Arkitektur och användargränssnitt hos ögonstyrda tillämpningar

Utveckling av ett generiskt
ögonstyrningssystem och en prototyptillämpning

CHRISTOFFER BJÖRKLUND

KTH Numerisk analys
och datalogi

Examensarbete
Stockholm, Sverige 2005

TRITA-NA-E05179

Numerisk analys och datalogi
KTH
100 44 Stockholm

Department of Numerical Analysis
and Computer Science
Royal Institute of Technology
SE-100 44 Stockholm, Sweden

Arkitektur och användargränssnitt hos ögonstyrda tillämpningar

Utveckling av ett generiskt ögonstyrningssystem och en prototyptillämpning

CHRISTOFFER BJÖRKLUND

TRITA-NA-E05179

Examensarbete i datalogi om 20 poäng
vid Programmet för datateknik,
Kungliga Tekniska Högskolan år 2005
Handledare på Nada var Björn Eiderbäck
Examinator var Yngve Sundblad

Arkitektur och användargränssnitt hos ögonstyrda tillämpningar

Utveckling av ett generiskt ögonstyrningssystem och en prototyptillämpning

Sammanfattning

Med blickmätning kan en dator avgöra var på skärmen användaren tittar. Denna information kan användas till att antingen analysera användarens beteende eller till att styra en datortillämpning. Denna rapport beskriver ett examensarbete om arkitekturen och användargränssnittet hos ögonstyrda tillämpningar.

En undersökning görs om vilken underliggande generisk funktionalitet som behövs av en ögonstyrd tillämpning. Ett ögonstyrningssystem som innehåller denna funktionalitet implementeras sedan. För att testa systemet och för att ge ett exempel på hur interaktion med ett ögonstyrt användargränssnitt kan gå till utvecklas en prototyp av en ögonstyrd filhanterare.

Filhanteraren består av tre olika komponenter, en listvy, en trädvy och en ny typ av verktygsfält, ögonverktygsfältet. Ögonverktygsfältet är en blandning av den traditionella menyn och verktygsfältet. Det uppvisade stora förbättringar i snabbhet vid val av kommando. Med filhanteraren kan användaren flytta och exekvera filer samt undersöka kataloger genom att scrolla vyerna. Användaren kan välja mellan att använda en knapp eller fördröjningstid för att utföra ett kommando.

Komponenterna i filhanteraren kommunicerar med ögonstyrningssystemet genom att registrera rektangulära regioner (blickregioner) på skärmen vilka meddelas när användarens blick träder in över eller lämnar dessa regioner. Ögonstyrningssystemet använder ett fixationsfilter för att få en stadig blickpunkt vilken används till att avgöra vilken blickregion som användaren tittar på. Filhanteraren fungerade väl och ögonstyrningssystemet levererade den basfunktionalitet som krävs av en ögonstyrd tillämpning.

Architecture and user interface of eye controlled applications

Development of a generic eye control system and an application prototype

Abstract

With eye tracking a computer application can tell where on the screen the user is looking. This information can be used to either analyze the user´s behaviour or to control a computer application. This report describes a master's project about the architecture and user interface of eye controlled applications.

An investigation is made on what underlying generic functionality is needed by an eye controlled application. An eye control system which contains this functionality is then implemented. To test the system and to give an example of how interaction with an eye controlled user interface can be performed, a prototype of an eye controlled file browser is developed.

The file browser consists of three different components, a listview, a treeview and a new kind of toolbar, the eyetoolbar. The eyetoolbar is a mixture of the traditional menu and toolbar. It showed great speed gains in command selection. With the file browser the user can move and execute files and browse catalogs by scrolling the views. The user can choose between using a button or dwelltime for command execution.

The components of the file browser communicate with the eye control system by registering rectangular regions (gaze regions) of the screen which get notified when the user's gaze enters or leaves these regions. The eye control system uses a fixation filter to get a steady gaze point which is used to determine which gaze region the user is looking at. The resulting file browser performed well and the eye control system delivered the basic functionality needed for an eye controlled application.

Innehållsförteckning

1 Introduktion

1.1 Bakgrund

Denna rapport sammanfattar ett examensarbete gjort vid institutionen för Numerisk Analys och Datalogi, NADA, på Kungliga Tekniska Högskolan, KTH, i Stockholm. Arbetet genomfördes under våren och sommaren 2003 på uppdrag av företaget Tobii Technology som en del i civilingenjörsutbildningen i datateknik. Arbetet har gått ut på att dels utveckla ett ögonstyrningssystem för att bygga grafiska ögonstyrningsgränssnitt samt att med detta system skapa en prototyp till en ögonstyrningstillämpning. Utvecklingen av ögonstyrningssystemet har haft en utredande karaktär då området är i sin barndom och ett vedertaget sätt hur ett ögonstyrt användargränssnitt ska se ut och bete sig ej ännu finns. Det finns dock en del forskning inom ämnet som kommer att tas upp och diskuteras.

1.2 Tobii Technology

Uppdragsgivaren Tobii Technology är ett ungt utvecklingsföretag som har tagit fram ett system som detekterar var på datorskärmen en användare tittar. Detta kallas för blickmätning. Tobii Technology har tillhandahållit den utrustning som krävts för examensarbetets genomförande. Företaget har tidigare varit inriktat på diagnostiska tillämpningar av blickmätning och hade för avsikt att med detta examensarbete undersöka hur ögonrörelser kan användas till att kommunicera med datorn. Lösningen för blickmätning som Tobii Technology har tagit fram hör till den nya generationens blickmätningssystem där användaren inte har någon fysisk kontakt med utrustningen. Detta är ett stort steg mot visionen hos Tobii Technology, att ögonstyrningen i framtiden kommer att vara en lika självklar del av användargränssnittet hos en dator som musen är idag.

2 Problemdefinition

2.1 Problembeskrivning

De problem som arbetet huvudsakligen behandlade var:

- Hur *kan* interaktion med ett grafiskt användargränssnitt hos en vanlig programtyp gå till då ögonrörelser är den primära modaliteten så att det underlättar arbetet för användaren?
- Hur ska ett ögonstyrningssystem utformas för att det på ett lätt sätt ska kunna erbjuda mjukvaruutvecklare att skapa ögonstyrda användargränssnitt?

Examensarbetets syfte var att utvärdera och ta fram ett exempel på hur ögonstyrning på ett effektivt och användarvänligt sätt ska kunna användas i ett grafiskt användargränssnitt hos en existerande programtyp samt hur det underliggande systemet kan se ut för att underlätta för framtida utveckling av ögonstyrda tillämpningar. Notera att det inte var den optimala lösningen som söktes utan en godtagbar sådan, m.a.o. ett exempel på en möjlig lösning.

2.2 Målsättning

Inom ramen för det ovan beskrivna syftet fastställdes följande målsättning:

- En undersökning av tidigare forskningsresultat ska göras för att fastställa vad som kan vara värdefullt för detta projekt
- Tekniker för hur interaktion med ett grafiskt användargränssnitt kan ske m.h.a. ögonrörelser ska tas fram
- Ett generellt ögonstyrningssystem som på ett enkelt och effektivt sätt erbjuder andra att skapa ögonstyrda användargränssnitt ska designas och implementeras
- En prototyp av en vanligt förekommande tillämpningstyp som använder sig av nämnda ögonstyrningssystem för att exponera ett ögonstyrt grafiskt användargränssnitt ska byggas

2.3 Avgränsningar

Examensarbetet hade inte för avsikt att ta fram den bästa möjliga lösningen på prototyptillämpningen utan mer ett exempel på hur en sådan tillämpning skulle kunna fungera i en ögonstyrd miljö. Det var ej heller tänkt att inom arbetets ram utföra någon användarstudie då en stor del av vikten låg på utformandet av det underliggande ögonstyrningssystemet.

3 Teori

3.1 Det mänskliga seendet

När man utvecklar användargränssnitt som bygger på det mänskliga ögat är det viktigt att känna till ögats fysiska egenskaper samt de underliggande psykologiska processer som sker i det mänskliga synsystemet. Denna kunskap är betydande vid utvecklandet av interaktionsmetoder. Vissa av dessa egenskaper är till fördel vid ett ögonbaserat användargränssnitt andra kan vara till besvär. Informationen i 3.1.1, 3.1.2 och 3.1.3 är hämtat från Duchowski (2003).

3.1.1 Ögats anatomi

De yttre, uppenbara delarna av ögat man lägger märke till hos en människa är senhinnan, iris och pupillen. Senhinnan är den vita delen av ögat, ett tjockt skyddande lager. Den färgade delen av ögat kallas iris. I mitten av iris finns det en öppning, pupillen. Iris är omgiven av små muskler som kan utvidga och dra samman pupillen och på så sätt kontrollera mängden ljus som passerar ögat. Framtill på ögat övergår senhinnan till att bli genomskinlig och byter där namn till hornhinnan. Den är täckt av en tunn fuktgivande film. Ljusstrålar från omgivningen når ögat via hornhinnan, här sker också den större delen av brytningen av ljuset. Ytterligare brytning sker sedan av linsen vilken fokuserar ljuset på näthinnan. Linsen kan ändra form för att skifta fokus mellan närliggande och avlägsna objekt.

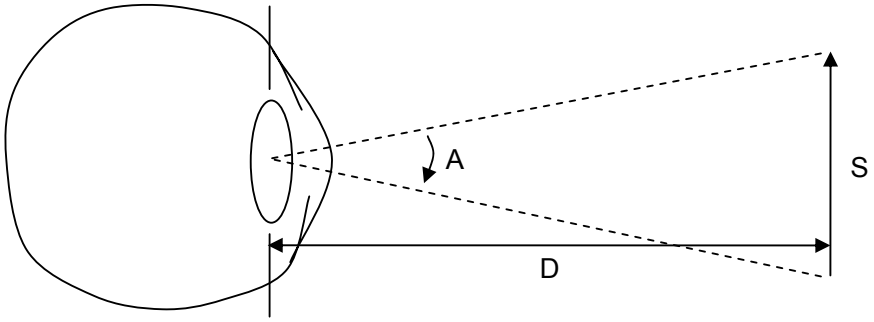
Näthinnan, vilken täcker bakre insidan av ögat, är täckt av miljoner av ljuskänsliga receptorer, fotoreceptorer. Dessa omvandlar ljuset till elektriska impulser som skickas via nerver till visuella centra i hjärnan. Fotoreceptorena delas in i två grupper, stavar och tappar. Förenklat kan man säga att stavarna är känsliga för svagt och akromatiskt ljus medan tapparna reagerar på starkare, kromatiskt ljus. Fovea, eller gula fläcken som den ofta felaktigt kallas eftersom den antar en gulaktig färg hos en avliden, ligger nära centrum av näthinnan och är tätare täckt av fotoreceptorer än övriga delar av näthinnan. Här utgör tapparna majoriteten av fotoreceptorerna, medan stavarna dominerar i näthinnans periferi. Således är fovea avsett för detaljerat seende och omkringliggande delar mer för orienteringsmässigt seende. Fovea svarar på frågan ”*Vad* är det jag tittar på” och de perifera delarna svarar på frågan ”*Var* är det jag tittar”.

3.1.2 Ögats egenskaper

Dimensioner på näthinnan mäts oftast i antal grader av den *visuella vinkel* som det betraktade objektet upptar då det projiceras på näthinnan och definieras som:

$$A = 2\arctan\frac{S}{2D},$$

där S är storleken på objektet och D är avståndet till objektet, se Figur 3.1.



Figur 3.1 Den visuella vinkeln.

En grad av den visuella vinkeln motsvarar ungefär ett avstånd på 300 μm på näthinnan. Fovea har en utsträckning på ca 5° visuell vinkel och 147000 tappar/mm^2 och ett mindre antal stavar. Vid omkring 10° faller antalet tappar kraftigt till mindre än 20000 tappar/mm^2. Synfältet täcker grovt räknat en area på 23400 grader i kvadrat definierat av en ellipsoid med den större horisontella axeln täckandes 180° visuell vinkel och den mindre vertikala axeln täckandes 130°.

Den delen av näthinnan med högst skärpa har en utsträckning på 1° och den del med hög skärpa en utsträckning omkring 4° - 5°. Därefter minskar skärpan kraftigt. Vid 5° visuell vinkel är skärpan endast 50 %. Det ”användbara” synfältet har ett omfång kring 30°, resten har dålig upplösande förmåga och används mest för uppfattning av omgivande rörelser. Eftersom ett objekt kan ses som skarpast inom 1° hos fovea är det inte möjligt att avgöra var en person tittar med större precision än en cirkel med radien av 1°.

Förmågan att uppfatta temporala skillnader är inte homogen över hela synfältet. Rörelse kan uppfattas över hela synfältet, men de foveala regionerna är känsligare för långsammare rörelser än de yttre områdena. Vidare upplevs hastigheten hos rörligt föremål vara lägre i periferin än i fovea. Sålunda behövs en högre hastighet i periferin för att matcha den upplevda hastigheten i fovea. Trots den minskade känsligheten i periferin har rörelse större betydelse där än i det centrala synfältet. D.v.s. periferin är mer känslig för rörliga föremål än stillastående sådana. Upptäckandet av rörelse är faktiskt en huvuduppgift hos det periferiska seendet, det fungerar som ett varningssystem för rörliga föremål som träder in i synfältet.

Känsligheten för färg är starkast i fovea där tapparna är av tre olika typer som reagerar kraftigast på blått, grönt och rött ljus, respektive. Mycket forskning är gjord kring färgseende i fovea men lite om färgkänsligheten i de periferiska områdena av näthinnan. Sådan kunskap skulle vara värdefull vid t.ex. utformandet av en flygplanscockpit. Dock vet man att p.g.a. att majoriteten av de färgkänsliga tapparna finns i fovea har det periferiska seendet relativt dålig förmåga att särskilja kromatiskt ljus.

