således at der opstår en klar struktur, hvor der ikke er tvivl om hvilken klasse der tilgår hvilken resource/opgave. Følgende eksempel beskriver omskrivningen fra *use case 3 - Start måling* (se figur ??) til *sekvensdiagram 3 - Start måling* (se figur 18).

Som tidligere nævnt varetager hver klasse en opgave med henblik på en af systemets grænseflader. GUI er varetaget af en klasse. De tre genererede filer log, session, og kalibrering er håndteret af tre klasser. Kommunikation med eksternt kamera er håndteret af sin egen klasse. Kommunikation med algoritmen har sin egen klasse. Følgende specificere hver classes opgave, og hvilke krav disse kunne have til implementeringen.



Figur 18: Sekvensdiagram for use case 3 - Start måling

### 6.3.1 UserInterface

Denne klasse foretager al kommunikation med aktøren *bruger* igennem et grafisk bruger-interface (GUI). For at have en konstant opdatering af interfacet, forventes denne klasse at blive afviklet i egen tråd. Denne klasses funktionalitet gives direkte ud fra de forskellige use cases.

### 6.3.2 SessionHandler

Her håndteres alle præferencer tilknyttet til en oprettet session. Denne klasse skal kunne instantieres og benyttes som en datapakke, for hver gang der skal sendes data imellem klasser. Herved kan alle indstillinger foretaget af bruger (enten igennem opsætning af session, eller igennem det grafiske bruger-interface) holdes i et samlet konsistent data-format. Følgende værdier er tilstede i SessionHandler klassen:

- Notes: Noter til bruger af systemet. Disse har ingen funktion for programmet.
- Screen 2 resolution: Angiver opløsningen på skærmen brugt i opstillingen (se figur ??). Opløsningen angives som bredde og højde adskilt af 'x'.
- Log filename: Alternativt navn for log-filen. Er feltet tomt bliver der auto-genereret et navn af formatet YYMMDD-HHMM (år måned dato - time minut).
- Session path: Peger på fil-stien hvor sessionsfilerne (præferencer, log og kalibrering) bliver oprettet. Bliver selv skrevet ind af programmet når man opretter en ny session.
- Using camera source?: Angiver hvorvidt der bliver brugt kamera eller video-fil som input til algoritmen.
- Source: Hvis der er valgt kamera, peget denne værdi på kamera-kilden. Hvis der er valgt video-fil, fortæller denne filstien.
- Recording video?: Angiver hvorvidt det brugte input skal gemmes som en ny video-fil.
- Filepath for recorded data: Angiver hvor den generede video-fil skal gemmes. Hvis intet er angivet gemmes filen i session path.
- Calibration type: Beskriver hvilken type kalibrering der skal benyttes. Man kan således have flere forskellige kalibreringstyper.

- Load calibration file: Giver mulighed for at vælge en allerede oprette kalibreringsfil. Når en ny kalibrering er fuldført, vil denne værdi pege på den nye fil.
- New calibration log filename: Angiver navnet på den oprettede kalibreringsfil. Hvis intet er angivet gemmes filen i session path med navnet calibration.clog.
- V1 til V10: Angiver ti frie variabler til brug i algoritmen. Felterne name bliver ikke benyttet af systemet, og er blot til brugerens nytte. Felterne value kan indeholde alt der opfylder korrekt syntaks for Python og Numpy-biblioteket.

### 6.3.3 LogHandler

Denne klasse håndterer eye-tracking-programmets log-filer (log-fil fra måling samt kalibrerings-log-fil). Data fra eye-tracking bliver her pakket og gemt i en log-fil. Kommunikation af målings-relevant data til UserInterface-klassen forventes foretages af denne klasse.

