Tampereen ammattikorkeakoulu
Tietojenkäsittelyn koulutusohjelma
Mika Wilén

Opinnäytetyö

Tietokonenäkö havaitsevissa käyttöliittymissä

Työn ohjaaja FL Paula Hietala
Työn tilaaja Flander SW Oy, valvojana linjapäällikkö Ville Tuominen
Tampere 4/2009

Tampereen ammattikorkeakoulu
Tietojenkäsittelyn koulutusohjelma

Tekijä	Mika Wilén
Työn nimi	Tietokonenäkö havaitsevissa käyttöliittymissä
Sivumäärä	43
Valmistumisaika	Huhtikuu 2009
Työn ohjaaja	Paula Hietala
Työn tilaaja	Flander SW Oy

TIIVISTELMÄ

Opinnäytetyössä tutkitaan tietokonenäön hyödyntämistä havaitsevissa käyttöliittymissä ja esitellään konstruktiivisena osuutena toteutetun tietokonenäön MotionKit-kirjaston rakennetta ja toimintaa. Kirjaston käyttöä havainnollistetaan sen mahdollisuuksia esittelevillä käyttöliittymäprototyypeillä. Opinnäytetyö on laadittu Flander SW Oy:lle.

Opinnäytetyön idea syntyi teknologialähtöisesti vuonna 2008, kun Qt-osaamisen kysyntä kasvoi räjähdysmäisesti ja toteutettiin harrastusluonteisena projektina tietokonenäkösovellus Qt:llä ja OpenCV-kirjastolla. Valmiidenkin tietokonenäön algoritmien käyttö tuntui tarpeettoman hankalalta ja sai pohtimaan yksinkertaista apukirjastoa, joka mahdollistaisi tietokonenäön hyödyntämisen ilman algoritmien tuntemista tuoden sen siten laajemman käyttäjäkunnan ulottuville. Tämä antoi opinnäytetyölle myös uuden käytännönläheisemmän näkökulman, jolla se poikkeaa aihepiirin algoritmikeskeisistä ja lähes matemaattisista tutkimuksista.

Opinnäytetyössä tutkittiin ihmisen ja tietokoneen vuorovaikutuksen teoriaa ja tunnistettiin nykyisten käyttöliittymien heikkous: niitä ei ole suunniteltu tehtäväkeskeisiksi ja läpinäkyviksi, eikä vuorovaikutuksessa hyödynnetä ihmisen täyttä kommunikointikapasiteettia. Tietokonenäön todettiin mahdollistavan käyttäjän liikkeitä ja eleitä tarkkailevat käyttöliittymät, jotka vievät vuorovaikutuskokemuksen lähemmäksi ihmisen luontaisia taipumuksia.

Opinnäytetyön aineistona käytettiin ihmisen ja tietokoneen vuorovaikutusta ja käyttöliittymien teoriaa käsitteleviä lähteitä, Qt:n ja OpenCV:n perusteoksia sekä kirjoittajan omaa 9 vuoden työkokemusta ohjelmistotuotannosta.

Avainsanat Tietokonenäkö, käyttöliittymä, OpenCV, Qt

TAMK University of Applied Sciences
Business Information Systems

Writer	Mika Wilén
Thesis	Computer Vision in Perceptual User Interfaces
Pages	43
Graduation time	April 2009
Thesis Supervisor	Paula Hietala
Co-operating Company	Flander SW Oy

ABSTRACT

In this thesis, the use of computer vision in perceptual user interfaces is studied. The thesis includes describing the design and the functionality of a computer vision helper library called MotionKit which was implemented as a part of this thesis work. MotionKit usage is explained through examples of user interface prototypes which demonstrate the potential of the library.

The idea for this thesis originated from the sudden increase in the demand of Qt-experts in 2008. This inspired to implement a computer vision application as a recreational project for learning the Qt and OpenCV-library. Implementing computer vision was very challenging, even with the algorithms provided by the OpenCV. This raised the idea about a very simple helper library, which would make it possible to use computer vision without knowing about algorithms, thus making it reachable by a larger developer base. In this practical aspect, this thesis differs from the usual mathematical and algorithm-centred researches about the subject.

The thesis begins with the theory of human-computer interaction and with identifying the weaknesses in the current user interfaces: they do not focus on the task, they are not transparent and they do not take the full advantage of the human interaction capacity. Computer vision makes it possible to observe the user's movements, thus helping the interaction experience to meet the natural human habits.

The resources for this thesis include literature about human-computer interaction, user interfaces, Qt and OpenCV, in addition to the 9 years of software development experience of the writer.

Keywords Computer vision, User interface, OpenCV, Qt

SISÄLLYSLUETTELO

1 Johdanto 5

2 Ihminen ja tietokone 7
2.1 Vuorovaikutus 7
2.2 Käyttöliittymä 8
2.3 Tietokonenäkö 10
2.4 Havaitseva käyttöliittymä 11

3 Qt 13
3.1 Yleistä Qt:stä 13
3.2 Qt:n rakenne 14
3.3 Käyttöliittymä 15
3.4 Grafiikka 16
3.5 Qt Designer 17

4 OpenCV 19
4.1 Yleistä OpenCV:stä 19
4.2 OpenCV:n rakenne 20
4.3 Kuvan käsittely 21
4.4 Kuvan muunnos 23
4.5 Histogrammi 25
4.6 Objektin tunnistus ja seuranta 26
4.7 Koneellinen oppiminen 27

5 MotionKit 28
5.1 Yleistä MotionKit:istä 28
5.2 MotionKit:in rakenne 29
5.3 Toiminta 30
5.4 Alustus 31
5.5 Kuvan lukeminen videolähteestä 33
5.6 Kuvan analysointi 34
5.7 Tuloksen palautus käyttäjäsovellukselle 35

6 Käyttöliittymäprototyyppejä 36

7 Loppusanat 39

Lähteet 42

Liitteet 43
Liite 1: Kuvaus käyttöliittymäprototyyppien videotallenteista 43
Liite 2: Käyttöliittymäprototyyppien videotallenteet [DVD]

1 Johdanto

Tutkin opinnäytetyössäni tietokonenäön hyödyntämistä havaitsevien käyttöliittymien ohjelmoinnissa. Tavoitteena on toteuttaa opinnäytetyön konstruktiivisena osuutena tietokonenäön palveluita tarjoava MotionKit-kirjasto, joka helpottaa tietokonenäön soveltamista käyttöliittymäohjelmointiin ja jolla pyritään siten edesauttamaan uusien käyttöliittymäinnovaatioiden syntymistä. Konstruktiiviseen osuuteen kuuluvat myös MotionKit-kirjaston käyttöä esittelevät käyttöliittymäprototyypit, jotka demonstroivat kirjaston tarjoamia mahdollisuuksia. Tavoitteena on myös dokumentoida opinnäytetyössä käytetyt teknologiat ja toteutus riittävän tarkasti, jotta lukija saisi tehdystä toteutuksesta ja sen tarjoamista palveluista selkeän käsityksen.