3.1.3 Ögonrörelser

Ögat hålls på plats av tre par av motstående muskler som kan röra ögat upp-ner, vänster-höger samt rotera kring axeln som går mellan fovea och pupillen. Generellt sett rör sig inte ögat mjukt över synfältet utan gör små hopp, s.k. sackader, tillsammans med andra specialiserade rörelser. Sackader används för att projicera ett önskat visuellt objekt i omgivningen på fovea. Sackader är av ballistisk natur, d.v.s. så fort en sackad har påbörjats går det inte att ändra dess destination eller väg. Sackader kan sträcka sig 1° - 40° visuell vinkel, men sträcker sig oftast 15° - 20° och tar 30-120 millisekunder. Då en sackad pågår är synsystemet kraftigt hämmat. Vidare kan inte en ny sackad påbörjas förrän tidigast 100-200 millisekunder efter att en har avslutats.

Efter en sackad följer ofta en fixation vilket är en period där ögat hålls relativt stabiliserat för att iaktta ett objekt. Även under en fixation håller sig inte ögat helt stilla, utan gör flera små, skakiga rörelser inom en grads radie. Fixationen kan driva iväg sakta men korrigeras då av ett litet, sackadliknande, hopp, en s.k. mikrosackad. Fixationer pågår mellan 200 – 600 millisekunder, därefter påbörjas en ny sackad. De små och skakiga rörelserna är nödvändiga för att ett objekt ska kunna betraktas, om objektet skulle vara fixerat på näthinnan skulle det efter några sekunder tonas ut och försvinna.

Ögat kan endast röra sig mjukt när det följer efter ett rörligt objekt. Denna ögonrörelse är mycket långsammare än en sackad och kan inte uppstå då ögat betraktar ett statiskt stimuli. En annan specialiserad ögonrörelse kallad nystagmus uppstår för att kompensera för huvudrörelser eller då man iakttar ett rörligt repeterande mönster. Detta händer exempelvis då man mjukt följer ett rörligt objekt med blicken tvärs över synfältet för att därefter hastigt hoppa med blicken i motsattriktning och följa ett nytt objekt då det första försvann ut ur synfältet.

I ögonrörelserna beskrivna ovan gör båda ögonen samma rörelser, men de kan även röra sig olika som t.ex. då man skiftar fokus från ett avlägset objekt till ett på nära håll. Då rör sig ögonen mot varandra. Även rotation kring axeln mellan fovea och pupillen kan förekomma vilket bl.a. beror på vinkeln hos nacken. Denna rörelse går inte att upptäcka med en blickmätare.

De viktigaste rörelserna vid ögonbaserade tillämpningar är fixationer, sackader samt den mjuka rörelse som uppstår då ögat följer ett rörligt stimuli. Man antar att dessa rörelser är uttryck för avsiktlig visuell uppmärksamhet. Fixationer visar en persons vilja att studera ett intressant statiskt objekt, och en mjuk förföljning samma vilja för ett rörligt objekt. Sackader tolkas som viljan att avsiktligt byta uppmärksammat område.

3.1.4 Visuell uppmärksamhet

När vi rör våra ögon gör vi det för att ge en viss del av synfältet en hög upplösning så att vi i detalj kan se det som ligger i linje med blicken. Ofta riktar vi även vår uppmärksamhet till denna punkt för att fokusera vår koncentration på objektet eller området i fråga. Följaktligen, om vi kan spåra någons ögonrörelser

kan vi också följa den väg längs vilken personens uppmärksamhet har riktats. Detta kan ge oss information om vad denna person fann intressant, d.v.s. vad som tilldrog sig hans/hennes uppmärksamhet, och kanske även hur personen i fråga upplevde det som han/hon tittade på. Det behöver dock inte alltid vara så att uppmärksamheten riktas åt samma håll som blicken. T.ex. så riktar astronomer sin uppmärksamhet till synfältets periferi då de med bara ögat letar efter svaga stjärnbilder. Detta p.g.a. att det periferiska seendet är mycket känsligare för svag, otydlig stimuli. En svagt lysande stjärna är mycket lättare att se om man tittar bredvid den än om man skulle titta rakt på den. Detta är ett välkänt problem inom blickmätningsforskningen. En blickmätare kan endast fånga ögonens rörelser, de kan inte fånga de dolda rörelserna hos den visuella uppmärksamheten. Följaktligen gör man inom all blickmätning ett underförstått men viktigt antagande, nämligen att uppmärksamheten följer blickens riktning samtidigt som man är medveten om att det inte alltid behöver vara på det sättet.

3.2 Blickmätningstekniker

Det finns flera olika sorters tekniker för att mäta ögonrörelser hos människor. Vissa används främst i kontrollerade miljöer medan andra är mer anpassade för vardagligt arbete. Det finns två huvudtyper av blickmätningstekniker, de som mäter ögats position relativt till huvudet samt de som mäter ögats orientering i rummet. Nedan beskrivs de tre största blickmätningsteknikerna. Källan till avsnitten 3.2.1, 3.2.2 samt 3.2.3 är Duchowski (2003).

3.2.1 Elektro-okulärgrafi

Elektro-okulärgrafi var den mest tillämpade metoden för ungefär 40 år sedan och används än idag. Den bygger på att näthinnan är betydligt mer elektriskt laddad än resten av ögat att det går att uppmäta en potentialskillnad mellan näthinnan och hornhinnan. När ögat rör sig sker ändringar i denna potential, dessa ändringar fångas upp av elektroder placerade på huden omkring ögonhålan. Denna metod mäter ögonens relativa position till huvudet och lämpar sig bättre för mätandet av relativa ögonrörelser än absoluta positioner. Noggrannheten är dålig och p.g.a. de för användaren besvärliga elektroderna passar metoden bättre i kontrollerade miljöer, exempelvis laboratorier.

3.2.2 Kontaktlins

Den mest noggranna, men minst användarvänliga blickmätningsmetoden är då ett fysiskt objekt används som mekanisk eller optisk referens vilket monteras på en kontaktlins som i sin tur fästs på ögat. Därefter återstår det bara att spåra denna referens som bl.a. kan bestå av ett magnetiskt hölje eller en spegel. Även denna metod mäter endast ögonens position relativt huvudet.

3.2.3 Videobaserad med pupill- och hornhinnereflektion

Den mest praktiska lösningen innefattar en kamera, som iakttar användaren på håll, samt ett därtill kopplat bildbehandlingssystem med uppgift att tolka kamerans bilder. I denna videobaserade uppställning görs ofta bildbehandlingen i

realtid. Ögats position kan bestämmas genom att söka efter särdrag hos ögat, t.ex. gränsen mellan senhinnan och iris eller gränsen mellan iris och pupillen. Ögat är ofta belyst av knappt synbart, infrarött ljus. Placeras ljuset koaxialt med kameran uppträder pupillen mycket starkare än sin omgivning och blir följaktligen lätt att identifiera. Detta förfarande räcker dock inte för att avgöra var användaren tittar. Om systemet upptäcker en liten rörelse hos ögonen går det inte att avgöra om användaren antingen flyttade huvudet eller om han/hon rörde ögonen. En för användaren obekväm lösning är att fixera huvudet m.h.a. en bettskena eller ett hakstöd för försäkra sig om att alla fångade rörelser är ögonrörelser. En mer praktisk lösning på problemet är att mäta två olika särdrag hos ögat vilka rör sig olika med hänsyn till varandra då blicken ändrar riktning. Då huvudet flyttas rör sig dessa särdrag tillsammans och förhållandet dem emellan är konstant. När ögonen rör sig ändras däremot förhållandet mellan särdragen. Det enda krav som ställs på huvudrörelser med denna teknik är att huvudet ska finnas inom kamerans synfält. De två särdrag hos ögat som används till detta i störst utsträckning är pupillcentrum och näthinnereflektion. Avståndet dem emellan ändras vid rena ögonrotationer men hålls relativt konstant vid mindre huvudrörelser. Denna metod erbjuder en användarvänlig och pålitlig blickmätningslösning för normala datoranvändare. Den blickmätare som har använts i detta arbete, en Tobii ET-17, hör till den sistnämnda kategorin.

3.2.4 Kalibrering

Dagens videobaserade blickmätare kräver oftast en personlig kalibrering innan de kan användas. Detta p.g.a. att egenskaperna hos olika personers ögon skiljer sig åt. Kalibreringen utgörs ofta av att en sekvens av enkla stimulis, t.ex. svarta punkter på en vit bakgrund, visas för användaren en efter en på utvalda extrema positioner på skärmen. Användaren uppmanas att fästa blicken vid kalibreringspunkterna varvid blickmätaren observerar förhållandet mellan pupillen och näthinnans reflektion vid dessa punkter för att sedan kunna interpolera fram användarens blickpunkt vid punkter mellan kalibreringspunkterna. En ytterligare anledning till att kalibrera kan vara att blickmätarens optik kan behöva justeras för att särdragen hos ögat ska kunna mätas korrekt. Individuella skillnader som exempelvis långa ögonfransar, glasögon och kontaktlinser kan vara ett skäl till detta. En kalibrerings kvalité är ofta beroende av dess livslängd p.g.a. att förhållanden i miljön där blickmätaren används, t.ex. ljuset, ändras med tiden vilket gör att kalibreringens information blir missvisande. Ett mål inom blickmätningen idag är att skapa system som inte kräver kalibrering alternativt långlivade kalibreringar som fungerar lika bra efter en månad som en timme efter att de skapades.

3.3 Blickmätningstillämpningar

Området av blickmätningstillämpningar är mycket brett och kan delas in i två huvudkategorier, *diagnostiska* och *interaktiva*. Diagnostiska tillämpningar använder blickmätaren för att erhålla information om var en försöksperson riktar sin uppmärksamhet. I interaktiva tillämpningar används användarens blick som en inmatningsenhet vilken kan användas av program i realtid.

3.3.1 Diagnostiska tillämpningar

I diagnostiska tillämpningar sparas generellt sett bara information om var en försöksperson tittar på skärmen under ett försök. Denna information analyseras och bearbetas sedan efteråt. Vanligtvis reagerar inte det som visas på skärmen på användarens blick och ibland kan det vara nödvändigt att dölja att man använder en blickmätare i försöket för försökspersonen. Diagnostiska tillämpningar används bl.a. inom områdena neurologi och psykologi, reklam samt användbarhetsstudier.

Inom reklam kan t.ex. blickmätning användas för att avgöra var en annons bör placeras på en webbsida för att dra till sig uppmärksamhet. Ett försök kan då göras genom att man låter ett antal försökspersoner surfa runt på en webbsida med annonsen i fråga samtidigt som man räknar antalet fixeringar på annonsen. Detta försök kan sedan repeteras flera gånger med annonsen placerad på olika ställen för varje försök. Slutsats kan sedan dras genom att studera de registrerade fixationerna.

Tillämpningar inom neurologi och psykologi används bl.a. för att forska om hur vissa delar i hjärnan fungerar, hur man upplever vissa miljöer och hur människor med vissa sjukdomar utför handlingar som att läsa exempelvis.

3.3.2 Interaktiva tillämpningar

När blickmätaren fungerar som en inmatningsenhet skall tillämpningen ifråga på något sätt reagera på användarens blick. Dessa interaktiva tillämpningar kan delas in i två undergrupper, *selektiva* och *blickbetingade* (Eng. gaze-contingent) (Duchowski, 2002). I den förra gruppen används blicken antingen till att utföra uppgifter explicit eller för att implicit låta programmet ta olika beslut beroende på var användaren tittar. I blickbetingade tillämpningar används vetskapen om var användaren tittar på skärmen samt kunskapen om det mänskliga synsystemet till att optimera användningen av någon resurs. En optimering man kan göra är att låta detaljrikedomen på skärmen vara som störst där användaren tittar för att sedan avta ut mot periferin. Skälet är oftast att minimera användningen av bandbredd, som i t.ex. videotelefoni, eller att det inte är möjligt att i grafiska tillämpningar visualisera hela, komplexa datamängder i realtid.

Ett enkelt exempel på en selektiv interaktiv tillämpning är ett ögonstyrt användargränssnitt där muspekaren följer blicken. Det finns en hel del arbete gjort inom ögonstyrda användargränssnitt vilket kommer att tas upp mer i detalj i nästa avsnitt. Ett annat användningsområde för selektiva tillämpningar är kollaborativa system där blicken bl.a. kan användas till att avgöra vem som pratar med vem samt vem som pratar om vad, se Vertegaal m.fl. (2000).

3.4 Ögonstyrda användargränssnitt

Möjligheten att avgöra var en människa tittar är inte ny, men de tillängliga lösningarna har länge varit både dyra och besvärliga och därmed begränsat användningen. Blickmätningsteknikerna har till största delen tidigare tillämpats diagnostiskt inom psykologisk forskning av perceptuella och kognitiva processer samt i användbarhetsstudier. Idag är blickmätningssystem fortfarande dyra och används främst i olika forskningssammanhang, men har blivit betydligt mindre besvärliga då ingen fysisk kontakt med systemet behövs. Dock går utvecklingen framåt och billigare och mer avancerade system skapas hela tiden. I framtiden kommer förmodligen ögonstyrning att nå vanliga datoranvändare och bli en alternativ inmatningsform. En hel del forskning har gjorts inom ögonstyrning och kommer i detta avsnitt att tas upp mer i detalj.

3.4.1 Klassificering

Direkt manipulering av objekt i ett grafiskt användargränssnitt har likheter med existerande mänskliga färdigheter, t.ex. peka och flytta objekt, snarare än tränade beteenden. Detta tänkande är svårare att föra över till ett ögonstyrt användargränssnitt då det är få objekt i den verkliga världen som reagerar på ögonrörelser. Man kan urskilja två egenskaper hos ögonrörelsebaserad interaktion, dels karaktären hos användarens ögonrörelser, dels karaktären hos gränssnittets respons på ögonrörelserna. Enligt Jacob (1995) kan var och ett av dessa ses som antingen *naturlig*, d.v.s. baserat på en analogi med den verkliga världen, eller *onaturlig*, d.v.s. ingen motsvarighet i den verkliga världen.