### 6.3.4 VideoCapture

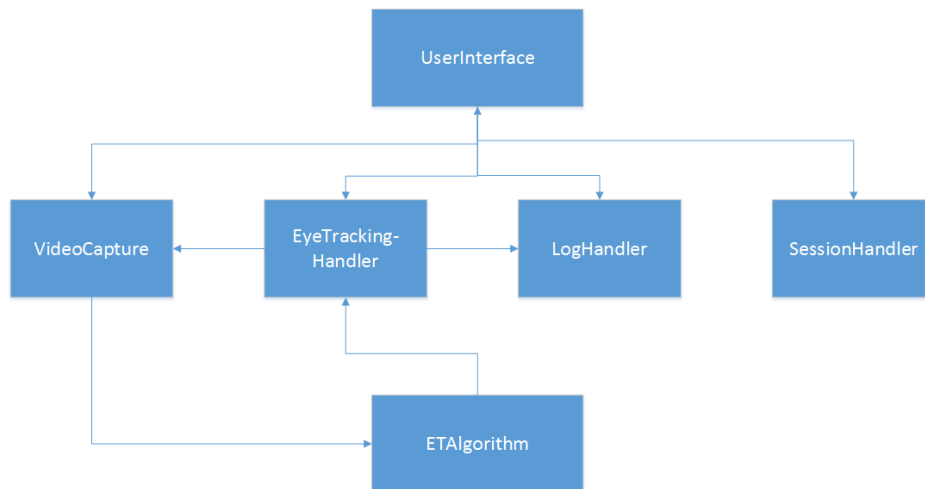
Håndtering af data fra kamera eller video-kilde foregår i denne klasse. Ved hjælp af OpenCV-biblioteket kan der nemt indlæses video-filer, skrives video-filer, samt åbne og lukke for kamera-input. Klassen er også ansvarlig for at læse værdier fra video-kilder såsom opdateringshastighed.

### 6.3.5 EyeTrackingHandler

EyeTrackingHandler håndterer hvornår algoritmen skal køre, og hvilke værdier der skal sendes til algoritmen. For at sikre prioritet til algoritmen, skal denne klasse implementeres med mulighed for trådning (og potentielt multitrådning). Klassen står også for at modtage resultater fra algoritmen, og behandling af disse.

### 6.3.6 Interaktion mellem klasser

Man kan beskrive software-arkitekturen som bestående af tre lag. Det øverste lag består af det grafiske bruger-interface med den tilknyttede klasse UserInterface. Denne blok kan ses som en main-funktion hvorfra programmet bliver afviklet. Derved har brugeren mest mulig kontrol over systemet. Andet lag består af de forskellige 'handler'-klasser: SessionHandler, LogHandler, EyeTrackingHandler og VideoCapture. Sidste lag er eye-tracking-algoritmen i



Figur 19: Klasseinteraktioner

klassen ETAlgorithm. Når brugeren sætter ny måling igang opretter UserInterface en ny instans af SessionHandler-klassen og sender denne data til LogHandler. LogHandler opretter så en log-fil i filstien angivet i SessionHandler, og sender SessionHandler videre til EyeTrackingHandler. EyeTrackingHandler instantierer VideoCapture med korrekt data fra SessionHandler, og opretter derefter en tråd til afvikling af algoritmen i ETAlgorithm. Data til ETAlgorithm bliver pillet ud af SessionData og sendt igennem VideoCapture, hvor der bliver tilføjet et video-frame.

Ved hvert trin kæden igennem bliver der sat en variabel der fortæller de forskellige klasser at en måling kører.

Et diagram over interaktionen ses på figur 19.

## 6.4 Algoritme-arkitektur

Klassen ETAlgorithm er kernen af eye-tracking-systemet. Her foretages alle udregninger med henblik på eye-tracking. Denne klasse ønskes isoleret så meget som muligt fra resten af systemet, således at der altid kan foretages ændringer i klassens interne kode, uden at det påvirker resten af systemet. Konsistente grænseflader for denne klasse er derfor essentielt. ETAlgorithm bliver håndteret af klassen EyeTracking, modtager et videosignal og et sæt af variabler, og returnerer XY-koordinat, trigger-niveau, samt et sæt af variabler.

Da program API'en skal kunne understøtte forskellige typer af eye-tracking-algoritmer, har det været nødvendigt at definere inputs og outputs for algoritmeklasse ETAlgorithm. Input-strukturen består af en video-frame i korrekt



format givet af OpenCV-arkitekturen, kalibreringsdata, og et sæt af 16 frie variabler. Disse variabler har på forhånd ikke noget format, og giver derfor brugeren mulighed for at videregive en række værdier til algoritmen uanset format. Herved kan enhver algoritme med op til 10 inputvariabler implementeres uden nødvendighed for ændringer i API'en.

For eksempel benytter dette projekt sig af en række variabler til at angive: koordinater for sidste pupil-center, afgrænsninger af sidste sæt øjne, hvorvidt Viola Jones skal køres, thresholdværdier, og maks antal RANSAC-iterationer. Disse værdier indeholder både arrays, matricer, enkelt værdier og booleans. De 10 frie inputvariablers startværdier kan indstilles fra GUI, og gemmes i præference-filen.