Tietokonenäkö on aktiivinen tutkimuskenttä, ja sen aihepiiristä löytyy useita tutkimuksia tämänkin opinnäytetyön avainsanoilla. Näissä vahvasti teoriapainotteisissa ja matemaattisissa tutkimuksissa keskitytään usein tiettyihin algoritmeihin ja esitellään niiden ympärille rakennettua toiminnallisuutta. Tässä opinnäytetyössä on otettu käytännönläheisempi lähtökohta: ydinajatuksena on tuottaa apukirjasto, jonka avulla käyttöliittymäohjelmoijat voivat hyödyntää tietokonenäköä havaitsevien käyttöliittymien ohjelmoinnissa ilman tietokonenäön algoritmien tuntemista. Tämä mahdollistaa keskittymisen tietokonenäköä hyödyntäviin käyttöliittymäkonsepteihin tietokonenäön toteutuksen sijaan.

Työn toimeksiantajana on Flander SW Oy, jossa olen työskennellyt vuodesta 2000 lähtien mobiiliohjelmistotuotantoon liittyvissä kehitys- ja tutkimustehtävissä Symbian-asiantuntijana. Flander on 1997 perustettu ohjelmistotestaukseen ja -kehitykseen keskittynyt kasvuyritys, joka tarjoaa jalostettuja ohjelmistopalveluita ja -prosesseja sulautettuihin ja muihin vaativiin ratkaisuihin. Flanderin asiakkaita ovat tietoliikenne-, teollisuus-, IT- ja palvelualojen yritykset. Flander työllistää yhteensä 500 henkilöä Suomessa, Ruotsissa ja Kiinassa.

Opinnäytetyöni aihe hahmottui teknologialähtöisesti vuoden 2008 aikana, jolloin Qt-osaamisen kysyntä kasvoi räjähdysmäisesti ja halusin vahvistaa asemaani muuttuvilla työmarkkinoilla. Suunnittelin ja toteutin tuolloin harrastusluonteisen pilottiprojektin, jossa selvitin Qt:n ja OpenCV:n soveltuvuutta eleohjaukseen. Pilottisovelluksessa

käyttäjän kasvoja kuvattiin, ja niiden sijainti ja etäisyys projisoitiin reaaliaikaisesti 3D-näkymään ohjaamaan kameran sijaintia. Esitin pilottisovelluksen konseptin avulla työnantajalleni opinnäytetyöni aiheeksi tietokonenäön apukirjastoa, jota voitaisiin käyttää Qt-osaamisen referenssinä, ajankohtaisten teknologioiden itseopiskelumateriaalina ja tietenkin havaitsevien käyttöliittymien toteuttamiseen ohjelmistokehityshankkeissa matalalla oppimiskynnyksellä.

MotionKit-kirjaston toteutukseen valitut teknologiat ovat helposti perusteltavissa. Trolltechin Qt-käyttöliittymäkirjasto on tämän hetken kysytyintä osaamista visuaalisesti näyttävien käyttöliittymien kehityksessä ja Intel OpenCV puolestaan on tietokonenäön algoritmeja tarjoava kirjasto, jonka optimoidut algoritmit mahdollistavat tietokonenäön suorituskykyisen käytön ilman kymmeniä tuhansia koodirivejä, vaikka sen käyttö edellyttääkin algoritmien ymmärtämistä. Kummatkin kirjastot toimivat yleisimmissä Linux-, Windows- ja Mac-käyttöjärjestelmissä, joten teknologioiden valintaa puoltaa paitsi niiden ajankohtaisuus ja kattavat ominaisuudet, myös niiden soveltuvuus monelle alustalle.

Opinnäytetyön aineisto perustuu käytettyjen teknologioiden perusteoksiin, käyttöliittymiä ja ihmisen ja tietokoneen vuorovaikutusta käsitteleviin lähteisiin, sekä kirjoittajan omaan työkokemukseen ohjelmistojen suunnittelusta, käyttöliittymäohjelmoinnista ja graafisista algoritmeista.

Tämä opinnäytetyöraportti on kokonaisuutena suunnattu tietojenkäsittelyn ammattikorkeakoulutason tiedot hallitsevalle lukijalle, mutta tietyt osat tekstistä, kuten OpenCV-kirjastoa käsittelevä luku 4, ovat syventävää aineistoa tietokonenäöstä kiinnostuneelle lukijalle, joka jo tuntee grafiikkaohjelmointia ja haluaa tietää tarkemmin esimerkiksi MotionKit-kirjaston toteuttamisessa käytetyistä algoritmeista.

2 Ihminen ja tietokone

Tässä luvussa käsitellään ihmisen ja tietokoneen vuorovaikutusta, käyttöliittymiä sekä tietokonenäköä johdatukseksi havaitsevien käyttöliittymien tarjoamiin mahdollisuuksiin. Opinnäytetyön konstruktiivisena osuutena toteutettu MotionKit-kirjasto on saanut keskeiset lähtökohtansa tässä luvussa esiteltävästä teoriasta ja pohdinnasta. Ihmisellä ja tietokoneella on omat rajoituksensa ja vahvuutensa, joiden ymmärtäminen on toimivan vuorovaikutuksen suunnittelun perusedellytyksiä (Abowd, Beale, Dix & Finlay 2004, 194).

2.1 Vuorovaikutus

Teoksessa Human-Computer Interaction (Abowd ym. 2004) on otettu vuorovaikutuksen tarkastelun lähtökohdaksi ihminen ja tietokone itsenäisinä systeemeinään, joilla on tiettyjä vuorovaikutuksellisia ominaisuuksia.

Ihmisellä on kyky vastaanottaa ja lähettää tietoa useilla I/O-kanavilla: visuaalisella kanavalla, äänellisellä kanavalla, tuntokanavalla ja liikekanavalla. Ihminen tallentaa tietoa pika- tai lyhytmuistiinsa tai pitkäkestoiseen muistiinsa, jotka ovat kaikki virhealttiita. Ihminen osaa prosessoida tietoa järkeilemällä ja ratkomalla ongelmia, mutta on näissäkin toiminnoissa altis virheille ja ulkopuolisille vaikutuksille. Toistuvat samankaltaiset tilanteet mahdollistavat ihmisen taitojen kehittymisen tietyllä alueella tietorakenteiden määritysten parantuessa, mutta tämäkin voi johtaa virheisiin tilannekontekstin vaihtuessa. Ihmisen havainnointikyky ja kognitiivisuus ovat monimutkaisia ja hienostuneita, mutta niillä on rajoituksensa ja yksilökohtaisia eroja. (Abowd ym. 2004, 55.)