Jacob menar att i det naturliga fallet skulle användaren betrakta skärmen på samma sätt som han/hon skulle ha gjort även då det inte hade funnits någon blickmätningsutrustning närvarande. Det onaturliga alternativet skulle vara att instruera användaren att utföra vissa speciella, tränade ögonrörelser. De båda fallen där ögonrörelserna resulterar i naturliga reaktioner i gränssnittet är ett svårt fall p.g.a. att det, som tidigare nämnts, endast finns en liten mängd företeelser i den verkliga världen som reagerar på ögonrörelser. Tillämpningar med onaturlig respons kan således betraktas som ett stort område inom ögonstyrning. Användningen av onaturliga ögonrörelser bör undvikas då detta är besvärligt för användaren eftersom människan inte är van vid, eller skapt för, att tänka på hur man använder ögonen. Å andra sidan kan det vara positivt då det ökar bandbredden från människan till datorn, vilket kan vara nödvändigt för personer som är begränsade till att endast använda ögonen, exempelvis handikappade.

Ytterligare klassificering av ögonstyrningsgränssnitt kan göras, de gränssnitt som kommunicerar med användaren *explicit* respektive *implicit* (Jacob, 1995). Denna indelning gäller inte endast ögonstyrda gränssnitt utan är applicerbar på alla typer av gränssnitt. Explicit interaktion innebär att användaren medvetet ger kommandon åt datorn att utföra någon uppgift, exempelvis att kopiera en fil. I ett ögonstyrningsgränssnitt med implicit interaktion å andra sidan, observerar systemet användarens blick för att ta reda på vad användaren är intresserad av,

vad han/hon vill, och agerar sedan utifrån det. Vidare kan ögonstyrningsgränssnitt benämnas som *rena* eller *kombinerade*. I rena ögonstyrningsgränssnitt förlitar sig användaren helt på ögonrörelser för att kommunicera med datorn. Detta kan bl.a. användas i situationer där händerna är upptagna med annat eller där det helt enkelt inte finns plats för övriga inmatningsenheter. En annan målgrupp är handikappade som har begränsad rörlighet. I ett kombinerat ögonstyrningsgränssnitt kombineras blickmätaren med andra inmatningenheter. Detta kan vara bra för att undvika de nackdelar ögonstyrning medför samtidigt som man utnyttjar ögonstyrningens fördelar.

3.4.2 Fördelar med ögonstyrning

Då användaren i ett konventionellt musstyrt användargränssnitt skall klicka på ett objekt för han/hon först blicken till objektet och därefter muspekaren. Av detta kan vi dra två slutsatser. Dels att det är naturligt och lätt för användaren att använda ögonen för att välja objekt p.g.a. att han/hon alltid först tittar på objektet i fråga innan han/hon väljer att klicka på det med musen. Den andra slutsatsen är att det går mycket snabbare att välja ett objekt med ögonen, dels p.g.a. att steget där förflyttningen av musen sker försvinner. Då gränssnittet redan vet vilket objekt användaren avser behöver användaren inte göra något annat än att t.ex. trycka på en knapp för att objektet ska väljas. En annan anledning till att ögonstyrning är snabbt är att ögat kan röra sig väsentligt snabbare än andra delar av kroppen.

Om ögonstyrning används som alternativ till manuella inmatningsenheter, minskar arm- och handrörelser och därmed också skador som dessa rörelser kan åstadkomma.

Att användarens blick kan registreras utan att han/hon behöver ha någon fysisk kontakt med utrustningen är en stor fördel. Användaren kan alltså använda ett gränssnitt trots att han/hon inte kan använda händerna, det kan vara att de är upptagna med annat, att andra inmatningsenheter inte får plats eller att personen har ett handikapp som gör att han/hon är oförmögen att använda händerna. Om fallet är att händerna används till annat erbjuder ögonstyrning en extra modalitet och förhöjer på så sätt interaktiviteten.

3.4.3 Nackdelar med ögonstyrning

Det förmodligen enklaste gränssnittet baserat på ögonrörelser skulle kunna åstadkommas genom att låta ögonen styra muspekaren. Detta är dock opassande då ögonrörelser ofta är omedvetna och människan är van vid att använda ögonen utan att ändringar i blickriktningen ”betyder” något. Detta är en skillnad jämfört med manuella inmatningsenheter som aktiveras endast då användaren vill att de ska aktiveras. Ögonen är således alltid ”på”. Denna egenskap hos ögonen gör det svårt för ett gränssnitt att endast från ögonrörelser avgöra när användaren vill exekvera ett kommando och när användaren bara vill titta sig omkring. Detta p.g.a. att ögonen används både som en in- och ut-modalitet för användaren. Detta kallas för Midas beröring-problemet (Jacob, 1991). Det måste följaktligen finnas ett sätt för användaren att tala om för systemet när ett kommando ska utföras. Tre tekniker har utvecklats för att lösa detta problem, nämligen:

- Fördröjning
- Blinkning
- Användning av en annan inmatningsenhet i kombination med blickmätaren

Fördröjning innebär att ett kommando utförs efter att man har fixerat blicken på ett bestämt område tillräckligt länge. Här är det viktigt att tiden innan ett kommando utförs varken är för lång eller för kort. Är den för lång blir användaren frustrerad och är den för kort är det lätt hänt att felaktiga kommandon exekveras. Att använda blinkningar för att utföra kommandon bör undvikas då det är onaturligt för människan att utföra avsiktliga blinkningar, dessutom måste man hantera problemet att kunna skilja på avsiktliga och oavsiktliga blinkningar. Kombinationen av ögonstyrning och en manuell inmatningsenhet kan vara den bästa lösningen i många sammanhang men fungerar uppenbarligen inte i sammanhang där blickmätaren är den enda inmatningsenheten.

Att utföra något med precision är svårt med ögonen. Med musen är det möjligt att välja små ikoner och även enstaka pixlar. Några av anledningarna till detta problem är de små och skakiga rörelserna som ögonen gör under en fixering samt att det är svårt att med hög precision avgöra var användaren tittar. Därför kan det vara bra att låta en annan inmatningsenhet ta hand om precisionsarbete.

Trots att vi rör våra ögon hela tiden utan några som helst svårigheter kan det vara svårt att avsiktligt utföra något med ögonen då de styrs av ett undermedvetet intresse. När vi avsiktligt ska röra och fixera våra ögon på vissa punkter kan denna ansträngning leda till besvär som bl.a. trötthet. Ytterligare nackdelar med ögonstyrning är att utrustningen fortfarande är förhållandevis dyr, men detta kommer med all sannolikhet att ändras i framtiden när efterfrågan förmodligen kommer att öka. Att man med dagens system behöver göra en kalibrering innan man kan använda ett ögonstyrningsgränssnitt kan ses som en nackdel. Men även det kommer troligtvis att ändras i framtiden.

3.4.4 Gränssnitt med implicit interaktion

I ett ögonstyrningsgränssnitt med implicit interaktion utför användaren inte några explicita kommandon utan gränssnittet försöker ta reda på vad användaren vill genom att observera blicken. Nedan ges några exempel på tillämpningar som använder sig av denna interaktionsform.

Little Prince

Hansen m.fl. (1995) har utvecklat en idé som de kallar *Intresse- och känslo-känslig media* (eng. *Interest and Emotion Sensitive media, IES*) vilket är en implicit interaktionsteknik som bygger på samlad data av användarens ögonrörelser. Man försöker avgöra vad användaren är intresserad av genom att analysera vad det är användaren tittar på. Känslor registreras genom att mäta frekvensen på blinkningar samt användarens pupillstorlek. Denna information används sedan för att ändra egenskaper i gränssnittet. *Little Prince* (Starker & Bolt, 1990) är en tillämpning som använder sig av denna teknik. Den bygger på en

barnbok med samma namn och visar en grafisk modell av en planet där en prins bor samtidigt som en röst berättar historien om denna prins. Om användarens blick riktas mot speciella delar av planeten kommer rösten att berätta mer ingående om just de delarna. Men om användaren betraktar alla delar av planeten lika mycket kommer berättelsen att handla om planeten som en helhet.

World of Windows

Bolt (1981) har skapat ett gränssnitt han kallar *World of Windows*, eller *Värld av Fönster*. Gränssnittet består av en stor skärm som täcker en vägg. På skärmen visas flera fönster vilka innehåller information och som är valbara med blicken. Avsikten är att underlätta för beslutstagare att handskas med stora mängder information. Betraktaren sitter bekvämt i en stol på ett kontor och tittar på skärmen. Systemet organiserar presentationen av fönster genom att använda blicken som en indikation på var betraktaren riktar sin uppmärksamhet. Fönster som får lite uppmärksamhet försvinner helt samtidigt som fönster som får mycket uppmärksamhet blir större.

3.4.5 Gränssnitt med explicit interaktion

I ögonstyrningsgränssnitt som använder sig av explicit interaktion ger användaren direkta kommandon till datorn, som t.ex. *kopiera*, *flytta* och *ta bort*. Traditionella gränssnitt med tangentbord och mus hör till denna kategori. Nedan ges några exempel på gränssnitt och tekniker som använder sig av ögonstyrning med explicit interaktion.

Handikappsgränssnitt

Trots att onaturliga ögonrörelser som avsiktliga blinkningar bör undvikas kan det ibland vara nödvändigt att använda sig av dessa i gränssnitt. För att komman undan Midas beröring-problemet har Kaufman m.fl. (1993) i sitt gränssnitt avsett för svårt handikappade använt sig av blinkningar som en bekräftelse på att ett kommando ska utföras. Gränssnittet skiljer på avsiktliga och oavsiktliga blinkningar genom att endast reagera på blinkningar med ett öga i taget. Oavsiktliga blinkningar sker alltid samtidigt. När användaren tittar på ett objekt på skärmen blir detta markerat, inget mer händer förrän användaren blinkar med ena ögat. Flera blinkningar i följd samt beroende på vilket öga som används kan bestämma vilket kommando som utförs.

EyeCon

EyeCon är namnet på ett grafiskt användargränssnitt vilket använder sig av fördröjningsmetoden för att ge användaren möjlighet att utföra kommandon med ögonen (Velichkovsky & Hansen, 1996). Gränssnittet består av knappar vilka ger information till användaren om hur mycket av fördröjningstiden som har passerat. När användaren fixerar på en knapp kommer denna att visa en animation av ett öga som gradvis stängs. När det animerade ögat har stängts helt kommer knappen att tryckas in och ett objekt som är associerat med knappen kommer att väljas. Användaren kan när som helst avbryta den stundande knapptryckningen under animationens gång genom att låta blicken lämna knappen. Knapparna i EyeCon är placerade bredvid de objekt knapparna associeras med, därför kan användaren

studera objektet i fred utan att behöva oroa sig över att en felaktig knapptryckning utförs.

Quick Glance Selection Method

En ytterligare metod för att välja objekt med ögonen är Quick Glance Selection Method (Ohno, 1998). Här är de objekt som kan väljas uppdelade i två delar, kommandonamn och valruta. När användaren tittar på kommandonamnet visas en symbol i valrutan, om användaren väljer att titta på denna symbol väljs objektet i fråga. På så sätt kan användaren, liksom hos EyeCon ovan, studera kommandonamnet ifred utan att behöva oroa sig för att fel objekt väljs.

MAGIC-pointing

Zhai m.fl. (1999) menar att endast använda ögonen för interaktion med datorn är onaturligt då ögonen är ett av våra främsta perceptuella organ och inte är utvecklade för att kontrollera saker med. De har utvecklat en teknik, MAGIC-pointing (Manual And Gaze Input Cascaded-pointing), som kombinerar ögats snabbhet med musens precision. Det finns två varianter av MAGIC-pointing. Vanligtvis tittar användaren först på objektet han/hon vill klicka på för att därefter föra muspekaren dit. Den första varianten går ut på att muspekaren flyttas direkt till den punkt användaren tittar på då musen sätts i rörelse, förutsatt att muspekaren befinner sig på ett visst avstånd från punkten användaren tittar på. Användaren upplever detta som att muspekaren framträder ur ”tomma intet” i området där han/hon tittar. På så sätt sköts den grova, långa förflyttningen av muspekaren med ögonen och finjusteringen med handen. Detta leder till att den sträcka som pekaren flyttas manuellt med handen blir mycket liten och därmed minskar risken för skador hos användaren. I den andra varianten av MAGIC-pointing flyttas muspekaren till varje nytt klickbart objekt som användaren tittar på. Om pekaren inte är direkt på objektet kan musen användas för att justera den manuellt. Om användarens blick är inom en viss radie från muspekaren påverkar inte blicken muspekarens position.

Intelligent Gaze-added Operating System, IGO

Salvucci och Anderson (2000) hävdar att den brusiga informationen från blickmätaren samt svårigheten att skilja mellan när användaren vill göra något och när han/hon bara vill titta omkring kräver att gränssnittet har en intelligent karaktär. De utvecklade därför *Intelligent Gaze-added Operating System (IGO)*, vilket är ett ögonstyrt, grafiskt användargränssnitt, traditionellt utformat med menyer, ikoner och knappar. Systemet har en knapp som fungerar precis som en musknapp, objektet som påverkas då knappen trycks är det som användaren för stunden tittar på. Exempelvis, för att markera en ikon tittar användaren på ikonen samtidigt som han/hon trycker på knappen. Det naiva förfarandet för att avgöra vilket objekt som användaren tittar på är att välja det objekt som är närmast blickpunkten. Salvucci m.fl. påstår att den naiva algoritmen ofta misslyckas med att göra rätt val, d.v.s. välja det objekt som användaren faktiskt tittar på, p.g.a. brusig data från blickmätaren samt individuella skillnader. IGOs lösning på problemet är en probabilistisk algoritm som tolkar användarens avsikt baserat på två kriterier: var användaren tittar samt i vilket sammanhang han/hon utför en uppgift. Då

systemet ska avgöra vilket objekt använder tittar på viktar det alla objekt nära blickpunkten. Det objekt med högst vikt blir valt. Objekten viktas olika beroende på vad användaren håller på med för tillfället samt vad användaren har gjort tidigare. T.ex. så får stänga-knappen i ett fönster en låg vikt efter att användaren markerat ett objekt i det, då det är mindre troligt att användaren vill stänga fönstret efter en sådan handling.