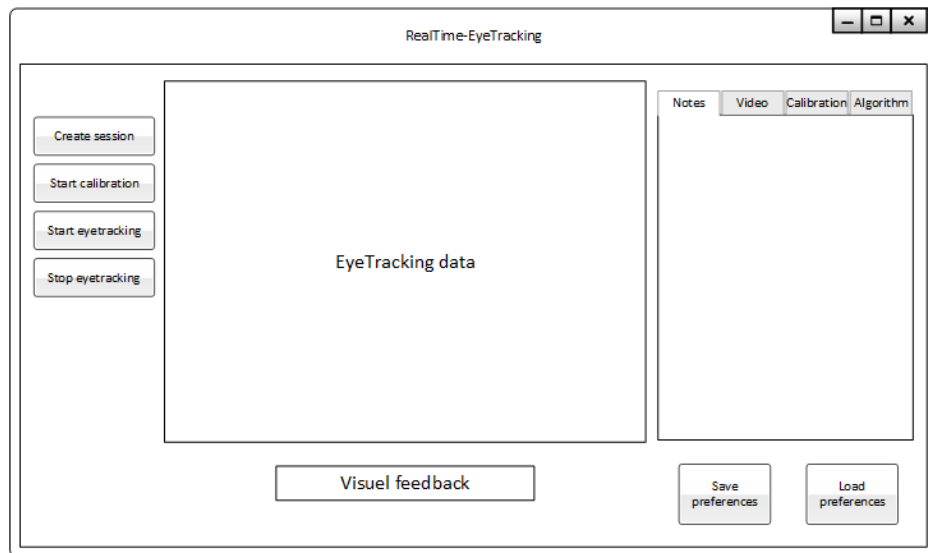
## 6.5 Bruger-interface

Interfacet er designet med henblik på intuitivt brug, så ledes at det ikke er nødvendigt med dybere introduktion til programmet. Forskellige stadier af hvad man kan gøre i interfacet reducerer muligheden for uønskede handlinger. Det er for eksempel ikke muligt at starte eye-tracking før nødvendig opsætning er fuldført.

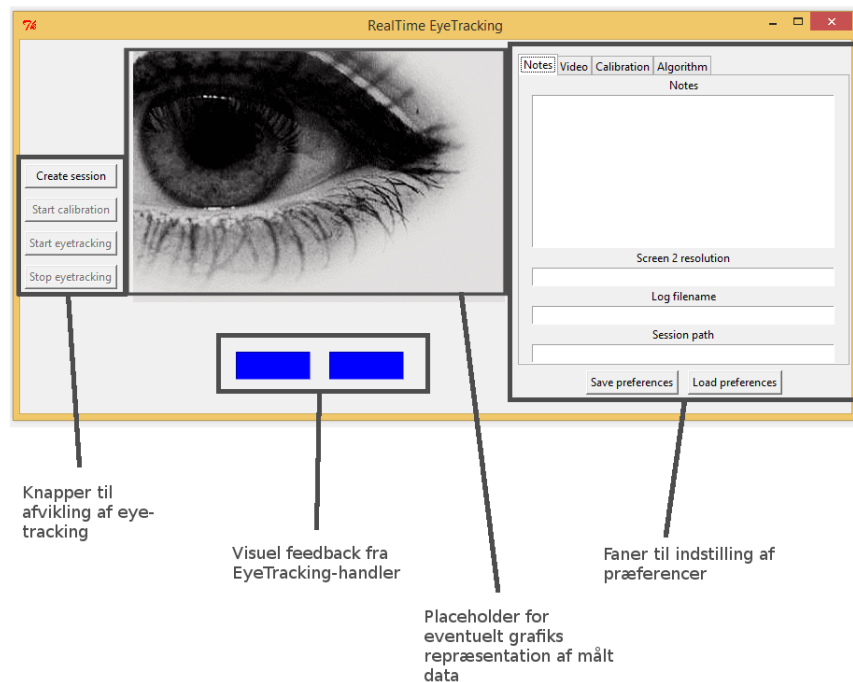
Figur 20 beskriver det forventede grafiske bruger interface. Interfaces er designet ud fra use case diagrammerne beskrevet i kravspecifikationen. De fire faner til højre i interfacet - notes, video, calibration, algorithm - tillader brugeren at læse og inputte ønskede præferencer. Følgende værdier er tilstede:

## 6.6 Diskussion

Designfasen har udledt i et softwarearkitektur-dokument, som har vist sig at være et rigtig godt fundament for implementering af systemet. Den tilnærmelsesvis direkte omskrivning fra use case til sekvensdiagram har medført at al funktionalitet kunne implementeres korrekt fra starten.