Tietokoneella on myös useita I/O-kanavia, kuten näppäimistö, hiiri, puheentunnistus, kosketuslevy, ohjainlaitteet ja kosketusnäyttö, jotka palvelevat kahta tehtävää: tekstinsyöttöä ja osoittamista. Tietokoneen näyttö toimii pääasiallisena lähtökanavana, mutta ääntäkin hyödynnetään. Tietokoneen ajonaikainen muisti vastaa ihmisen lyhytkestoista muistia ja magneettiset sekä optiset mediat pitkäkestoista muistia, mutta nämä eivät ole suorituskyvyllisesti rinnastettavissa ihmisen muistiin. Tietokoneen prosessointinopeus voi vaikuttaa vuorovaikutuskokemukseen, ja sillä on useita teknisiä rajoittavia tekijöitä, kuten laskentakyky, muistin nopeus, grafiikan käsittelyn nopeus ja verkkoviiveet. (Abowd ym. 2004, 120 - 121.)

Näiden kahden systeemin vahvuuksien ja rajoitusten ymmärtäminen on yksi niiden välisen vuorovaikutuksen suunnittelun perusasioita. Nämä systeemit eivät ole myöskään vuorovaikutustilanteessa samanarvoisia, vaan on tärkeää huomata, että vuorovaikutus suunnitellaan käyttäjän tarpeisiin, sillä tietokonehan vain suorittaa ihmisen antamaa tehtävää. Vuorovaikutuksen suunnittelun ydin on asettaa käyttäjä edelle, pitää hänet keskipisteenä ja muistaa hänet lopussa (Abowd ym. 2004, 195).

Tehokkaassa vuorovaikutuksessa tulisi hyödyntää mahdollisimman kattavasti ihmisen I/O-kanavia eikä kuormittaa liikaa rajallisia ja virhealttiita ominaisuuksia, kuten muistia. Tietokoneen I/O-kanavia taas on mahdollista laajentaa tehostamaan vuorovaikutusta: tämänkin opinnäytetyön ydinajatuksena on antaa käyttöliittymäsovellukselle näkökyky, jolla se voi havainnoida käyttäjäänsä ja edesauttaa siten vuorovaikutusta.

Kahden ihmisen välinen keskustelu kasvokkain on teknisesti primitiivistä, mutta kommunikaatiokeinoiltaan rikkaimpia mahdollisia vuorovaikutusmekanismeja. Siinä käytetään esimerkiksi puhetta, kuuloa, ruumiinkieltä, eleitä, ilmeitä ja katsekontaktia (Abowd ym. 2004, 476 - 478). Viestintää voidaan tehostaa jopa fyysisellä kontaktilla, jos on oikein painokasta sanottavaa. Tämän kaiken rinnalla ihmisen viestintä tietokoneelle pelkän näppäimistön ja hiiren avulla vaikuttaa ihmisen kommunikaatiokapasiteetin selvältä alikäytöltä. Rajoittavana tekijänä ihmisen ja tietokoneen vuorovaikutuksessa on käyttöliittymä.

2.2 Käyttöliittymä

Tietokoneen käyttöliittymien kehitys on ollut verkkaista. 60-luvulle asti käytetyt reikäkortit ovat jääneet jo historiaan, mutta sen jälkeen kehitetyt käyttöliittymien muodot ovat vieläkin käytössä. Näistä merkittävimpiä ovat komentorivipohjaiset käyttöliittymät (CLI, Command-Line Interface) ja WIMP:iin (Windows, Icons, Menus and Pointers) perustuvat graafiset käyttöliittymät, joita käytetään yleisesti Windows-, Mac- ja UNIX-pohjaisten käyttöjärjestelmien käyttöliittymissä (Abowd ym. 2004, 136 - 142).

WIMP-pohjaisissa käyttöliittymissä on tiettyjä peruselementtejä, kuten sen nimestäkin löytyvät ikkunat, ikonit, valikot ja osoittimet. Näihin elementteihin lukeutuvat myös

painikkeet, työkalupalkit ja dialogit. Näistä WIMP-käyttöliittymän elementeistä käytetään yleisnimitystä widget (Abowd ym. 2004, 145).

On yllättävää, miten vähän tietokoneiden käyttöliittymät ovat muuttuneet tietokoneiden suorituskyvyn ja oheislaitevarustelun kehittyessä. Nykyaikaista tietokonetta, josta löytyy web-kamera, tv-kortti, äänikortti, nopea verkkoyhteys ja suuri laskentakapasiteetti, mutta jonka käyttöliittymässä hyödynnetään vain näppäimistöä ja hiirtä, voitaisiin verrata komentorivipohjaisella käyttöjärjestelmällä varustettuun tietokoneeseen, johon kytkettyä hiirtä käytetään vain etäisyyksien mittaamiseen pöydän pinnalta. Teknisiä rajoitteita seuraavaan kehitysaskeleeseen WIMP-käyttöliittymästä eteenpäin ei ole, vaan kyse on urautumisesta vakiintuneisiin käyttöliittymäkonsepteihin.

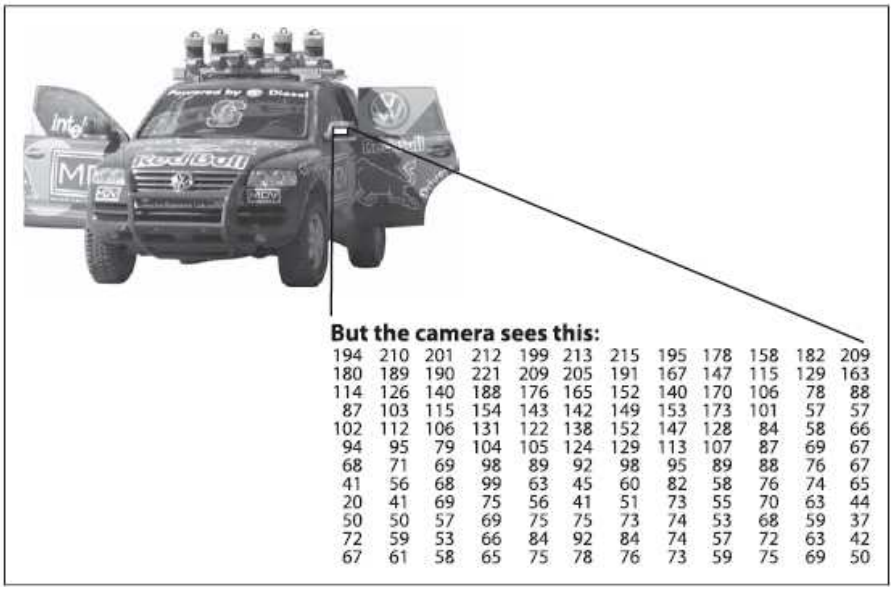
Donald Arthur Norman pureutuu ongelman ytimeen kokoelmateoksessa The Art of Human-Computer Interface Design (1990) perustellessaan Mac-käyttöjärjestelmän käyttöliittymän erinomaisuutta: *”And the development of the internal toolbox has caused developers to be consistent in their use of screen, mouse and keyboard, even when their natural tendencies would have led them elsewhere.”* (Norman 1990, 209). Käyttöliittymien kehityksessä pidetään yhä lujasti kiinni totutuista käytännöistä, standardeista ja yhtenevyyksistä luontaisten taipumusten kustannuksella. Hyvänä puolena on se, että kerran omaksuttu käyttöliittymä toimii muissakin sovelluksissa. Huonona puolena taas on kehityksen pysähtyminen käytäntöihin, jotka eivät ole ideaalisia vuorovaikutuksen kannalta. Millainen sitten olisi luontaisia taipumuksia mukaileva käyttöliittymä? Normanin vision mukaan se on käyttöliittymä, joka sulautuu niin saumattomasti itse tehtävään, että se katoaa tietoisuudesta (Norman 1990, 219).