4 Utveckling

4.1 Problem

Användargränssnitten i moderna operativsystem som exempelvis Windows kräver ofta en hög precision av inmatningsenheten, ibland ända ner till en enskild pixels nivå. Eftersom musen styrs av handen som är bra på motoriskt precisionsarbete är detta inget problem. Blickmätningssystem är ofta inte mer noggranna än 1 grads visuell vinkel (ca 1 cm på skärmen) med alla fel inberäknade. Man måste också tänka på den fysiologiska begränsning som det mänskliga ögat har. P.g.a. foveas utsträckning kan ett blickmätningssystem teoretiskt sett inte avgöra var en person tittar med en precision högre än 0.5 grader. Att försöka lägga på ögonstyrnings-funktionalitet på ett existerande program som exempelvis Microsoft Excel är därför svårt p.g.a. den höga precision programmet kräver av inmatningsenheten. Inom exjobbet gjordes ett test att styra befintliga Windows-tillämpningar via musen. Muspekarens position kopplades till användarens blickpunkt genom ett lågpassfilter som tog bort fel i blickmätarens utdata samt ryckiga rörelser hos ögat. För att klicka användes en separat knappenhet. Denna metod visade sig fungera dåligt då det underliggande gränssnittet som inte är avsett för ögon-styrning inte kunde hjälpa användaren på traven då denne avsåg att trycka på en knapp. P.g.a. en sämre kalibrering och inte perfekt precision hos blickmätnings-systemet hamnade inte muspekaren direkt på den knapp som skulle tryckas in utan ungefär två pixlar utanför, på en yta där musklickningar inte har någon betydelse. Tillämpningen skulle ha kunnat gissa sig till att användaren avsåg den knapp som låg två pixlar ifrån den punkt där musklicket verkligen skedde. Det skulle kunna vara möjligt att sänka upplösningen på skärmen för att öka storleken på de grafiska komponenterna och därmed öka chansen att pricka rätt men detta resulterar fortfarande i ett gränssnitt som inte är anpassat för ögonstyrning samtidigt som man förlorar fördelarna med en högupplöst skärm. Möjligheten att försöka ögon-styra en befintlig tillämpning som är avsedd och designad för musinteraktion är följaktligen begränsad. Därför krävdes det att en ny tillämpning byggdes från grunden upp. Det var inget krav inom arbetet att ögonstyrning skulle vara den enda modaliteten utan det var tillåtet att kombinera med andra inmatningsenheter som t.ex. mus och tangentbord.

Förutom att ta fram ett exempel på hur man kan styra ett grafiskt användargräns-snitt med ögonen var ett ytterligare syfte med projektet att skapa ett generiskt ögonstyrningssystem mot vilket prototyptillämpningen arbetade. Vid utveckling av ett musstyrt användargränssnitt finns det oftast hos de flesta operativsystem en uppsättning funktioner, objekt och färdiga grafiska komponenter som en utveck-lare kan använda sig av. Det kan röra sig om att tillämpningen automatiskt får reda på när muspekaren rör sig över ett fönster eller färdiga knappar som kan tryckas ner m.h.a. musen. Meningen med ögonstyrningssystemet är att det ska fungera som ett ögonstyrt operativsystem och tillhandahålla den basfunktionalitet som behövs för en ögonstyrd tillämpning.

4.2 Metod

Eftersom arbetet var tänkt att resultera i en ögonstyrd version av en existerande programtyp var det lättast att bestämma vilken typ detta skulle vara, för att därefter identifiera de funktioner i användargränssnittet som skulle styras med ögonen. Hur interaktionen med användargränssnittet skulle gå till togs fram under brainstormingmöten för att därefter implementeras, testas och analyseras. Var det något man var missnöjd med ändrades det för att sedan testas igen. Detta förfarande itererades ända tills en tillfredställande lösning hittades. En användarstudie skulle vara det ideala att göra i testningsfasen men utvecklandet tog för mycket tid för att något sådant skulle vara möjligt att genomföra inom projektets ram. Istället fick åsikter från examensarbetaren och anställda hos uppdragsgivaren avgöra i testningen.

Innan någon implementation av det grafiska användargränssnittet kunde ske behövdes det underliggande ögonstyrningssystemet designas och implementeras. Efter att ett förslag på användargränssnittet tagits fram identifierades olika generiska delar och gränssnitt som skulle utgöra ögonstyrningssystemet. Utifrån detta lades sedan en skiss fram på hur det underliggande systemet skulle kunna se ut.

Parallellt med utvecklandet av ögonstyrningssystemet och användargränssnittet skapades prototyptillämpningen. Det kom sig naturligt då det var lika bra att testa interaktionen i den miljö den var tänkt att slutligen användas i. Efter att man nått en tillfredställande lösning var det nödvändigt att gå tillbaka och städa upp efter sig det man gjorde temporärt bara för att testa ifall det fungerade tillfredställande. Den slutliga uppläggningen av genomförandet blev alltså:

- Bestämma vilken typ av tillämpning som prototypen skall vara
- Bestämma vilka interaktionstekniker i tillämpningen som ska kunna styras med ögonen
- Komma upp med idéer till hur dessa tekniker ska kunna styras med ögonen
- Designa generella ögonstyrda grafiska komponenter som utför dessa interaktionstekniker
- Designa ett generellt ögonstyrningssystem som de ögonstyrda grafiska komponenterna kommunicerar med
- Implementera ögonstyrningssystemet och de ögonstyrda grafiska komponenterna
- Använda ögonstyrningssystemet och de grafiska komponenterna till att skapa prototyptillämpningen
- Testa och analysera huruvida funktionaliteten är tillfredställande, om inte modifiera existerande funktionalitet eller tänk ut nya idéer som därefter implementeras och testas. Iterera denna punkt tills en godtagbar prototyp är utvecklad

- När en tillfredställande lösning har tagits fram, se över funktionaliteten, ögonstyrningssystemets gränssnitt samt implementationen och gör eventuella finjusteringar och optimeringar

4.3 Utrustning och miljö

Det blickmätningssystem som användes i arbetet var den hos uppdragsgivaren utvecklade produkten Tobii ET-17, se Figur 4.1. Tekniken den använder är video-baserad pupill- och hornhinnereflektion och består av en TFT-skärm med en kamera monterad undertill samt lysdioder som sänder ut nära infrarött ljus för att åstadkomma reflektionerna i ögonen. Skärmen är kopplad till en dator vilken sköter bildbehandlingen. Resultatet av bildbehandlingen, d.v.s. punkten där systemet tror att användaren tittar, kan antingen användas av en tillämpning som kör på samma dator som bildbehandlingen eller skickas via TCP/IP till en annan dator. I det senare fallet får bildbehandlingen och tillämpningen varsin processor att förfoga över vilket kan vara önskvärt i vissa fall. Prototypen som utvecklades i detta projekt blev aldrig särskilt processorkrävande, därför kunde bildbehand-lingen och prototypen köras på samma dator. ET-17 har en noggrannhet på 0.5 grader visuell vinkel vilket betyder att om man befinner sig på 50 cm avstånd från skärmen kan systemet med en precision på 1 cm avgöra var man tittar på skärmen. Systemet har en uppdateringsfrekvens på 30 Hz och tolererar huvudrörelser inom en volym av 20 x 15 x 15 cm.

Figur 4.1 Den blickmätare som användes i arbetet, Tobii ET-17.

Det gränssnitt en utvecklare av en ögonstyrd tillämpning har mot Tobii ET-17 är enkelt. Trettio gånger i sekunden avgör systemet var användaren tittar och meddelar den ögonstyrda tillämpningen resultatet genom att skicka två par av X-

och Y-koordinater. Koordinaterna talar om vilken punkt på skärmen höger respektive vänster öga är riktat mot. För att få den punkt användaren upplever att han/hon tittar på tas medelvärdet av de båda punkterna.

Ögonstyrningssystemet implementerades i C++ som Microsoft COM-objekt. De grafiska användargränssnittskomponenterna implementerades i C++ som Microsoft ActiveX-kontroller. Anledningen till att välja COM och ActiveX var att objekt baserade på dessa teknologier kan användas från flera olika programmeringsspråk, bl.a. Visual Basic, .Net-språk, Java och C++.

4.4 Val av tillämpningstyp samt funktionalitet

Den typ av tillämpning som den ögonstyrda prototypen skulle representera skulle helst vara en ofta förekommande och frekvent använd programtyp hos medelanvändaren. Prototypen är tänkt att användas i demonstrationssyfte i avseende att visa upp uppdragsgivaren Tobiis produkt ET-17. Då programtypen antagligen redan är bekant för den person som testar prototypen, på en mässa exempelvis, kan han/hon snabbt avgöra om den ögonstyrda versionen har några fördelar gentemot den icke ögonstyrda versionen. De förslag på vanliga tillämpningstyper som togs upp var ordbehandlare, webbläsare samt filhanterare. Det som var viktigt att beakta var vad det är en användare gör med respektive tillämpning. Nedan följer en grov översikt av interaktionstekniker som är karakteristiska för de olika tillämpningarna:

Ordbehandlare

- Hantera menyer
- Trycka på knappar
- Markera text
- Flytta text
- Scrolla

Webbläsare

- Hantera menyer
- Trycka på knappar
- Följa länkar
- Scrolla
- Skriva in URLer

Filhanterare

- Hantera menyer
- Trycka på knappar
- Markera objekt
- Flytta objekt
- Exekvera objekt
- Scrolla
- Navigera i trädstruktur

Man kan här konstatera att scrolla, hantera menyer samt trycka på knappar är gemensamma interaktionstekniker för de tre typerna. Det unika hos ordbehandlaren är att man måste kunna hantera diverse operationer på text. Detta kan vara ett problem vid ögonstyrning p.g.a. precisionen hos blickmätningsutrustningen. Texten i en ordbehandlare är ofta så liten att det kan vara svårt att urskilja vilket av två närliggande ord som användaren tittar på. Den blickmätningsteknik som finns idag kan följaktligen betraktas som något omogen för denna typ av tillämpning. I den andra tillämpningstypen, webbläsaren, är de huvudsakliga interaktionsteknikerna att följa länkar samt att scrolla. Denna funktionalitet ansågs inte tillräckligt omfattande samt erbjöd inte tillräckligt med utmaning för att fungera som mål i ett examensarbete. Det sista förslaget däremot, filhanteraren, betraktades som en mycket god kandidat med många möjligheter till förbättring genom ögonstyrning. Den har något som de andra typerna i viss mån saknar, nämligen hantering av visuella objekt. Det är ett område som är mycket intressant att undersöka om det går att förbättra med ögonstyrning. Samtidigt har en filhanterare behov av att använda menyer, knappar samt scrollning. Att navigera i trädstrukturer är också något de andra tillämpningstyperna inte har och som vore intressant att utreda. Detta var några av de skäl som gjorde att filhanteraren blev den typ av tillämpning som den slutgiltiga prototypen skulle bli.

4.5 Utformning av grafiska komponenter

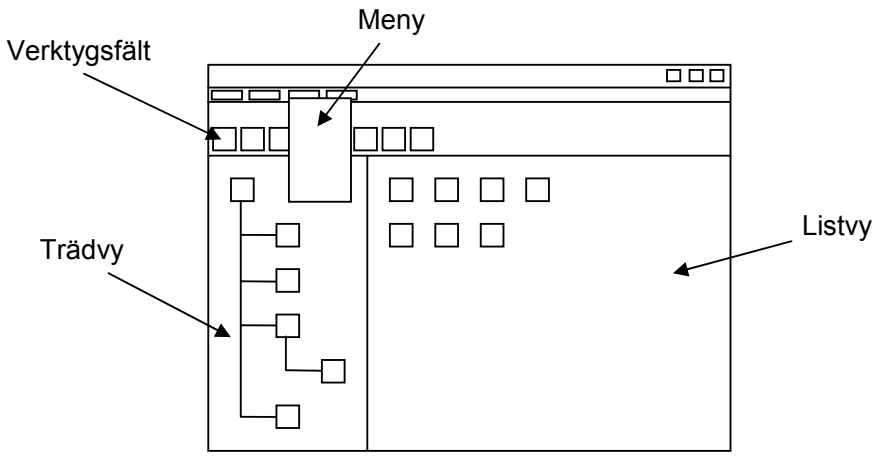
Ett grafiskt användargränssnitt är oftast uppbyggt av flera små specialiserade komponenter. Dessa kan vara knappar, inmatningsfält och menyer för att nämna några. Till den ögonstyrda filhanteraren behövdes ett antal komponenter för att uppfylla filhanterarens uppgift, nämligen att hantera filer. De komponenter som ansågs nödvändiga för filhanteraren listas i Tabell 4.1 tillsammans med de interaktionstekniker varje komponent förknippas med.

Komponent	Interaktionstekniker
Listvy	Markera, flytta samt exekvera objekt, scrolla
Trädvy	Markera och flytta objekt, scrolla
Verktygsfält	Trycka på knappar
Rullgardinsmenyer	Hantera menyer

Tabell 4.1 Komponenterna med tillhörande interaktionstekniker.