Figur 20: Udkast til det grafiske bruger-interface



Figur 21: Implementeringen af det grafiske user-interface med kommentarer

## 7 Implementering

### 7.1 Indledning

### 7.2 Bruger-interface og applikation

I følgende afsnit vil implementeringen af bruger-interfacet og applikationen blive beskrevet.

#### 7.2.1 UserInterface

Tkinter objekter (se afsnit 3.3.2) er benyttet til at opbygge det grafisk brugerinterface som givet i design. Hver knap er blevet oprettet med et tilknyttet funktionskald der udfører den ønskede funktion, som givet af de forskellige use cases. Brugeren ser kun Tkinter objekter. Knapperne til venstre i interfacet er sat i rækkefølge efter naturlig opsætning og eksekvering af real-time eye-tracking. Ikke tilgængelige funktioner har deaktiverede knapper således at man ikke kan foretage ulovlige handlinger.

De fire faner til højre er til indstilling af præferencer. Herfra kan man tilgå, læse og ændre alle præferencer, samt gemme og indlæse præference-filer.

I bunden af bruger-interfacet er to farvede bokse. Disse bokse bruges til at give visuelt feedback bestemt af klassen `EyeTrackingHandler`. Farven blå indikerer at der ikke kører nogen eye-tracking. Grøn indikerer at eye-tracking kører. Igennem `EyeTrackingHandler` er det muligt at sætte farven på de to bokse til rød, hvilket kan bruges til at indikere eventuelle fejl. I denne implementering med Starburst-algoritmen er hver boks sat til at repræsentere et øje, således at rød farve indikerer at algoritmen ikke kan finde det tilsvarende øje. Interaktion med andre klasser: For hver tilgang til præference-filen bliver der oprettet en ny instans af dataklassen `SessionData` fra klassen `SessionHandler`. Ved at aflæse indholdet/værdien af de forskellige Tkinter-objekter og skrive dette data til den instansen af `SessionData`, der senere bliver skrevet til præference-filen af klassen `SessionHandler`, vil der altid være konsistens mellem værdierne skrevet i brugerinterfacet, værdierne i præference-filen, og værdierne brugt i eye-tracking-algoritmen.

Programbibliotekerne `tkFileDialog` og `tkMessageBox` giver værktøjer til at kommunikere med brugeren. `tkFileDialog` håndterer arbejdet med valg af filer, og giver brugeren mulighed for at bevæge sig rundt i mapper på computeren. `tkMessageBox` bruges til pop-up-vinduer, hvilket benyttes til at tvinge brugeren til at gennemføre valg for at fortsætte, samt til at prompte brugeren om fejl og ulovlige handlinger.

### 7.2.2 SessionHandler

`SessionHandler`'s opgave er at stille et data-format til rådighed for kommunikation af sessions-indstillinger de forskellige klasser imellem. Klassens vigtigste opgave er derfor at opretholde et korrekt format. Ved oprettelse eller opdatering af præference-fil, kreerer `SessionHandler` en streng med henholdsvis navn på variabeltype, variabel værdi konverteret til streng (hvis variabelen er tom, skrives strengen "None"), efterfulgt af et linebreak. Når strengen er oprettet for alle variabler, skrives strengen til en præference-fil med suffiks `.pref`.

Når en præference-fil indlæses, verificeres den først af `SessionHandler` ved at lede efter alle variabelnavne i den læste fil. Herefter kan hver variabel udtrækkes fra filen. Det er nødvendigt at lave tjek på hver variabel, da konvertering fra python- og numpy-værdi til streng kan medføre komplikationer.

Instanser af `SessionHandler` brugt som data-format bliver oftest omtalt som `SessionData`.

Figur 22: Eksempel på præference-fil

```

1 SESSIONPATH C:/RealTimeEyeTracking
2 USINGCAM True
3 CAMNR None
4 VIDEOPATH 0
5 NOTES Dette er en testsession
6 RESOLUTION 1600x900
7 CALTYPE Numbers
8 LOGFILENAME testlog
9 CALFILENAME testcallog
10 LOADEDCAldata None
11 RECORDVIDEO False
12 RAWDATAPATH None
13 VARIABLENAMES[ 'e_center', 'last_eyes', ...
14 VARIABLEVALUES[ '[0,0]' '$ ' [] '$ 'True'$ '20'$ '1500'$ ...