Yhtenä esimerkkinä luontaisia taipumuksia mukailevasta käyttöliittymästä voisi olla tietokoneen työpöytä, jota voidaan tilan loppuessa laajentaa oikealle. Nykyisessä Ubuntu-käyttöjärjestelmässä työpöytää laajennetaan painamalla näppäimistöltä CTRL + ALT + →. Saman toiminto olisi mahdollista toteuttaa tietokonenäön avulla huomattavasti luontevammin ja saumattomammin kallistamalla päätä oikealle.

2.3 Tietokonenäkö

Tekoälyn asiantuntijat uskoivat 40 vuotta sitten tietokoneen näkökyvyn vastaavan haastavuudeltaan opiskelijan kesäprojektia. Ongelma on yhä ratkaisematta, ja tietokonenäkö on kehittynyt omaksi aktiiviseksi tieteenhaarakseen, jolla on yhteyksiä matematiikkaan, tietotekniikkaan, fysiikkaan, havainnoinnin psykologiaan ja neurotieteeseen. Tietokonenäön kehitystä on hidastanut näön biologisen mekanismin tuntemattomuus, joka tekee sen emuloimisen tietokoneella vaikeaksi, ja toisaalta puhtaasti tietokonelogiikkaan pohjautuvan näkökyvyn toteutusten vastoinkäymiset. Onnistumisiakin on saavutettu, erityisesti matemaattisiin algoritmeihin pohjautuvan geometrisen tietokonenäön alueella. (Hartley & Zisserman 2004, 15.)

Tietokonenäöllä tarkoitetaan yksittäisen kuvan tai liikkuvan kuvan sisältämän tiedon muuntamista tietokoneella joko päätöksentekoon tai uuteen esitystapaan (Bradski & Kaehler 2008, 2). Esimerkiksi käyttäjän kasvoja kuvattaessa tietokonenäkö voi muuntaa kuvan tiedot päätökseen siitä, onko kuvassa mukana kasvoja vai ei, ja esitystapaan, jossa kasvojen keskipisteen sijainti ilmaistaan x- ja y-koordinaatteina. Ihmisen on helppo tunnistaa kasvot kuvasta näön biologisen mekanismin avulla, mutta tietokoneelle tehtävä on paljon vaikeampi. Seuraava kuva (Kuvio 1) havainnollistaa tietokoneen ongelmaa kuvien tulkitsemisessa. Ihminen osaa kyllä kertoa, onko kuvan autossa sivupeilejä, mutta tietokone joutuu sen tunnistaakseen turvautumaan monimutkaisiin matemaattisiin algoritmeihin kuvapisteiden sisältämän tiedon tulkitsemiseksi.



Kuvio 1: Tietokoneelle sivupeili on sarja numeroita (Bradski & Kaehler 2008, 3)

Tietokonenäön ongelmia monimutkaistaa vielä se, että tietokoneen tulkitsema kuva on kaksiulotteinen esitys kolmiulotteisesta maailmasta, jolloin jo kuvaa muodostettaessa menetetään merkittävä osa alkuperäisestä informaatiosta. Tehtävää vaikeuttavat lisäksi kuvan häiriöt, vääristymät, heijastukset, liikkeen aiheuttamat virheet, linssin epätäydellisyys, rajoittunut valotusaika ja kuvanpakkauksen aiheuttamat virheet (Bradski & Kaehler 2008, 4). Näiden häiriöiden korjaamiseen ja tietokonenäön muihin perustehtäviin, kuten objektien tunnistamiseen ja niiden liikkeen seurantaan, on kehitetty matemaattisia algoritmeja. Yhteen tällaisia algoritmeja tarjoavaan kirjastoon ja myös itse algoritmeihin tutustutaan tarkemmin luvussa 4. Vaikka tarvittava tekniikka tietokonenäön hyödyntämiseen havaitsevissa käyttöliittymissä onkin jo arkipäivää, edellyttää sen käyttö ohjelmoijalta yhä tietokonenäön algoritmien toteuttamista tai vähintään niiden tuntemista. Se on vaikeuttanut ja hidastanut tekniikan käyttöönottoa.

Tietokonenäön algoritmien kehittyminen on jo mahdollistanut useita geometriseen tietokonenäköön pohjautuvia käytännön sovelluksia. Niitä löydetään esimerkiksi viihde-elektroniikasta pelikonsoleiden ohjauksesta ja ajotietokonelaitteista. Yhtenä tietokonenäön läpimurtona viihde-elektroniikan kuluttajien tietoisuuteen voidaan pitää Sony Playstation 2 -pelikonsolin mikrofonilla varustettua EyeToy-kameraa, jonka avulla pelaaja näkee itsensä televisioruudulla pelin kanssa vuorovaikuttavana hahmona. EyeToy-pelin havaitseva käyttöliittymä, jossa vastustajan tyrmätäkseen pelaajan tulee huitaista häntä kohti kädellään ja tehostaa lyöntiä huutamalla, vastaa hyvin Normanin visiota luontevasta ja saumattomasta tehtävään sulautuvasta käyttöliittymästä.

2.4 Havaitseva käyttöliittymä

Graafisten käyttöliittymien tuen laajentaminen kohti suurempaa valikoimaa skenaarioita, tehtäviä, käyttäjiä ja mieltymyksiä edellyttää siirtymistä käyttörajapintoihin, jotka ovat luonnollisia, intuitiivisia, mukautuvia ja tunkeilemattomia. Tähän suuntaan ihmisen ja tietokoneen vuorovaikutusta laajentavaa aluetta kutsutaan havaitseviksi käyttöliittymiksi (Perceptual User Interfaces, PUI). Havaitsevilla käyttöliittymillä pyritään ennen kaikkea muuttamaan vuorovaikutusta ihmisten keskinäisen vuorovaikutuksen kaltaiseksi. (Turk 1998, 1.)