Listvyn är den komponent som visar själva filerna och katalogerna under en given katalog. Trädvyn är den komponent som låter användaren få översikt över filsystemet samt navigera i det. Ett verktygsfält är en samling knappar som är grupperade efter funktion. Rullgardinsmenyer är kommandon som grupperas under rubriker och som visas då rubriken aktiveras. Dessa komponenter är

generella och är inte låsta till att användas i en filhanterare. Listvyn, exempelvis, skulle kunna användas till att visualisera objekt i en databas. En schematisk bild av filhanteraren kan ses i Figur 4.2.



Figur 4.2 Schematisk bild av filhanteraren.

4.5.1 Listvyn och trädvyn

En musstyrd listvy brukar ofta kunna visualisera sina objekt på flera olika sätt. Dessa sätt kan innefatta att objekten visas som stora ikoner eller uppradade vertikalt i tabeller med detaljerad information om objekten. Den senare typen av visualisering lämpar sig mindre bra för ögonstyrning då raderna i tabellen måste vara förhållandevis korta på höjden för att en användbar mängd rader skall få plats. Därför valdes den typ där objekten, i det här fallet filer och kataloger, ligger som ikoner i ett rutnät. Då får man plats med objekt på både bredden och höjden och därmed kan man visa fler objekt samtidigt samt ha ett bra avstånd mellan objekten. I en ögonstyrd tillämpning är det alltid positivt att ha ett avstånd mellan objekt för att lättare kunna urskilja vilket objekt som användaren avser. De interaktionstekniker som undersöktes för filhanterarens listvy var markering av filer, flyttning av filer, exekvering av filer samt scrollning.

Markering av fil Det förslag på markering av en fil som togs fram var att användaren först tittar på en fil i listvyn för att därefter på något sätt säga till systemet ”Markera den fil jag tittar på”. Med ögonen finns det inte något naturligt sätt att indikera att något ska hända. Två lösningar testades, dels att använda en knapp av något slag, t.ex. på tangetbordet eller på en extern knappsats, och dels att använda sig av fördröjningstekniken. I den första lösningen tryckte användaren helt enkelt på en knapp för att utföra valet och i den andra markerades filen efter att användaren tittat på filen en kort period.

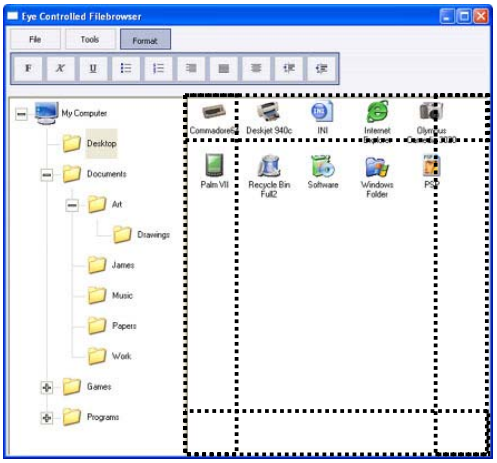
Flyttning av filer Flyttning av filer kan ske inom listvyn och mellan listvyn och trädvyn. Flyttningen av en fil i listvyn sker antingen till en katalog som visas i listvyn eller till en katalog i trädvyn. Två tekniker prövades här. I den första tekniken användes både mus- och ögonstyrning. Den gick ut på att användaren först tittar på filen han/hon vill flytta vilken markeras genom fördröjningstekniken beskriven ovan. Därefter trycker användaren ner en musknapp för att indikera att han/hon vill börja flytta filen för att sedan dra filen med musen till den önskade katalogen och sedan släppa musknappen. Det finns ingen synlig muspekare här och musen används endast som en relativ positioneringsenhet, den börjar röra sig från stället där den markerade filen finns. Den andra tekniken baserades helt på ögonstyrning. Där tittar användaren på den önskade filen samtidigt som han/hon trycker ner en knapp, varpå filen markeras. Så länge den nedtryckta knappen hålls nere kommer nu filen att följa efter användarens blick. Användaren kan nu titta på valfri katalog och sedan släppa upp knappen varpå filen flyttas till denna katalog. Knappen som används kan vara samma knapp som används vid markering av objekt. Då endast markering är önskat sker ingen nerhållning av knappen utan den släpps upp direkt efter att den har tryckts ner.

Expandera och kollapsa kataloger I trädvyn som visar katalogstrukturen i filsystemet bör det finnas ett sätt att expandera och kollapsa kataloger. Att expandera en katalog innebär att alla kataloger direkt under denna visas i trädvyn, och att kollapsa en katalog innebär motsatsen, d.v.s. att gömma en katalogs alla underkataloger. Som beskrivits ovan väljs eller markeras ett objekt genom att först titta på det och sedan trycka på en knapp eller genom fördröjning. Detta förfarande gäller naturligtvis även när man väljer en katalog i trädvyn. Efter att en katalog är vald visas katalogens innehåll i filhanterarens listvy. För att kunna expandera eller kollapsa en katalog i trädvyn placerades en ruta till vänster om varje katalog trädvyn. Med denna kan användaren expandera och kollapsa kataloger på samma sätt som han/hon väljer kataloger, d.v.s. användaren tittar på rutan till vänster om katalogen i trädvyn han/hon vill expandera/kollapsa (beroende på katalogens tillstånd) samtidigt som han/hon trycker på en knapp eller väntar på att fördröjningsperioden ska förflyta.

Exekvera filer Objekten som visas i filhanterarens listvy kan exekveras på olika sätt. Objekten som visas är olika typer av filer samt kataloger. Att exekvera ett objekt innebär här att utföra en handling som är knuten till objektet. När man exekverar en katalog innebär det att katalogen öppnas och innehållet i katalogen visas i listvyn. Då en godtycklig fil exekveras startar ett program som är förknippat med filens typ. I musstyrda miljöer sker detta oftast med dubbelklick. Den tekniken utnyttjades även här. Användaren tittar helt enkelt på filen eller katalogen som han/hon vill exekvera och utför samtidigt ett snabbt dubbelklick med en knapp. Samma knapp som används för att markera/välja och flytta objekt kan här användas till att exekvera objekt.

Scrollning Filhanterarens listvy speglar innehållet i en katalog i filsystemet. Ibland kan det hända att innehållet i en katalog inte får plats i listvyn, den innehåller helt enkelt fler filer än vad listvyn klarar av att visa. Samma situation

kan uppstå i trädvyn, hela kataloghierarkin kan inte visas samtidigt i den grafiska komponenten. Storleken på listvyn och trädvyn kan naturligtvis ändras både vid design av en tillämpning samt under körning av den samma för att rymma fler objekt. Men den nämnda situationen kan aldrig helt undvikas på detta sätt utan måste lösas genom något annat förfarande. I musstyrda miljöer är detta löst genom scrollning, användaren drar m.h.a. musen i ett ”handtag” till höger om listvyn upp och ner varpå innehållet i listvyn följer efter. Hur kan då scrollning fungera i ögonstyrd tillämpning? Den metod som testades var följande. Först behandlas scrollning i vertikal ledd. Då användaren märker att det finns fler filer i listvyn att se än de som visas för tillfället tittar han/hon längst ner i listvyn, i ett litet område innanför kanten, varpå listvyns innehåll åker upp och visar det som finns nedanför där användaren tittade. Om användaren fortsätter att hålla blicken vid listvyns nedersta kant så kommer scrollningen att fortsätta. Meningen är att då den fil som användaren söker efter kommer att rulla upp i listvyn så kommer användarens blick att fästas vid denna fil och därmed lämna det område som startar scrollningen.



Figur 4.3 Filhanteraren med listvyns scrollområden markerade.

Eftersom listvyn ska gå att scrolla upp, ner, vänster och höger så finns det alltså ”scroll-områden” längs listvyns alla fyra kanter, se Figur 4.3. Vill man scrolla åt vänster tittar man på vänster sida, vill man scrolla uppåt tittar man på den övre delen o.s.v. Tanken är att scrollningen ska upplevas som att den sker automatiskt, som att datorn vet när användaren vill scrolla. Denna metod för scrollning implementerades även i trädvyn.

4.5.2 Verktygsfält och menyer

Menyers roll i musstyrda tillämpningar är att spara plats genom att gruppera och gömma kommandon bakom en sorts knappar, eller klickbara fält, som representerar dessa kommandon. Fälten brukar ligga i en horisontell rad i övre delen av programmets fönster och bär oftast namn som säger något om vilken typ

av kommandon som ligger under fältet ifråga. När användaren klickar på ett fält fälls själva menyn ner i vilken användaren kan välja bland flera kommandon. Verktygsfälten i musstyrda tillämpningar ligger vanligtvis under menyn och består av en eller flera rader med ikoner. Dessa ikoner fungerar som knappar och utför ett kommando då användaren klickar på en. Dessa kommandon är nästan uteslutande speglingar av de som finns i menyerna. Meningen med verktygsfälten i musstyrda tillämpningar är att erbjuda en snabbare väg för användaren till kommandon som används ofta. Att hitta ett kommando i en meny är en relativt långsam handling.

Med vetskapen om att ögat är mycket snabbare än musen föddes tanken på att eliminera både menyer och verktygsfält och istället ha något som är en blandning av de båda. Resultatet blev ögonverktygsfältet. Ögonverktygsfältet består av två horisontella rader med knappar, se Figur 4.4. Den övre raden är analog med de övre fälten i en meny, när användaren aktiverar en knapp i den övre raden visas i den undre raden knappar som associeras med den aktiverade knappen.

Figur 4.4 Ett ögonverktygsfält.

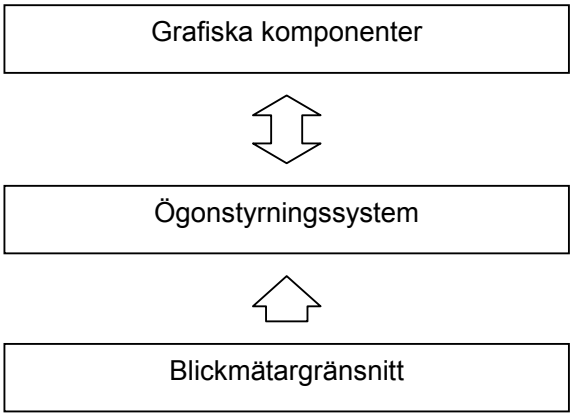
Varje knapp i den undre raden utför ett kommando som t.ex. ”ångra” eller ”kopiera”. Den undre raden kommer hela tiden att ockuperas av en rad med knappar, dessa fälls inte in efter att användaren har använt en av dem eller valt att inte använda dem. Däremot kommer uppsättningen av knappar på den undre raden att ändras då användaren aktiverar knappar på den övre raden. Anledningen till att ha den undre raden ständigt synlig har att göra med hur aktiveringen av knapparna på den övre raden sker. En knapp på den övre raden aktiveras nämligen så fort användaren tittar på den, ingen fördröjningsmekanism eller knapp används alltså. Detta för att valet av kommando ska gå så fort som möjligt. Att utföra en handling genom att bara titta på objekt är inom ögonstyrning oftast inte att rekommendera eftersom användaren lätt kan utföra en oönskad handling (Midas beröring-problemet, se avsnitt 3.4.3). I ögonverktygsfältets fall är detta inget problem då det enda oönskade som kan ske är att knapparna på den undre raden byts ut mot andra. Om inte den undre raden hos ögonverktygsfältet hade en reserverad plats utan t.ex. skulle ligga ovanpå listvyn och trädvyn, vilka ligger rakt under ögonverktygsfältet i filhanteraren, samt att den gömdes efter att användaren inte tittat på den ett tag skulle det vara irriterande för användaren att så fort han/hon oavsiktligt snuddade med blicken vid ögonverktygsfältets övre del så skymdes

listvyn och trädvyn av en rad med knappar. I det fallet skulle det vara bättre att aktivera knapparna på den övre raden med fördröjning eller en extern knapp. Denna metod skulle alltså vara långsammare men ta mindre plats hos tillämpningen. Båda dessa metoder prövades under utvecklingen.

4.6 Ögonstyrningssystemet

4.6.1 Arkitektur

Tanken med ögonstyrningssystemet är att det ska vara ett lager mellan blickmätaren och den ögonstyrda tillämpningen. Ögonstyrningssystemet ska förenkla för utvecklaren genom att hantera sådant som är gemensamt för alla ögonstyrningstillämpningar och agera som medlare mellan ögonstyrningskomponenter i en tillämpning, se Figur 4.5. Utvecklaren ska dels kunna använda sig av en uppsättning färdiga ögonstyrningskomponenter som är byggda ovanpå ögonstyrningssystemet samt dels kunna bygga sina egna ögonstyrda komponenter vilka också arbetar mot ögonstyrningssystemet. Ett av designmålen med ögonstyrningssystemet var att så mycket funktionalitet som möjligt skulle ligga i systemet och endast det som var komponentspecifikt skulle behövas implementeras i själva komponenten. Det lager som ögonstyrningssystemet har under sig är blickmätartillverkarens gränssnitt för att komma åt ”rå” blickdata. Dessa blickdata består dels av blickens position på skärmen men innehåller även information om ögonens position i blickmätarens synfält, ögonens avstånd till blickmätaren samt pupillstorlek. Inom detta projekt användes endast informationen om blickens position på skärmen.



Figur 4.5 Ögonstyrningssystemets roll i en ögonsstyrd tillämpning.

Ett grundläggande beslut som togs i tidigt skede var att göra hela systemet händelsebaserat. D.v.s. en komponent meddelas av ögonstyrningssystemet när

något har inträffat. De huvudsakliga händelser som en ögonstyrd komponent är intresserad av är:

- Då användarens blick träder in över komponenten
- Då användarens blick lämnar komponenten

För att meddelas om när detta händer krävs det först att komponenten registrerar sig hos ögonstyrningssystemet så att denna vet vilka komponenter den ska meddela. Ögonstyrningssystemets huvudsakliga roll är att förmedla dessa två händelser till komponenten ifråga. Det gör den genom att hela tiden hålla reda på vilken komponent som användaren för närvarande tittar på. Endast en komponent i taget kan ha blicken på sig.