```

Figur 22 er et eksempel på præference-fil med korrekt format (Værdierne variablenames og variablevalues er trunckeret for formaterings skyld).

### 7.2.3 LogHandler

For hver eye-tracking måling der igangsættes bliver der oprettet en instans af LogHandler klassen med argumentet SessionData, som er en instans af klassen SessionHandler. LogHandler-klassen består af tre simple funktioner: StartNewTracking, StopTracking og LogData. StartNewTracking er en del af kommunikationsvejen fra UserInterface til EyeTrackingHandler, og har til formål at pege på sig selv, således at EyeTrackingHandler kan kalde LogData i den rette instans af klassen. Ligeledes StartNewTracking er StopTracking en del af kommunikationsvejen, men har kun til formål at sætte klassens variabel Tracking til False. Funktionen LogData bliver kaldt fra EyeTrackingHandler med en streng som argument, og har til formål at skrive denne streng - efterfulgt af et linebreak - til en log-fil peget på af SessionData.

Figur 23 er et eksempel på en log-fil med korrekt format (Timestamp, X, Y, Triggerniveau, Fejlmeddelelse).

### 7.2.4 EyeTrackingHandler

Ved implementering af denne klasse har det været vigtigt at gøre en række overvejelser angående modulariseringen af koden, da det er denne klasse der indirekte står for afviklingen af algoritmen. Funktionen StartVideoCapture bliver kaldt med data-formatet SessionData og en pointer til den relevante instans af LogHandler-klassen. Funktionen opretter en instans af klassen Vi-

Figur 23: Eksempel på log-fil

```

1 16:28:46.483000, 89, 69, 0, None
2 16:28:46.662000, 89, 69, 0, None
3 16:28:46.731000, 90, 70, 0, None
4 16:28:46.782000, 90, 71, 0, None
5 16:28:46.951000, 92, 74, 0, None
6 16:28:47.093000, 89, 68, 0, None
7 16:28:47.244000, 89, 51, 1, None
8 16:28:47.322000, 89, 52, 1, None
9 16:28:47.360000, 91, 52, 1, None
10 16:28:47.395000, 89, 51, 1, None
11 16:28:47.598000, 0, 0, 0, Maximum ransac iterations exceeded
12 16:28:47.651000, 89, 51, 1, None

```

deoCapture med de korrekte argumenter taget fra SessionData. Funktionen StopVideoCapture kalder blot funktionen StopTracking fra klassen VideoCapture, og sætter variablen running til returværdien derfra.

For at sørge for at algoritmen kan afvikles uden forstyrrelser ved opdateringen af det grafiske bruger-interface, indeholder EyeTrackingHandler en EyeTrackingThread-klasse implementeret som en tråd. Når EyeTrackingHandlers funktion StartVideoCapture bliver kaldt, vil en ny process med tråden blive oprettet. Denne tråd vil kalde funktionen updateVideo fra klassen VideoCapture med de valgte argumenter, vente tiden 1/opdateringshastighed, og køre sig selv fra starten så længe at variablen running=True.

Til at modtage data fra algoritmen er de tre funktioner PackWithTimestamp, ReturnError og LastRunInfo. PackWithTimestamp bliver kaldt med argumenterne XY-koordinat og triggerniveau, hvorefter den opretter en streng med den nuværende tid, x-koordinatet, y-koordinatet, triggerniveauet og værdien 'None'. Denne streng sendes derefter som argument til den relevante instans af LogHandler-klassen ved funktionen LogData. ReturnError kan bruges af algoritmen til at returnere fejlmeddelelser. Funktionen modtager fejlmeddelelsen som en streng og opretter en ny streng i samme fæcon som PackWithTimestamp, blot med koordinaterne (0,0), triggerniveau 0, og fejlmeddelelse. Derefter sendes denne streng også til LogHandler-klassens LogData. LastRunInfo giver algoritmen mulighed for at ændre i de ti frie variabler. Variablerne modtages af algoritmen som argument til funktionen LastRunInfo, der derefter opdatere variablerne i SessionData, således at de bliver benyttet næste gang updateVideo kører. Herved er der fri mulighed for at sende værdier så som fejlmeddelelser, sidste kendte koordinater og så videre tilbage til algoritmen.