Havaitsevien käyttöliittymien etuina perinteisiin graafisiin käyttöliittymiin verrattuna ovat siirtyminen komento-vastaus -dialogista luonnollisempaan vuorovaikutukseen, käyttöliittymän oppimiskynnyksen madaltuminen, ajattelun muuttuminen laitekeskeisestä käyttäjäkeskeiseksi ja havainnoinnin läpinäkyvyyden saavuttaminen. Tämän toteutuminen edellyttää tietokonenäön hyödyntämistä, puheen ja äänen tunnistusta ja koneellista oppimista. (Turk 1998, 2.)

Näköpohjaiset käyttöliittymät (Vision Based Interfaces, VBI) on havaitsevien käyttöliittymien alakenttä, joka keskittyy kehittämään visuaalista havainnointia. Se pohjautuu tietokonenäön algoritmien hyödyntämiseen objektien tunnistamiseksi ja paikantamiseksi, niiden liikkeen seuraamiseksi, pään ja kasvojen mallintamiseksi, kasvojen ominaisuuksien seuraamiseksi ja ihmisen liikkeiden ja toimintojen tulkitsemiseksi. (Turk 1998, 3 - 4.)

Havaitsevilla käyttöliittymillä tuskin voidaan täysin korvata perinteisiä tietokonesovellusten käyttöliittymiä, sillä niillä on omat kiistattomat etunsa tietyissä tehtävissä. Esimerkiksi kryptinen Perl-ohjelmointi suullisesti tai viittomalla ei vaikuta lainkaan houkuttelevalta. Tämän opinnäytetyön puitteissa keskitytäänkin WIMP-pohjaisten käyttöliittymien tehostamiseen tietokonenäöllä. Avainsanana on tehostaminen, sillä tarkoituksena ei ole yrittää täysin korvata osoitinlaitteita ja näppäimistöä sovellusten ohjaamisessa, vaan ennemminkin tarjota käyttöliittymäsovelluksille uusi havainnointimekanismi laajentamaan käyttöliittymän toimintamahdollisuuksia ja mahdollistamaan kokonaan uusia käyttöä helpottavia konsepteja.

Tässä luvussa 2 on käyty läpi opinnäytetyön konstruktiivista osuutta perustelevaa teoriaa. Tarkoituksena on ollut osoittaa perinteisten WIMP-pohjaisten käyttöliittymien vuorovaikutuksen rajoituksia ja selvittää havaitsevien käyttöliittymien mukanaan tuomia etuja ihmisen ja tietokoneen vuorovaikutukselle. Seuraavissa luvuissa esitellään MotionKit-kirjaston kehityksessä käytettyjä teknologioita, kuten Qt-kehitysympäristöä ja -käyttöliittymäkirjastoa sekä tietokonenäön algoritmeja tarjoavaa OpenCV-kirjastoa, ja kuvataan lopuksi itse MotionKit-kirjaston sisäistä rakennetta ja toiminnallisuutta.

3 Qt

Käyttöliittymäsovellusten kehityksen lähtökohdat ovat muuttuneet viime vuosina. Vaikka Microsoft Windows onkin edelleen dominoiva käyttöjärjestelmien markkinoilla, ovat sen kilpailijat nostaneet profiiliaan ja saaneet sovelluskehittäjien huomiota. Enää ei ole itsestään selvää aloittaa käyttöliittymäsovelluksen ohjelmointia natiivilla Microsoft Windows -kehitysympäristöllä, sillä tarjolla on myös monelle alustalle yhteensopivia vaihtoehtoja, kuten Qt.

Qt on paitsi tämän hetken käytetyimpiä moderneja käyttöliittymäkirjastoja, myös kokonainen C++-kehitysympäristö. Se mahdollistaa yhden koodipuun käyttämisen sovellusten kääntämiseen Windows-, Mac- ja Linux-käyttöjärjestelmissä ajettavaksi. Tässä luvussa esitellään Qt-kehitysympäristöä, jolla tietokonenäön MotionKit-apukirjasto on kehitetty, ja jota suositellaan käytettäväksi myös sitä hyödyntävien käyttöliittymäsovellusten rakentamisessa. Qt:n esittelyssä keskitytään erityisesti niihin ominaisuuksiin ja käsitteisiin, jotka käyttöliittymäohjelmoijan tulee tuntea.

3.1 Yleistä Qt:stä

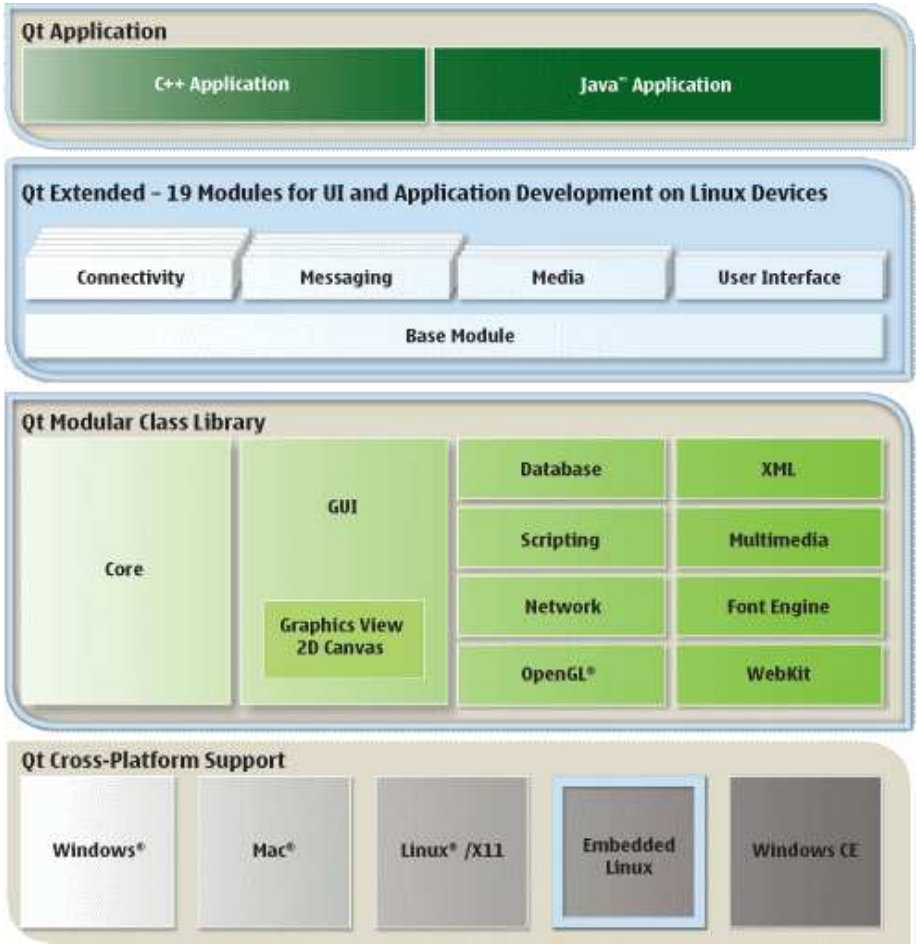
Qt-kehitysympäristö on saatavissa kahdella lisenssillä: ilmaiseksi avoimen lähdekoodin GPL-lisenssillä avoimen lähdekoodin sovelluskehitykseen ja maksullisesti kaupallisella lisenssillä kaupalliseen sovelluskehitykseen. Oleellisin ero näissä kahdessa lisenssimuodossa on, että GPL-lisenssin alaisena kehitetyn ja julkaistun sovelluksen lähdekoodikin on julkaistava, eli Qt-sovelluksen lähdekoodin pitäminen yksityisenä edellyttää Qt:n maksullista lisenssiä (Blanchette & Summerfield 2008, 589 - 590). Lisenssipolitiikka on tärkeää selvittää ja tiedostaa etukäteen suunniteltaessa avoimen lähdekoodin lisenssin alaisten moduulien käyttöä tai edes linkitystä osaksi kaupallisia projekteja.