För att kunna avgöra om användaren tittar på en av de komponenter som är registrerade i ögonstyrningssystemet krävs det att systemet vet komponentens storlek och position. Denna information får den från komponenten genom ett gränssnitt som är grundläggande för hela ögonstyrningssystemet, IGazeRegion, se Figur 4.6. Alla komponenter som är intresserade av händelser från ögonstyrningssystemet skall implementera gränssnittet IGazeRegion och sedan skicka med en referens till det då de registrerar sig. Nedan kommer den svenska motsvarigheten till IGazeRegion, blickregion, att användas i texten. Blickregion syftar mer på objektet som implementerar IGazeRegion än gränssnittet i sig.

```
interface IGazeRegion
{
      void OnGazeEnter(Point2D gazePos);
      void OnGazeOver(Point2D gazePos);
      void OnGazeLeave();

      Rectangle GetRectangle();
};
```

Figur 4.6 Pseudokod för gränssnittet IGazeRegion som representerar en blickregion.

Metoderna OnGazeEnter och OnGazeLeave kallas på blickregionen när blicken träder in över den respektive lämnar den. Argumentet gazePos är en två-dimensionell punkt som anger var på blickregionen som ögonstyrningssystemet estimerar att användaren tittar. OnGazeOver kallas för varje nytt blickdata som ankommer till ögonstyrningssystemet. Med den blickmätare som användes för projektet, Tobii ET-17 vilken mäter blicken i 30 Hz, kallas alltså denna metod 30 gånger i sekunden. Anledningen till detta är att låta komponenten få uppdaterad information om användarens blickpunkt. En blickregion har formen av en rektangel och uppger sin position och utsträckning via metoden GetRectangle. En komponent kan självklart välja att implementera fler än ett IGazeRegion-gränssnitt, t.ex. om den har en komplex struktur och kan delas upp i flera mindre

områden. För enklare komponenter som t.ex. knappar behövs det bara en blickregion per knapp.

Hur avgör då ögonstyrningssystemet var användaren tittar? Den information som kommer från blickmätaren, d.v.s. användarens blickpunkt, innehåller dels brus samt tenderar att hoppa omkring lite även då blicken fixeras på en punkt. Det senare är en följd av de små, skakiga rörelser som ögat hela tiden gör (Jacob, 1991). P.g.a. av dessa störningar är det inte lämpligt att i en tillämpning använda den blickpunkt som blickmätaren genererar som direkt angivelse om var användaren tittar. De data som kommer från blickmätaren bör behandlas på något sätt för att åstadkomma en stadigare blickpunkt som sedan kan användas i tillämpningen. Det vanligaste sättet att göra detta är med ett fixationsfilter. Ett fixationsfilter analyserar data från blickmätaren över tiden för att detektera om användaren fixerar på något. Utdata från fixationsfiltret är en punkt som är relativt stadig över tiden jämfört med de skakiga och obehandlade blickpunktsdata från blickmätaren.

4.6.2 Fixationsfiltret

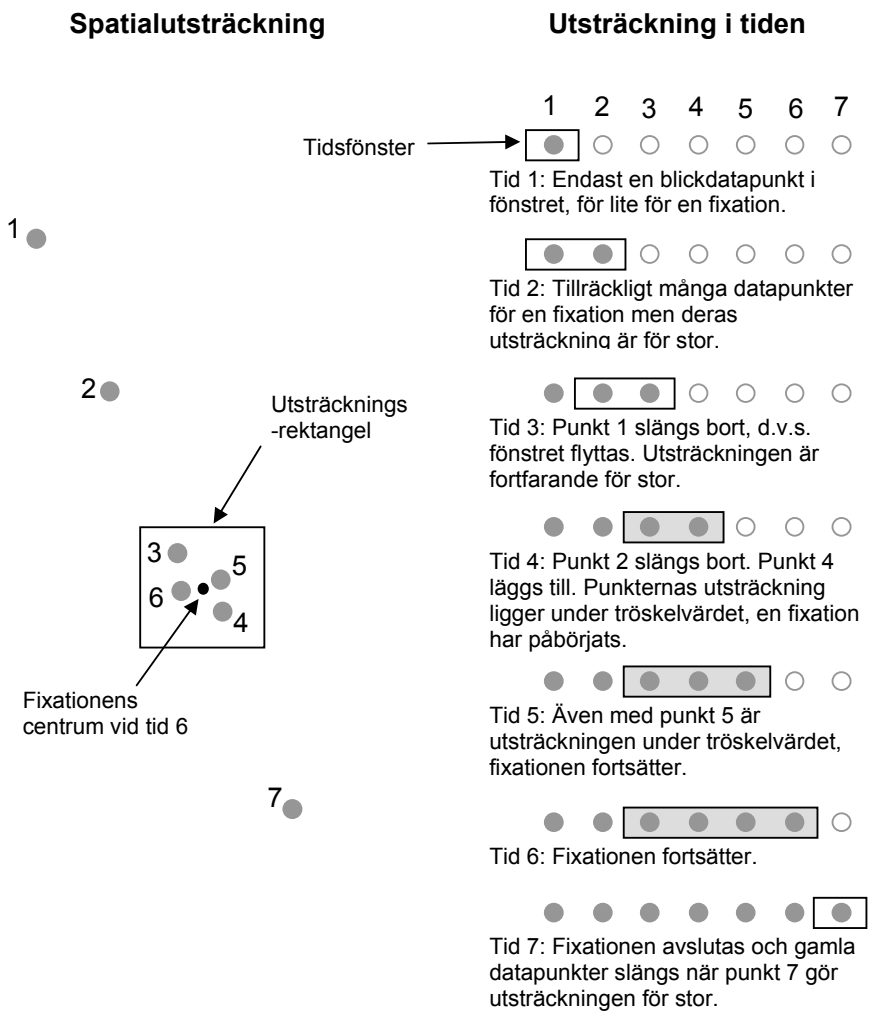
En fixation är som tidigare nämnts en ögonrörelse som stabiliserar blicken på ett stationärt objekt. Fixationer är den ögonrörelse som är mest relevant att använda/detektera i detta projekt. 90 % av den tiden man tittar utgörs av fixationer (Duchowski, 2003). När användaren fixerar på något i gränssnittet betyder det att han/hon är intresserad av vad som finns där. Den näst vanligaste ögonrörelsen, sackaden, innebär bara att användaren väljer att fixera på något annat . Under tiden en sackad pågår kan användaren inte heller tillgodogöra sig någon information (Duchowski, 2003). Möjligtvis kunde man ha användning av att detektera den ögonrörelse som uppkommer då ögat följer ett objekt som rör sig mjukt. Denna ögonrörelse kan ses som den rörliga motsvarigheten till en fixation. Men det är bara betydelsefullt att fånga den om gränssnittet skulle innehålla rörliga objekt, vilket inte är fallet i denna filhanterare.

Identifiering av fixationer är en statistisk beskrivning av observerade ögonrörelser som har till uppgift att översätta de råa blickpunkterna till fixeringspunkter. Enheten som gör detta i en tillämpning kallas för fixationsfilter. Det råder delade meningar vetenskapligt sett om när fixationer startar respektive slutar. Därför är identifieringen av fixationer, oavsett precisionen och flexibiliteten hos identifikationsalgoritmen, en subjektiv process. Således är ett effektivt sätt att validera sina algoritmer att jämföra de fixationsresultat man fick med betraktarens subjektiva intryck. Identifieringen av fixationer förenklar mycket för en ögonstyrd tillämpning, eftersom den inte behöver ta hänsyn till allt det brus som finns med i de råa data från blickmätaren. Då mycket information filtreras bort är det viktigt att detta görs på ett korrekt sätt. Det är viktigt att fixationsfiltret är korrekt också p.g.a. att fixationerna är den enda indata som en tillämpning använder som rör användarens blick. Dåliga fixationsfilter kan t.ex. producera för många eller för få fixationer, eller vara för känslig för skräpdata, d.v.s. data från när blickmätaren misslyckas att korrekt fastställa blickpunkten.

Det finns många olika tekniker för att identifiera fixationer, dessa kan klassificeras som antingen hastighets- eller spridningsbaserade algoritmer. I hastighetsbaserade algoritmer används hastigheten mellan två på varandra följande punkter som ett mått på om det är en sackad eller fixation. Är hastigheten över ett visst tröskelvärde befinner man sig i en sackad annars i en fixation. Denna typ av algoritm har dock visat sig vara svag beträffande robusthet (Salvucci & Goldberg, 2000), vilket också visade sig i tester som gjordes inom examensarbetet. Examensarbetaren valde att använda sig av den andra typen av identifieringsalgoritm, då den inte lider av samma robusthetsproblem samt har visat sig ha en högre noggrannhet än den hasighetsbaserade (Salvucci & Goldberg, 2000).

Den andra typen av fixationsidentifikationsalgoritmer, spridningsbaserade algoritmer, använder sig av vetskapen om att fixationspunkter, p.g.a. sin låga hastighet, tenderar att bilda små grupper. Algoritmerna identifierar fixationer som grupper av konsekutiva punkter med en viss maximal spridning. Härnäst följer en beskrivning av den algoritm som användes i ögonstyrningssystemet.

Algoritmen använder sig av ett flyttbart tidsfönster som sträcker sig över konsekutiva datapunkter och letar efter potentiella fixationer. Då fixationer typiskt varar minst 100 millisekunder används ett tröskelvärde på 100-200 millisekunder som representerar minsta varaktighet hos en fixation. Fönstret börjar med att spänna över ett minimum av punkter som bestäms av detta varaktighetströskelvärde. Därefter tittar algoritmen om punkterna i fönstret har en spatial utsträckning som är mindre än ett visst förbestämt värde, utsträckningströskelvärdet. Utsträckningen kan bestämmas på flera olika sätt. Man kan t.ex. behandla x- resp. y-led separat och se om antingen max(x) – min(x) eller max(y) – min(y) är mindre än tröskelvärdet. Då bildas en rektangel som punkterna testas mot. Ett annat alternativ är att använda sig av den spatiala variansen. I detta projekt användes den förra tekniken. Om utsträckningen visar sig vara större än tröskelvärdet representerar inte fönstret en fixation och den äldsta punkten tas bort samtidigt som en ny punkt adderas, m.a.o. fönstret flyttas. Om punkternas utsträckning däremot är mindre än tröskelvärdet anser man att fönstret representerar en fixation. Därefter utvidgas fönstret med nästkommande blickpunkter ända tills punkternas utsträckning sträcker sig över tröskelvärdet. Fixationens centrum beräknas som medelvärdet av alla punkterna i fixationen. Eftersom denna algoritm används i realtid beräknas centrumet för varje ny punkt som kommer från blickmätaren för att därefter skickas till tillämpningen. Ett exempel på identifiering av en fixation kan ses i Figur 4.7.



Figur 4.7 Beskrivning steg för steg hur fixationsfiltret detekterar en fixation med hjälp av tidsfönster och utsträckning. Filtrets tillstånd vid sju tidspunkter visas. Vid varje tidspunkt kommer ny blickdata från blickmätaren.

Algoritmen har följaktligen två parametrar, ett tröskelvärde för minsta tillåtna varaktighet samt ett för maximal tillåten utsträckning. Det bästa sättet att komma fram till dessa är att testa olika värden och göra subjektiva bedömningar. Tröskelvärdet för minsta varaktighet är teoretiskt mellan 100 – 200 millisekunder. Till ET-17 passade ett tröskelvärde på 70 millisekunder bra. Eftersom ET-17 är ett 30 Hz system hinner man få två blickpunkter innan man kan avgöra om det är en fixation eller ej. Två punkter kan låta lite men visade sig fungera bra. Man ska försöka hålla tröskelvärdet så lågt som möjligt för att så snabbt som möjligt kunna meddela tillämpningen att en fixation har börjat. Ju fler punkter filtret tillåts samla

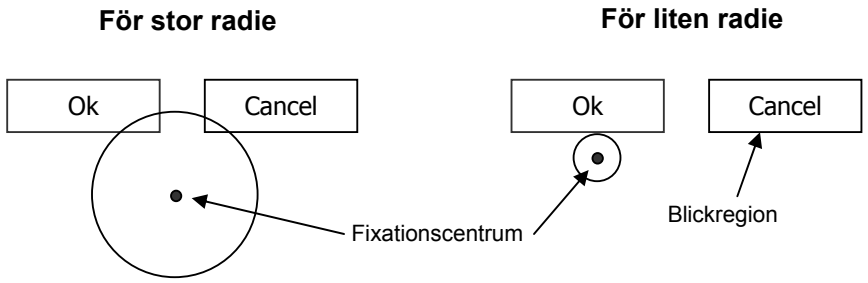
in innan den avgör om det är fixation desto säkrare blir den på att det är en fixation, men den blir samtidigt långsammare. I ett användargränssitt är det viktigt att responsen på användarens ögonrörelser är snabb. Därför sattes varaktighets-tröskeln till ett så lågt värde som möjligt.