### 7.2.5 VideoCapture

Indeholder public funktion `GetCameraInputs`, som kan tilgås uden instantiering af klassen. `GetCameraInputs` detekterer antal kameraer tilsluttet computeren ved at iterere igennem en liste af kamera-inputs givet af operativsystemet. Når et indeks på listen returnere en fejl, returnerer funktionen antal kameraer fundet inden fejlen.

De andre funktioner kan først tilgås når klassen er instantieret. Klassen instantieres med et video-input, og mulighed for kalibreringsdataet, samt ti frie variabler. Kalibreringsdataet og de ti frie variabler benyttes ikke til andet end at videresende til `ETAlgorithm` sammen med gyldig frame. At passe kalibreringsdataet og de frie variabler igennem `VideoCapture` er ikke optimalt, men spiller ingen rolle i det samlede arbejds-læs. Det kan forestilles at man i stedet ville sende to pointers til hvor disse data ville være lageret. Argumentet for at passe disse data igennem `VideoCapture` er, at `VideoCapture` så vil kalde `ETAlgorithm's Track()` med den fundne frame samt disse data, i stedet for at returnere den fundne frame til `EyeTrackingHandler` og derefter kalde `Track()`. `VideoCapture` håndterer al kommunikation med video-kameraer, og har tilsvarende ansvaret for at terminere forbindelsen med disse når en måling er overstået.

### 7.2.6 Kommunikation imellem applikationens klasser

Overordnet kan programmet befinde sig i tre stadier: `NotRunning`, `Calibrating` og `Running`. Næsten alle klasser har en variabel der fortæller hvor vidt algoritmen kører eller ej. Herved kan det sikres at systemet ikke forsøger at instantiere klasser, åbne video-inputs, eller overskrive filer der, allerede er i brug.

## 7.3 ETAlgorithm

Koden i figur 24 er en tom udgave af klassen `ETAlgorithm` med de nødvendige input-argumenter og funktionskald til `EyeTrackingHandler` er implementeret:

Afsnit 7.4 beskriver implementering af Starburst-algoritmen i `ETAlgorithm`.

Figur 24: Tom udgave af klassen ETAlgorithm

```

1 import numpy as np
2 import EyeTracking as et
3
4 #Her tilføjes diverse python-biblioteker
5
6 def Track(frame, calibration_data, variables):
7
8     #Egen kode her
9
10         et.PackWithTimestamp(screen_point, trigger_value)
11         et.LastRunInfo(variables)
12     return

```

## 7.4 Starburst-algoritmen

## 7.5 Optimering

### 7.5.1 Fremtidige optimeringsmuligheder

Python understøtter multitrådning, hvilket giver mulighed for at køre flere instanser af eye-tracking-algoritmen sideløbende. Skriv mere om muligheder for multitrådning, og hvorfor det kan virke.

## 7.6 Diskussion



## 8 Test

### 8.1 Indledning

### 8.2 Deltest

Deltestene har til formål at teste hver enkelte klasses funktionalitet. Da de forskellige klasser er skrevet i hver sin python-fil, og da python-filer uden problemer kan inkluderes i python-programmer, har hver klasse været mulig at teste ved at skrive kalde funktionaliteter med specifikke inputs og derefter læse returværdierne/observere hændelserne.

Ved at skrive en test-bench (her et python-program) har det været muligt at teste hver klasses funktionaliteter. Alle test-cases er udført i værktøjet Visual Studio, hvilket giver fri mulighed for at aflæse de forskellige variabler i runtime. Følgende er kort beskrivelse af deltest og resultater for hver klasse:

- **UserInterface:** Klassen er testet ved at køre programmet og observere at: Bruger-interfacet indeholder de samme knapper og tekstfelter som angivet i softwarearkitekturen, knapperne forsøger at kalde de rigtige funktioner når der trykkes på dem, man kan aflæse korrekte værdier i tekstfelterne, og at både program og bruger kan skrive værdier i tekstfelterne.
- **SessionHandler:** Her er testet at en oprettet instans af klassen kan: Gemme egen data i en fil, læse og sætte egen data fra en fil, og verificere korrekt - eller afvise ugyldigt - filformat.
- **LogHandler:** Her er testet at en oprettet instans af klassen kan: Oprette log-fil ud fra givet filsti og navn og tilføje strenge med korrekt format til den angivne log-fil.
- **EyeTrackingHandler:** Denne klasse er først testet for de funktionskald der skal komme fra ETAlgorithm. Her er testet at den kan: Modtage resultater og tilføje et korrekt timestamp til dem, modtage en fejlmeddelelse og tilføje et korrekt timestamp, formatere en streng i outputformatet givet i kravspecifikationen, og modtage et array på ti variabler. Herefter er det testet at klassen kan oprette en process hvori en tråd gentager sig selv med en hastighed på 1/opdateringshastighed sekunder.
- **VideoCapture:** Her er der testet at klassen kan benytte OpenCV til at detektere og benytte video-hardware, læse videofiler, samt læse opdateringshastigheden fra video-hardware og videofiler, eller sætte en

standard hastighed hvis ingen kunne læses (noget video-hardware kan ikke aflæses).

Hvis nogle punkter ikke har kunne godkendes når denne test har været kørt, har det været muligt at gå tilbage og ændre i implementeringen.

### 8.3 Integrationstest

Integrationstesten har til formål at teste kommunikationen klasser imellem. Ligesom i deltesten har denne test kunne udføres ved hjælp af en test-bench, dog med to klasser benyttet af gangen. Hver kommunikationsvej (som set på figur 19) er her testet. Funktionskald og de dertilhørende argumenter er i denne test blevet verificeret. Denne test har ikke medført nogle problemer, da der i softwarearkitekturen har været defineret en klar kommunikation klasserne imellem, og da Python benytter sig af et høj-niveau data-typer.

### 8.4 Accepttest

Accepttesten er formuleret efter de opstillede krav i afsnit 4. Formålet ved accepttesten er at teste det samlede produkt op imod de stillede krav. Testen afsluttes når alle specificerede test cases er gennemført og godkendt. I tilfælde hvor et krav ikke har kunne fuldføres, er der udfærdiget en problemrapport årsagen til underkendelsen.

### 8.5 Performance-evaluering

### 8.6 Diskussion af testresultater

## 9 Konklusion

## A Appendiks

## Figurer

1	V-modellen	7
2	Tidsplan for projektet	8
3	Kameraets position i forhold til testperson	12
4	Eksempel på tærskelværdier for trigger-signalet	12
5	Use case 3	16
6	Enkelt billede fra video optagelse	18
7	Udsnit af øje	19
8	Efter øgning af kontrast og lysstyrke	19
9	Refleksioner fjernet	20
10	Estimering af pupilmidtpunkt	20
11	Starburst, første iteration (Enkelt stråle)	20
12	Starburst, første iteration (Resterende stråler)	21
13	Starburst, anden iteration (Et startpunkt)	21
14	Starburst, anden iteration (Alle Punkter)	22
15	Resulterende ellipse	22
16	Resulterende gaze vektor	23
17	Systemdiagram	24
18	Sekvensdiagram - UC3	28
19	Klasseinteraktioner	31
20	Udkast til GUI	33
21	Implementering af GUI	34
22	Eksempel på præference-fil	36
23	Eksempel på log-fil	37
24	Tom udgave af klassen ETAlgorithm	39

## Referencer

- [1] Wikipedia. (2015). V-model (software development) — Wikipedia, the free encyclopedia, side: [http://en.wikipedia.org/wiki/V-Model\\_\(software\\_development\)](http://en.wikipedia.org/wiki/V-Model_(software_development)) (sidst set 28.05.2015).
- [2] O. M. Group. (2015). Unified modelling language, side: <http://www.uml.org/> (sidst set 28.05.2015).
- [3] Itseez. (2015). Opencv, side: <http://opencv.org/> (sidst set 28.05.2015).
- [4] P. S. Foundation. (2015). Comparing python to other languages | python.org, side: <https://www.python.org/doc/essays/comparisons/> (sidst set 01.06.2015).

- [5] S. van der Walt, S. Cobert og G. Varoquaux, “The numpy array: A structure for efficient numerical computation”, *Computing in Science Engineering*, 7. mar. 2011.
- [6] P. Hughes, “Python and tkinter programming”, *Linux J.*, 1. sep. 2000.
- [7] M. Fairchild, *Color Appearance Models*. 1998, citeret i D. Siboska, H. Karstoft og H. Pedersen, p.5.
- [8] D. Siboska, H. Karstoft og H. Pedersen, *Synchronization of electroencephalography and eye tracking using global illumination changes*.