Qt tarjoaa rikkaan kokoelman widgettejä, eli graafisissa käyttöliittymissä käytettäviä komponentteja, perinteisen GUI-toiminnallisuuden toteuttamiseksi. Qt esittelee uuden objektien välisen kommunikointimekanismin ja tarjoaa erinomaisen tuen multimedialle, 3D-grafiikalle, lokalisoinnille, SQL:lle, XML:lle ja alustakohtaisille laajennuksille. Käyttäjän syötteen käsittelyyn tarjotaan perinteinen tapahtumankäsittelymalli. Qt-sovellusten käyttöliittymiä voidaan rakentaa ohjelmakoodissa tai visuaalisella

Qt Designer -työkalulla, joka voidaan myös liittää osaksi integroitua kehitysympäristöä, kuten Eclipseä. (Qt 4.4 Whitepaper 2008, 1.)

3.2 Qt:n rakenne

Qt:n modulaarinen rakenne ja sen tarjoamien palvelujen luokittelu näkyvät seuraavassa rakennekaaviossa (Kuvio 2).



Kuvio 2: Qt:n rakennekaavio (http://www.qtsoftware.com/products)

Kuten kaaviosta ilmenee, käyttöliittymäkirjasto (GUI) on vain yksi osa Qt:n laaja-alaisesta tarjonnasta sovelluskehittäjille. Käyttöliittymäkehittäjän onkin hyvä tietää myös GUI-moduulin ulkopuolisista palveluista, kuten OpenGL-kirjastosta, joka mahdollistaa 3D-ominaisuuksien käytön, ja Multimedia-kirjastosta, josta löytyy esimerkiksi tuki Phonon-mediatoistimelle. Seuraavissa luvuissa käydään tarkemmin läpi myös näitä palveluita.

3.3 Käyttöliittymä

Qt:n käyttöliittymäkirjasto tarjoaa valmiita widgettejä, eli visuaalisia elementtejä, joita käytetään käyttöliittymien rakentamisen komponentteina. Näihin lukeutuvat käyttäjän syötteen vastaanottoon tarkoitetut widgetit, kuten QLineEdit yhden rivin tekstinsyöttöön, QCheckBox ja QRadioButton valintojen asettamiseen päälle ja pois, QDateTimeEdit ja QSlider määrien syöttämiseen, QProgressBar toiminnan etenemisen indikointiin, sekä QPushButton ja QToolButton painikkeisiin (Qt 4.4 Whitepaper 2008, 5).

Widgettejä, jotka on lähtökohtaisesti suunniteltu sisältämään muita widgettejä ja tarjoamaan niiden käsittelyyn liittyviä toimintoja, kutsutaan säiliöwidgeteiksi. Näihin lukeutuvat esimerkiksi QTabWidget ja QGroupBox. Seuraavassa kuvassa (Kuvio 3) näkyvät tähän mennessä esitellyt widgetit ja säiliöwidgetit.

Kuvio 3: Qt:n widgettejä ja säiliöwidgettejä (Qt 4.4 Whitepaper 2008, 5)

Layout mahdollistaa käyttöliittymän mukautumisen tyylien, orientaation ja esitettävän sisällön muutoksiin. Sen avulla käyttöliittymäkehittäjä voi välttää kiinteiden kokojen ja sijaintien käytön, mikä helpottaa käyttöliittymäsovelluksen lokalisointia ja käyttöliittymän mukautumista ympäristöön. Layout voi sijoittaa widgetit esimerkiksi vasemmalta oikealle, oikealta vasemmalle, ylhäältä alas tai alhaalta ylös, ja muuttaa dynaamisesti widgettien ominaisuuksia, kuten kokoa. (Qt 4.4 Whitepaper 2008, 6.)

QMainWindow-luokka tarjoaa tavallisia sovelluksen pääikkunan palveluita. Siihen sisältyy kokoelma tyypillisiä pääikkunatoimintoja tarjoavia widgettejä, kuten ikkunan yläosan valikkopalkki, työkalupalkit ja tilapalkki. SDI ja MDI (Single/Multi Document Interface) ovat tuettuja QMainWindow-luokassa. (Qt 4.4 Whitepaper 2008, 10.)

Dialogeja käytetään GUI-sovelluksissa käyttäjän kanssa kommunikointiin. Qt sisältää valmiiksi tehtyjä dialogeja, joissa on toteutettu aputoimintoja tavallisimpiin tehtäviin. Dialogit voivat toimia modal-tilassa, jolloin ne estävät toiminnan muissa käyttöliittymän osissa ennen kuin ne suljetaan, modeless-tilassa, jolloin ne toimivat samanaikaisesti muun käyttöliittymän kanssa, ja semi-modal tilassa, jolloin ne mahdollistavat sovelluksen samanaikaisen toiminnan käyttäjän huomaamatta. (Qt 4.4 Whitepaper 2008, 11.)

Qt:n tarjoama signals and slots -kutsumekanismi mahdollistaa tapahtumien välittämisen niistä kiinnostuneille objekteille ilman, että signaalin lähettäjän tarvitsee tietää vastaanottajista mitään. Widgetit lähettävät tapahtumien sattuessa signaalin tapahtumasta, jolloin se välittyy kyseiseen signaaliin kytkettyihin slotteihin samassa tai muissa widgeteissä.

Esimerkiksi sovelluksen ”Lopeta”-painikkeen clicked()-signaali voidaan yhdistää sovelluksen quit()-slottiin, jolloin painiketta painettaessa sovellus sulkeutuu (Qt 4.4 Whitepaper 2008, 7). On huomioitavaa, että vaikka signals and slots -mekanismi onkin Qt:n tavallisin tapa widgettien väliseen kommunikointiin, se ei kuitenkaan sulje pois perinteisten takaisinkutsumekanismien käyttöä. Esimerkiksi MotionKit-kirjastossa käytetään sisäisesti perinteistä takaisinkutsumekanismia abstraktin rajapinnan avulla toteutettuna ja ulkoisesti signals and slots -mekanismia viestinnässä käyttäjäsovellukselle.