Den andra parametern, tröskelvärdet för maximal tillåten utsträckning hos punkterna som utgör fixationen, ska teoretiskt sett vara mellan 0,5˚ - 1˚ visuell vinkel (Duchowski, 2003). Det är inom det området som ögats mikrosackader äger rum under en fixation. Till fixationsfiltret i filhanteraren användes, som tidigare nämnts, en rektangel för att avgöra om punkterna tillhör en fixation. En sida hos denna rektangel är alltså utsträckningströskelvärdet, och rektangelns centrum är medelvärdet av alla punkter i fönstret, d.v.s. fixationens position. Är denna rektangel för liten blir fixationens varaktighet för kort p.g.a. att punkter som ligger i fixationens periferi inte betraktas som medlemmar av fixationen och fixationen avslutas. Detta behöver inte vara ett problem i ett användargränssnitt, däremot kan det vara det i analyssammanhang. Att rektangeln är för stor är dock ett problem i ett användargränssnitt. Om användaren först fixerar på ett relativt litet objekt och därefter på ett likadant som ligger precis bredvid det första kommer fixationsfiltret, om rektangeln är tillräckligt stor, att tolka det som en och samma fixation. Detta eftersom användarens andra fixation hamnade innanför rektangeln för den första fixationen. Resultatet blir att fixationens centrum kommer att glida mot punkterna i den andra fixationen. Även vad det gäller denna parameter krävdes testning för att hitta en tillfredställande lösning.

Det bör noteras att denna algoritm är anpassad för realtidsbruk. I efterbearbetning av blickpunktsdata kan man på ett mer korrekt sätt och med högre noggranhet identifiera fixationer. T.ex. i lösningen ovan ändras fixationens centrum allteftersom filtret får mer och mer information att bygga sina beräkningar på jämfört med om hela fixationens data finns tillgängligt från början, då kan det korrekta centrumet beräknas direkt.

4.6.3 Val av blickregion

För varje ny blickpunkt som kommer från blickmätaren krävs det att systemet hittar den blickregion som användaren för närvarande tittar på, om någon alls. Det som först händer med blickpunkten från blickmätaren är att den behandlas av fixationsfiltret. Om fixationsfiltrets utdata talar om att en fixation pågår eller precis har börjat används fixationens centrum till att hitta blickregionen. Fixationens centrum får utgöra centrum hos en cirkel med en viss radie. Därefter går ögonstyrningssystemet igenom sin interna lista med registrerade blickregioner och testar varje blickregion för skärning mot denna cirkel. Metoden att använda en cirkel istället för en punkt löser en del av de precisions problem som blick-mätare kan ha. Det innebär att fixationens centrum inte behöver vara precis på den blickregionen som användaren tittar på utan den kan vara en bit utanför. Hur långt utanför bestäms av radien hos cirkeln. En för stor radie medför att en blickregion kan bli felaktigt vald medan en för liten radie har till följd att systemet blir alltför restriktivt och kräver att blickmätaren har hög precision, se Figur 4.8.



Figur 4.8 Exempel på för stor respektive för liten radie vid val av blickregion.

Eftersom en cirkel används innebär det att det kan bli konflikter där fler än en blickregion skär cirkeln. Här finns det två alternativ, antingen väljs ingen av regionerna eller så försöker systemet ta reda på vilken region som är troligast att användaren avser. Den metod som implementerades i ögonstyrningssystemet var följande. Om cirkelns mitt, d.v.s. fixationens centrum, ligger innanför en blickregion väljs denna region. Om cirkelns mitt inte ligger på en blickregion men cirkeln skär flera blickregioner väljs ingen. I det senare fallet är det viktigt att den skärande cirkeln är tillräckligt liten. Om cirkeln är för stor kommer den att ofta skära flera blickregioner samtidigt och därmed inte att välja någon.

4.6.4 Återkoppling

Det kan vara önskvärt för användaren att få bekräftat att ögonstyrningssystemet har hittat rätt blickregion, d.v.s. den blickregion som representerar den komponent som användaren tittar på. Detta är något som varje enskild komponent skulle kunna implementera. Exempelvis skulle en knapp kunna byta bakgrundsfärg när användaren tittar på den. Men då ett syfte med projektet var att skapa ett ögonstyrningssystem som skulle förenkla för utvecklaren så mycket som möjligt byggdes det in återkopplingsfunktionalitet i systemet som alla komponenter skulle kunna använda sig av. Med samma typ av återkoppling för alla komponenter får en ögonstyrd tillämpning en enhetlighet som inte skulle uppnås om varje komponent skulle använda sin egen återkoppling.

Hur ska då återkopplingen fungera? Att använda någon form av visuell återkoppling var det enda tänkbara alternativet. Att använda ljud som återkoppling skulle förmodligen bara vara irriterande och skulle inte säga någonting om vilken komponent som användaren tittar på utan bara att användaren tittar på någon komponent. Den första varianten av återkoppling som prövades var att en rektangel ritas ut runt den blickregion användaren för tillfället tittar på. Detta visade sig dock vara mindre bra. Ögat styrs ofta av omedvetna impulser och när något nytt föremål uppenbarar sig i ögats synfält drar det gärna till sig ögats uppmärksamhet. Alltså, när användaren riktar sin blick mot en komponents yta varpå en rektangel uppträder runt denna yta så vill ögat gärna undersöka denna nya form. Det leder till att användaren tittar i komponentens utkant. Med en inte

perfekt precision hos blickmätaren kan användarens fixation hamna så långt utanför komponenten att den inte längre är den valda komponenten. Det kan t.o.m. vara så att en närliggande komponent blir vald istället. Att ha en återkoppling som dyker upp i komponentens periferi är således ingen bra idé. Det som vore önskvärt istället är en återkoppling som drar till sig ögats uppmärksamhet och som centrerar blicken över komponenten så att risken att ögonstyrningssystemet väljer fel komponent minimeras. En återkoppling i form av en liten punkt som uppenbarar sig i mitten av en komponent skapades därför. För att göra den ännu mer tilldragande för ögat gjordes den att pulsera storleksmässigt.

Med en blickmätningsutrustning som kan spåra ögonen perfekt finns det ingen användning för återkoppling. Eftersom ögonstyrningssystemet då exakt skulle kunna avgöra var användaren tittar behövs det ingenting som talar om för användaren var han/hon tittar, det vet han/hon redan.

4.6.5 Övrig funktionalitet

Huvudfunktionaliteten i ögonstyrningssystemet är att tala om för blickregioner när användaren tittar respektive inte tittar på dem. Förutom detta kan det finnas annan information en tillämpning skulle vilja ha. Nedan redovisas det för övrig funktionalitet hos ögonstyrningssystemet.

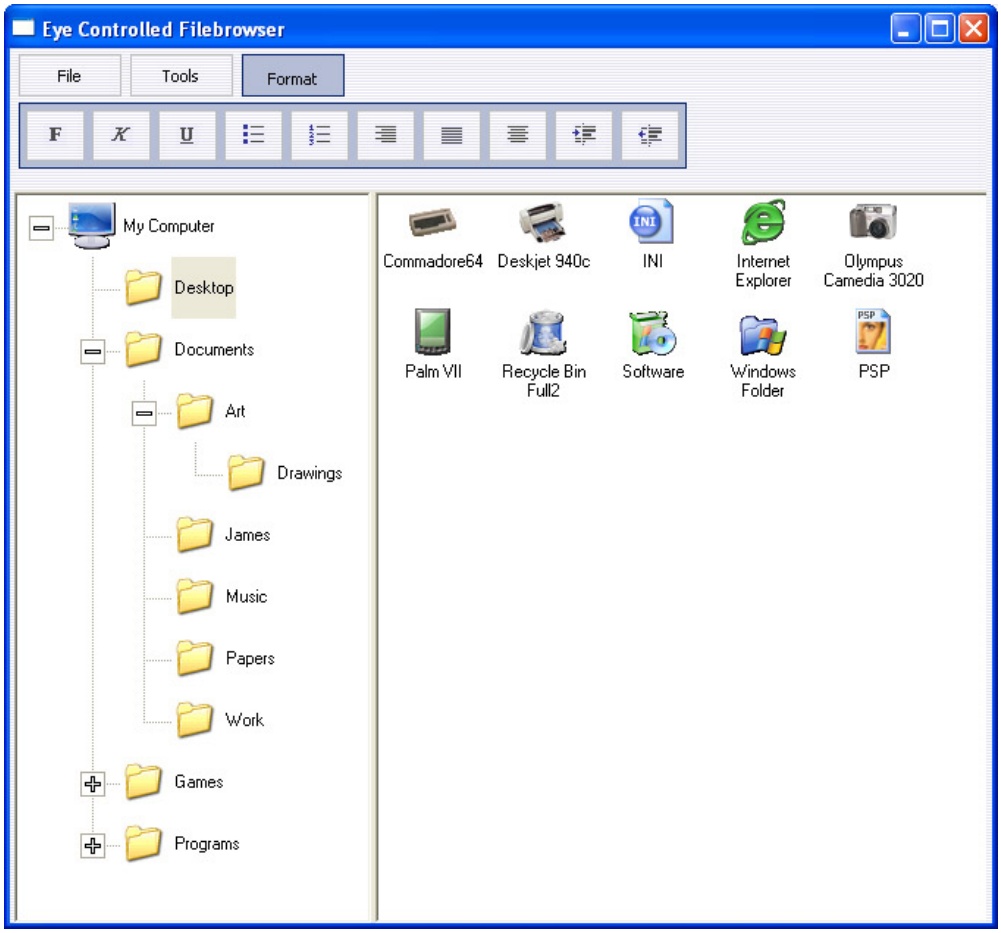
Systemet exponerar händelser från fixationfiltret som skickas då en fixation startar, pågår och slutar. Vidare så är arkitekturen hos systemet sådan att fixationfiltret implementerar ett speciellt fixationsfiltergränssnitt. På så vis kan en utvecklare skapa ett eget fixationsfilter som implementerar detta gränssnitt och säga till ögonstyrningssystemet att använda detta fixationfilter istället.

Att utföra ett kommando i en ögonstyrd miljö sker oftast genom knapptryckning eller fördröjning. Ögonstyrningssystemet erbjuder funktionalitet för båda metoderna. Systemet exponerar fyra olika händelser vilka var och ett är kopplade till en knapp hos en inmatningsenhet, t.ex. hos tangentbordet. Dessa händelser representerar fyra olika generiska knappar, man kan säga att de motsvarar musknapparna hos ett vanligt operativsystem. När systemet märker att en av dessa knappar har tryckts in skickas en händelse till den blickregion som användaren för tillfället tittar på. Genom att ha denna funktionalitet inbyggd i systemet istället för att varje enskild komponent ska hålla reda på sin egen inmatningsenhet gör inte bara att inmatningen hanteras enhetligt utan också att samspelet mellan blicken och inmatningen kan kontrolleras om det skulle behövas. För att använda fördröjningsmetoden kan varje enskild blickregion be ögonstyrningssystemet att få en händelse då användaren har tittat på regionen en viss tid.

5 Resultat och analys

5.1 Prototypen

De idéer om hur de grafiska komponenterna skulle styras visade sig fungera som väntat i de flesta fallen. Nedan diskuteras de observationer som gjordes då den ögonstyrda prototyptillämpningen testades av anställda hos uppdragsgivaren samt av examensarbetaren. Referera gärna till avsnitt 4.5 där det redogörs för utformningen av komponenterna. Figur 5.1 visar hur den färdiga filhanteraren såg ut.



Figur 5.1 Den färdiga filhanteraren.

5.1.1 Listvyn och trädvyn

En av de grundläggande frågorna var om fördröjningstekniken eller användning av en knapp var den bästa metoden för att utföra ett kommando. Det finns dock

ingen metod som är den rätta, utan det beror på kontexten vilken metod som kan/bör användas. Vid markering av en fil eller katalog fungerade det utmärkt med fördröjning då det kommando som utförs inte är något som ger upphov till någon stor förändring som är svår att ångra. Det enda som händer är att en fil markeras. Råkar användaren avmarkera en fil genom att titta på en annan är det bara att titta på den första filen igen för att ”rätta till felet”. Ett bra värde på tiden det tar innan en fil markeras efter att användaren börjat titta på den visade sig vara 150-250 ms. Det är en relativt kort period som inte tvingar användaren att stirra för länge på en fil innan den markeras. Att markera med en knapp fungerade också bra. Den tekniken kunde användas då användaren inte ville ”vänta” på att fördröjningen skulle utföra markeringen utan visste direkt att han/hon ville markera filen. De två teknikerna kunde alltså samexistera. Men upplevde man fördröjningstekniken irriterande fanns det möjlighet att stänga av den.

Som tidigare nämnts prövades två olika tekniker för flyttning av filer, en med kombinerad ögon- och musstyrning samt en med enbart ögonstyrning. Tanken med den första tekniken var att ögat är dåligt på precisionsarbete medan musen är bättre lämpat för sådant samtidigt som det kunde vara irriterande att ha filen följandes blicken. Det visade sig dock att den andra tekniken fungerade mycket väl. Då man i den första tekniken först ”sett ut” den tänkta destinationskatalogen för att sedan dra musen dit behöver man i den andra tekniken bara släppa knappen efter att ha sett ut den tänkta katalogen. Filens rörelse efter blicken kunde lösas på flera tillfredsställande och icke störande sätt. Ett sätt är att låta filen endast följa fixationer. Ett annat är att filtrera blickpunkten på ett sådant sätt att filen får en fördröjd och mjuk rörelse så att den under flyttningen aldrig ligger ivägen för blickpunkten utan hela tiden är en liten bit efter. Teknik nummer två ansågs vara överlägsen den första och fick därför bli den som behölls i listvyn och trädvyn.

Att exekvera filer var ganska rättfram. Här vill man inte använda fördröjning därför att en felaktig exekvering resulterar oftast i besvärliga konsekvenser som att ett program startar exempelvis. Därför användes en knapp istället. Anledningen till att använda dubbelklickning var att det var möjligt att använda samma knapp som används för att markera/välja och flytta objekt. Användningen av en och samma knapp till all funktionalitet i filhanteraren gjorde den mer intuitiv och lättanvänd för användaren.

