Rapport

Bachelorprojektet:
Real-time eye-tracking
Projektnummer: 15017

25/05/2015

Studerende: Søren Vøgg Krabbe Lyster (SVL) 10920,
Martin Degn Kristensen (MDK) 10441
Studieretning: Elektro
Vejleder: Preben Kidmose

Indhold

1 Prolog

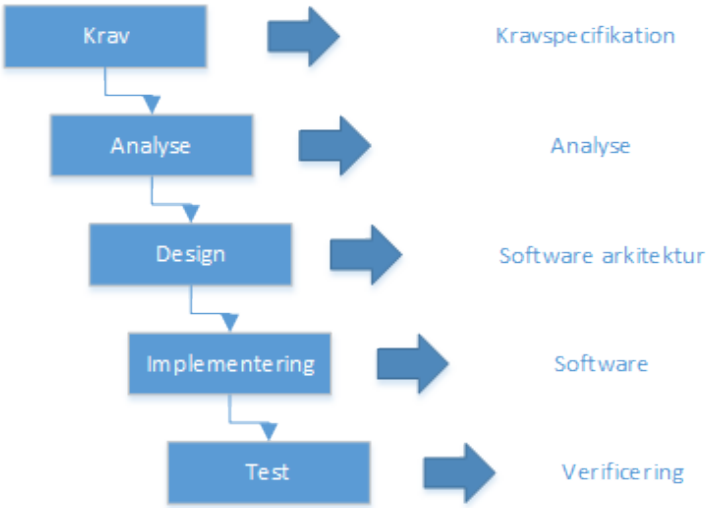
1.1 Indledning

1.2 Ordforklaring

1.3 Projektformulering

1.4 OpenEyes og Siboska

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



Figur 1

2 Metode

2.1 Metoder

Hvilken metode har været brugt til udførsel af projektet

2.2 Planlægning

Hvad har der været gjort angående strukturering af projektet

2.3 Diskussion

Har de valg vi har taget været relevante? Kort reflektion

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.

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. [1].

3.3 Python

Noget om python Numpy 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. [reference til Tkinter]

Find en reference der skitserer fordelene ved python over c++.

3.4 Diskussion

Har de valg vi har taget været relevante? Kort refleksion

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

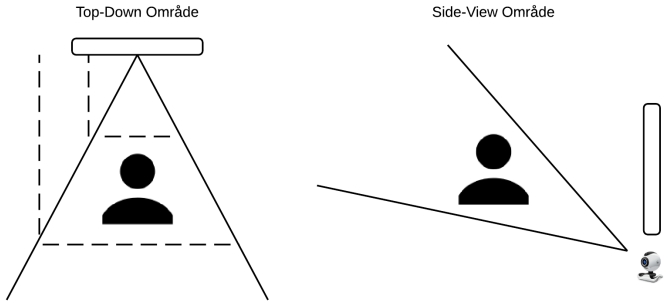
Følgende funktionelle krav for systemet er blevet stillet:

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. **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 4.5.

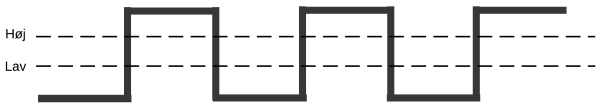
4.3 Kalibrering

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



Figur 2: 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 3: Eksempel på tærskelværdier for trigger-signalet

4.4 Output

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

- Kommasepareret målingsdata med følgende format:
  X-koordinat, Y-koordinat, Samplenummer, Trigger-niveau (0 for lav, 1 for høj)

4.5 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.

4.6 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 $<2°$[2].

- **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 60m. 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 [3].

4.7 Performance-evaluering

4.8 Diskussion

5 Analyse

5.1 Indledning

5.2 Systemoversigt

5.3 Overvejelser

5.3.1 Krav

5.3.2 Kodesprog

5.4 Starburst-algoritmen

5.5 OpenEyes og Siboska

5.6 Diskussion

6 Design

For at oprette en software arkitektur der overholder de krav stillet i kravspecifikationen, er der gjort brug af UML (Unified Modelling Language) [**?**]. 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.

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

6.1 Indledning

Kort om hvad dette afsnit indeholder, og hvorfor det er relevant.

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

Overvejelser om opdelingen af klasser. Hver klasse varetager 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.

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.

6.3.2 SessionHandler

Her håndteres al data tilknyttet opsætning af eye-tracking-session.

6.3.3 LogHandler

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

6.3.4 VideoCapture

6.3.5 EyeTrackingHandler

Denne klasse håndterer kommunikation og instantiering af klassen ETAlgorithm.

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 returnerer XY-koordinat samt trigger-niveau.

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.

Følgende figurer beskriver det forventede grafiske bruger interface. Interfaces er designet ud fra use case diagrammerne beskrevet i kravspecifikationen.

6.6 Diskussion

7 Implementering

7.1 Indledning

7.2 Bruger-interface og applikation

7.3 Starburst-algoritmen

7.4 Optimering

7.5 Deltest

7.6 Diskussion

8 Test

8.1 Indledning

8.2 Accepttest

8.3 Performance-evaluering

8.4 Diskussion af testresultater

9 Konklusion

10 Appendiks

Litteratur

[1] G. Bradski *Dr. Dobb's Journal of Software Tools.*

[2] M. Fairchild, *Color Appearance Models.* cited in D. Siboska, H. Karstoft and H. Pedersen, p.5.

[3] D. Siboska, H. Karstoft, and H. Pedersen, "Synchronization of electroencephalography and eye tracking using global illumination changes."