Säikeillä on käyttöä sovelluksissa, joissa tehdään ajallisesti pitkiä toimintoja ja joiden käyttöliittymässä halutaan kuitenkin säilyttää häiriöttömät käyttäjävasteajat. Qt tukee säikeiden käyttöä ja tarjoaa valmiit toteutukset säikeestä, mutexista, semaforista, säikeille yhteisestä tietovarastosta ja erilaisista lukintamekanismeja tarjoavista luokista (Qt 4.4 Whitepaper 2008, 14). MotionKit-apukirjasto suorittaa kuvien kaappauksen ja analysoinnin omissa säikeissään hidastamatta käyttäjäsovellusta.

3.4 Grafiikka

Qt tarjoaa tukea 2D- ja 3D-grafiikalle. 2D-grafiikassa voidaan käyttää rasteri- ja vektorikuvia ja ladata sekä tallentaa useita eri kuvaformaatteja. QPainter-luokka vastaa kuvien piirtämisestä widgetteihin. Se tarjoaa perustoiminnot, kuten muotojen piirtämisen, sekä myös kuvan käsittelyn toimintoja, kuten muunnokset ja rajaamiset (Qt 4.4

Whitepaper 2008, 19). MotionKit-apukirjasto käyttää sisäisessä kuvankäsittelyssään OpenCV:n tarjoamia toimintoja.

QPixmap ja QImage-luokkia käytetään yleisesti kuvien käsittelyssä Qt:ssä ja useat widgetit tukevat niitä kuvien piirtämisessä. QPixmap mahdollistaa kuvan nopeamman renderöinnin ja QImage soveltuu paremmin kuvamanipulaatioon ja kuvien käsittelemiseen eri värisyvyyksissä ja formaateissa (Qt 4.4 Whitepaper 2008, 19). MotionKit antaa käyttäjäsovellukselle kuvat QImage-muodossa, joten käyttäjäsovellus voi siis piirtää sen sellaisenaan useassa widgetissä tai jatkokäsitellä sitä.

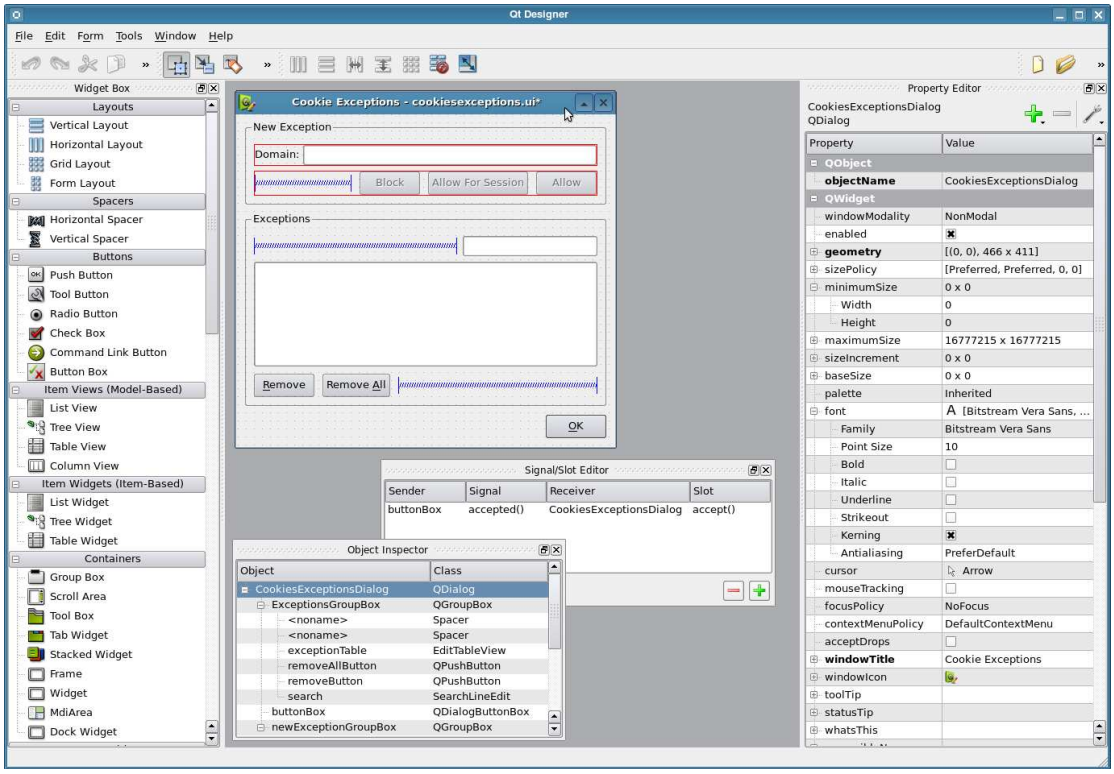
3D-grafiikkaa on mahdollista hyödyntää OpenGL-moduulin avulla. Sen tehdäkseen ohjelmoijan on periytettävä QGLWidget aliluokkaan ja piirrettävä siihen standardeilla OpenGL-funktioilla. Qt tarjoaa funktioita värimuunnoksiin QColorin ja OpenGL:n väriformaatin välillä. (Qt 4.4 Whitepaper 2008, 23.)

3.5 Qt Designer

Qt Designer on graafisen käyttöliittymän suunnittelutyökalu Qt-sovelluksille. Sovellukset voidaan kirjoittaa käyttöliittymää myöten suoraan lähdekoodissa, tai vaihtoehtoisesti käyttöliittymä voidaan suunnitella nopeasti Qt Designerillä, jonka tuottaman uitiedostoon tallentuvan käyttöliittymämäärityksen voi asettaa lähdekoodissa ajonaikaisesti ikkunan tai widgetin käyttöön. Qt Designer on erityisen hyödyllinen käyttöliittymäsuunnittelijalle, joka haluaa kokeilla mahdollisia konsepteja vähällä koodin kirjoittamisella. Tätä kannattaa hyödyntää myös MotionKit-kirjastoa käyttävien konseptien kehittelyssä.

Qt Designerin käyttö on yksinkertaista. Kehittäjä voi vetää widgettejä työkalupalkista lomakkeelle, käyttää tavallisia editointityökaluja ja muuttaa widgettien ominaisuuksia sekä asettaa widgettien käyttöön layoutteja. Qt Designer poistaa tarpeen koodin kääntämiseen, linkkaamiseen ja ajamiseen käyttöliittymän testauksessa, sillä se tarjoaa esikatselutoiminnon, jolla voidaan ennakkoon testata yleisimpiä toimintoja. (Qt 4.4 Whitepaper 2008, 15.)