Scrollnings-funktionaliteten var någonting som var svårt att få bra. I den första versionen var de områdena hos komponenten som startade scrollningen diskreta, d.v.s. om användaren tittade i området scrollade komponentens innehåll, om inte användaren tittade där scrollades det inte. En bättre metod var att tillåta steglös scrollningshastighet. När användaren tittar i ett scrollområde avgörs scroll-hastigheten av avståndet från blickpunkten till komponentens kant. Ju närmare kanten desto högre scrollhasighet. Vid scrollområdets kant som ligger närmast mitten av komponenten är scrollhastigheten noll. Denna metod fungerade bättre eftersom användaren har möjlighet att styra hastigheten. Dock var det inte fråga om att explicit styra scrollhastigheten genom att titta på specifika positioner utan

tanken var mer att ”ju närmare komponentens kant användaren tittar desto mer intresserad är han/hon av att se mer”.

5.1.2 Ögonverktygsfältet

Ögonverktygsfältet var en komponent som var tänkt att ersätta menyer och vanliga musstyrda verktygsfält. Två metoder prövades här. I den första aktiverades den övre radens knappar så fort blicken kom på dem och den undre raden med knappar hade ett dedikerat utrymme. I den andra metoden användes fördröjning eller knapptryckning till att aktivera den övre raden, vidare hade den undre raden inget eget utrymme utan skymde listvyn och trädvyn under. Efter att en knapp hade tryckts eller efter en tids inaktivitet gömdes den undre raden. Den första metoden infriade alla förväntningar och uppvisade en överlägsenhet i snabbhet gentemot den första. Men var det inte irriterande att den övre radens knappar aktiverades så fort användaren tittade på dem? Nej, eftersom ögonstyrningssystemet endast säger till en komponent att den har användarens blick på sig först efter att en fixation har påbörjats så sker inga onödiga aktiveringar bara för att användarens blick sveper (gör en sackad) förbi den övre raden. Den plats som den undre raden tog upp ansågs inte vara någon stor uppoffring i jämförelse med den snabbhet som erhölls. Inga fördelar med den andra metoden kunde urskiljas varpå den första metoden fick bli den som behölls.

5.2 Ögonstyrningssystemet

Att implementera de grafiska ögonstyrningskomponenterna med hjälp av ögonstyrningssystemet var enkelt och smärtfritt. Systemet erbjöd den basfunktionalitet som behövdes och det enda som behövdes läggas till var den komponentspecifika koden. Även den funktionalitet som ögonstyrningssystemet erbjöd i form av information om när fixationer börjar och slutar samt den inbyggda fördröjnings-mekanismen användes till komponenterna.

Parametrarna i systemet, bl.a. de i fixationsfiltret och för valet av blickregion, fastställdes på empirisk väg. Resultatet var för användaren ett system som svarade snabbt och kändes robust. Ögonstyrningssystemet byggdes inte initialt för att klara av stora mängder registrerade blickregioner utan målet var att *en* tillämpning kunde använda det. Om flera tillämpningar med många blickregioner tillsammans skulle använda systemet skulle det behövas en ny algoritm som letar efter den blickregion som användaren tittar på. Detta p.g.a. att den implementerade algoritmen skulle vara för långsam. Systemet är dock byggt att skala väl så ett sådant ingrepp skulle inte kräva alltför mycket jobb.

Den visuella återkoppling som implementerades i systemet fungerade bra och kändes inte irriterande. Genom att göra den pulserande punkten transparent skymde den inget av användargränssnittet. Precisionen hos dagens blickmätningssystem skiljer sig från person till person p.g.a. att egenskaperna hos ögonen är svårare att mäta hos vissa människor. Därför fungerade filhanteraren bättre för en del personer än för andra. För en del krävdes det att ha återkoppling så att de var säkra på att systemet tolkade deras blick rätt. Men för den grupp som fick bra

precision hos blickmätaren behövdes ingen återkoppling. De användarna fick så pass stort förtroende för systemet att de inte behövde verifiera att systemet hade hittat rätt blickregion innan de utförde ett kommando. Att inte använda återkoppling är en stor skillnad för användaren. Det upplevs som att systemet kan läsa användarens tankar.

6 Slutsats

I ett ögonstyrt system är kanske den viktigaste delen själva blickmätaren. En blickmätare med dålig precision och robusthet kan ingen ögonstyrningslogik i världen rädda. Därför var det tillfredställande att i projektet arbeta med en blickmätare som hade hög precision och robusthet. Samtidigt så är dagens blickmätare inte perfekta och därför behövs det i en ögonstyrd tillämpning logik som bl.a. fixationsfilter som strävar efter att ta bort fel i de data som kommer från blickmätaren. Därtill finns det ett behov av att delarna i en ögonstyrd tillämpning arbetar mot ett gemensamt system som fungerar som en medlare mellan dessa. Detta system bör vara generellt så att det kan återanvändas i flera tillämpningar. Dessa kriterier uppfyller det ögonstyrningssystem som utvecklades inom examensarbetet. I förlängningen kan man tänka sig att detta system kan vara en naturlig del av ett operativsystem. Lika naturligt som att det i operativsystemet finns funktioner för att utnyttja musen i sin tillämpning ska det finnas funktioner för ögonstyrning.

Den prototyp som utvecklades gav ett exempel på hur en tillämpning skulle kunna styras med ögonen. Att det var en tillämpning designad för att styras med ögonen, till skillnad från en musstyrd tillämpning där musen styrdes med ögonen, var ett starkt skäl till att den fungerade så bra som den gjorde. Den mest innovativa delen i prototypen var sannolikt ögonverktygsfältet. Den tog tillvara på den egenskap hos ögat som inte kan matchas av någon annan inmatningsenhet, nämligen hastigheten. Eftersom uppsättningen av möjliga kommandon hos ögonverktygsfältet ändras så fort blicken hamnar på någon av knapparna på den övre raden gör att användaren snabbt kan välja mellan en stor mängd kommandon. Detta resultat är svårt, om inte omöjligt, att uppnå med ett musstyrt användargränssnitt. Noteras bör att implementationen av flera av interaktionsteknikerna hos filhanteraren hade stora likheter med den traditionella musstyrningen, t.ex. markering och exekvering av fil. Vid markering exempelvis pekar användaren först ut objektet i fråga, med ögonen eller musen, för att därefter utföra markeringen med en knapptryckning. Anledningen till att det fungerade bra var att ögonstyrningssystemet innehöll logik som korrigerade för felen i blickmätarens data och kunde ”hjälpa till” när den tolkade blickpunkten inte var exakt på komponentens yta. Interaktionstekniken att flytta en fil liknar även den sin motsvarighet i en musstyrd miljö, men här togs det sista steget bort i den ögonstyrda varianten. Det sista steget, att dra filen till sin destination, behövdes inte i ögonstyrningsfallet då användaren redan hade valt destinationen då han/hon letade efter den med ögonen. Genom att modifiera befintliga interaktionstekniker för musstyrda gränssnitt går det alltså att skapa väl fungerande ögonstyrda diton. Den interaktionsteknik hos filhanteraren som utnyttjade någon form av implicit interaktion var scrollningsfunktionaliteten hos listvyn och trädvyn. Där tolkades blickens avstånd till kanten på komponenten som ett mått på hur intresserad användaren är av att se det som finns gömt bortom kanten. När blicken kom tillräckligt nära kanten började innehållet i komponenten att scrolla och scrollningshastigheten ökade ju närmare blicken kom i förhållande

till kanten. Prototypen visar att det går utmärkt att kombinera explicit och implicit interaktion. Samtidigt som användaren utför en explicit handling med blicken kan även blicken implicit påverka tillämpningen. Ett exempel på detta i filhanteraren är då användaren under flyttningen av en fil letar efter destinationskatalogen genom att scrolla. Då är flyttningen av filen den explicita interaktionen och scrollningen utgör den implicita.

Detta examensarbete har karaktär av både datalogi och människa-datorinteraktion (MDI). Ögonstyrningssystemet har utgjort den datalogiska delen och utformningen av ögonstyrningskomponenterna har utgjort MDI-delen. Det ideala hade varit att bara fokuserat på en del. Då hade mer tid kunnat läggas på att utreda vissa frågor mer. Men då de båda delarna är beroende av varandra var det inte möjligt. För att utveckla ögonstyrningskomponenter är man tvungen att ha ett underliggande ögonstyrningssystem av något slag och för att testa hur väl ögonstyrningssystemet fungerar krävs komponenterna. Därför utvecklades båda delarna inom examensarbetet. Men att utveckla båda delarna samtidigt var även positivt då man som utvecklare hade kontroll över hela processen. Eftersom gränssnittskomponenterna designades först kunde man därefter designa ett ögonstyrningssystem som mötte alla de krav som ställdes av komponenterna. Fann man att det saknades stöd för något som behövdes av en komponent var det bara att implementera detta i ögonstyrningssystemet. Att båda delarna skapades inom projektet kan också ha bidragit till att viss funktionalitet hos komponenterna blev av överhuvudtaget. Skulle ögonstyrningssystemet varit låst och begränsat kanske en del saker inte hade varit möjliga att genomföra.

7 Rekommendationer

Ögonstyrningssystemet som utvecklades i det här examensarbetet innehöll först och främst endast funktionalitet som täckte behovet hos komponenterna i prototyptillämpningen. Framtida ögonstyrningskomponenter kan tänkas vara i behov av annan funktionalitet. Ett intressant område är s.k. implicit interaktion, där tillämpningen försöker lista ut vad det är användaren är intresserad av genom att studera hans/hennes ögonrörelser. Filhanterarens komponenter använde till största delen explicit interaktion, d.v.s. användaren talar om för tillämpningen vad det är den ska utföra. Ett ögonstyrningssystem skulle kunna erbjuda en uppsättning filter som kan detektera vissa beteenden hos användaren. Dessa filter skulle exempelvis kunna detektera när användaren läser eller är förvirrad.

En begränsning ögonsystemet har är att det inte klarar av överlappande blickregioner. Det är något som bör lösas i framtiden. En annan förbättring skulle kunna vara att tillåta blickregioner med icke-rektangulära former, t.ex. ellipser och polygoner. Som nämnts är inte det utvecklade ögonstyrningssystemet byggt för att klara av stora mängder blickregioner vilket skulle kunna krävas när flera tillämpningar skulle använda det. En flaskhals är sökningen efter den blickregion som användaren för tillfället tittar på. Idag görs sökningen linjärt, vilket fungerar för ett hundratal registrerade blickregioner. För att fler blickregioner inte ska ha en negativ inverkan på systemets reaktionstid bör blickregionerna sorteras på lämpligt sätt för att underlätta sökningen.

Prototypen som skapades var visserligen ögonstyrd med en del nya koncept, dock var många idéer lånade från den musstyrda versionen. Det var lite av ett medvetet val att inte ta ett för stort första kliv i jakten på det perfekta ögonstyrningsgränssnittet. Ett av målen med exjobbet var som bekant att ta fram ett exempel på en ögonstyrningstillämpning. Därför vore det intressant att i framtida forskning ta fram helt nya koncept hur datorn kan styras med ögonen som inte har någon motsvarighet i den musstyrda världen.

Källförteckning

Bolt, R. A.1981. ”Gaze-orchestrated dynamic windows”, *Proceedings of the 8th annual conference on Computer graphics and interactive techniques,* s. 109-119

Duchowski, A. T. 2002 ”A Breadth-First Survey of Eye Tracking Applications”, *Behavior Research Methods, Instruments, and Computers,* vol. 34, nr. 4, s. 455-470

Duchowski, A. T. 2003. *Eye Tracking Methodology, Theory and Practice,* Springer-Verlag, London

Hansen, J. P., Andersen, A. W. & Roed, P. 1995. ”Eye-gaze control of multimedia systems”, *Proceedings of HCI International*, s. 6-9

Jacob, R. J. K. 1995 "Eye Tracking in Advanced Interface Design" *Virtual Environments and Advanced Interface Design*, s. 258-288

Jacob, R. J. K. 1991 "The Use of Eye Movements in Human-Computer Interaction Techniques: What You Look At is What You Get," *ACM Transactions on Information Systems*, vol. 9, nr. 3, s. 152-169

Kaufman A., Bandopadhay, A. & Shaviv, B. 1993 "An Eye Tracking Computer User Interface" *Research Frontiers in Virtual Reality Workshop Proceedings*, October 1993, s. 120-121

Ohno, T. 1998. ”Features of eye gaze interface for selection tasks”, *The Third Asia Pacific Computer Human Interaction*

Salvucci, D. D. & Anderson, J. R. 2000. ”Intelligent gaze-added interfaces”, *Proceedings of the SIGCHI conference on Human factors in computing systems*, s.273-280

Salvucci, D. D. & Goldberg, J. H. 2000. ”Identifying fixations and saccades in eye-tracking protocols”, *Proceedings of the symposium on Eye tracking research & applications 2000*, s.71-78

Starker, I. & Bolt, R. A. 1990 ”A gaze-responsive self-disclosing display”, *Proceedings of the SIGCHI conference on Human factors in computing systems,* s. 3-10

Velichkovsky, B. M. & Hansen, J. P. 1996. ”New technological windows into mind: there is more in eyes and brains for human-computer interaction”, *Proceedings of the SIGCHI conference on Human factors in computing systems*, s. 496-503

Vertegaal, R., Slagter, R., van der Veer, G. & Nijholt, A. 2000. ” Why conversational agents should catch the eye”, *Conference on Human Factors in Computing Systems*, s. 257-258

Yamato, M., Inoue, K., Monden, A., Torii, K. & Matsumoto, K., 2000. ”Button selection for general GUIs using eye and hand together”, *Proceedings of the Working Conference on Advanced Visual Interfaces*, s. 270 - 273

Zhai, S., Morimoto, C. & Ihde, S. 1999. ”Manual and gaze input cascaded (MAGIC) pointing”, *Proceedings of the SIGCHI conference on Human factors in computing systems: the CHI is the limit*, s. 246-253