Alla oleva kuva (Kuvio 4) esittää Qt Designerin toimintaa käyttöliittymän suunnittelussa. Qt Designer tarjoaa useita käyttöliittymäkehitystä helpottavia aputoimintoja, kuten ominaisuuksien hallintaikkunan.



Kuvio 4: Käyttöliittymän suunnittelu Qt Designerillä (Qt 4.4 Whitepaper 2008, 17)

4 OpenCV

Tietokonenäön sovelluskehitys on työlästä, jos sovelluskehittäjän on itse suunniteltava ja toteutettava tarvittavat tietokonenäön algoritmit. Algoritmeista tulee tällöin helposti sovelluskehityshankkeen haastavin osuus, mitä ei välttämättä osata ennakoida sovelluslähtöisessä suunnittelussa ja joka saattaa osoittautua liian korkeaksi kynnykseksi hankkeen toteutuksessa.

Tietokonenäön sovelluskehitystä helpottavat valmiit kirjastot, jotka tarjoavat optimoituja algoritmeja käytettäviksi tietokonenäön keskeisiin toimintoihin, kuten objektien tunnistamiseen ja seuraamiseen videolähteestä. Näiden kirjastojen käytön avulla tietokonenäön sovelluskehitystä voidaan nopeuttaa ja siirtää työn painopistettä algoritmeista itse sovelluksen toimintaan.

Tässä luvussa käsitellään yhtä tällaista tietokonenäön algoritmeja tarjoavaa OpenCV-kirjastoa siltä osin, kuin se olisi suotavaa tuntea MotionKit-kirjaston toteutuksen ja toiminnan ymmärtämiseksi. Tämän luvun algoritmien kuvaukset on tarkoitettu syventäväksi aineistoksi grafiikkaohjelmoinnin tuntevalle henkilölle, joka harkitsee OpenCV:n käyttöä ja on kiinnostunut sen tarjonnasta. Algoritmit esitellään yleisellä tasolla sanallisesti ilman matemaattisia kaavoja.

4.1 Yleistä OpenCV:stä

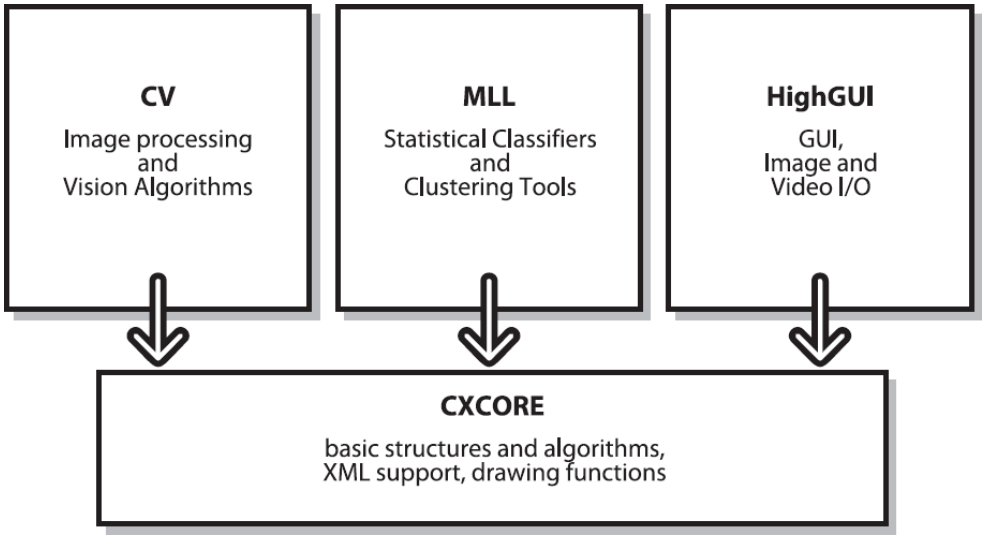
OpenCV on avoimen lähdekoodin tietokonenäön kirjasto, joka on kirjoitettu C:llä ja C++:lla ja joka toimii Linuxissa, Windowsissa sekä Mac OS X:ssä. Rajapinnat Pythonille, Ruby:lle ja Mathlab:ille ovat aktiivisessa kehityksessä. OpenCV tarjoaa kattavan infrastruktuurin tietokonenäön sovellusten kehittämiseen. Se sisältää yli 500 funktiota liittyen tietokonenäön eri osa-alueisiin ja tarjoaa lisäksi koneellisen oppimisen kirjaston, jota voidaan hyödyntää objektien tunnistamiseen tarvittavan informaation keräämisessä.

OpenCV on suunniteltu reaaliaikaiseen käyttöön tavoitteena laskennallinen tehokkuus. Sen suorituskykyiset algoritmit on kirjoitettu optimoidulla C:llä, ja se osaa hyödyntää moniydinprosessoreita ja Intelin Integrated Performance Primitives (IPP) -kirjaston matalan tason laskennallisia apurutiineja nopeuttaakseen algoritmejaan.

OpenCV:n avoimen lähdekoodin lisenssi on rakennettu sellaiseksi, että koko OpenCV:tä tai sen osia voidaan käyttää kaupallisessa projektissa ilman lähdekoodin avaamista. Kaupallisesti hyödynnettävän lisenssipolitiikan ansiosta OpenCV:n käyttäjäkunnasta löytyy nimekkäitä yrityksiä, kuten IBM, Microsoft, Intel, SONY, Siemens ja Google, sekä tutkimuskeskuksia, kuten Stanford, MIT, CMU, Cambridge ja INRIA. (Bradski & Kaehler 2008, 1 - 2.)

4.2 OpenCV:n rakenne

OpenCV on rakenteeltaan modulaarinen ja suunniteltu sellaiseksi, että sovellus voi hyödyntää tarvittaessa sen eri tasojen toimintoja. OpenCV:n pääkomponentit on kuvattu alla olevassa kaaviossa (Kuvio 5).



Kuvio 5: OpenCV:n pääkomponentit (Bradski & Kaehler 2008, 13)

CV sisältää kuvankäsittelyn perustoimintoja ja korkeamman tason tietokonenäön algoritmeja. MotionKit hyödyntää lukuisia CV:n palveluita, kuten objektien tunnistustietojen latauksen xml-tiedostosta, objektien tunnistuksen algoritmeja, kuvien segmentoinnin algoritmeja käsiteltäessä useita kuvasta löytyviä objekteja sekä piirtotoimintoja piirrettäessä indikaatioita IplImage-kuvaan.

MLL on koneellisen oppimisen kirjasto, joka sisältää tilastollisia luokittelijoita ja klusterointityökaluja. Koneellista oppimista voidaan hyödyntää rakennettaessa objektien tunnistukseen tarvittavaa informaatiota lähdemateriaalista. Haar-kaltaiset luokittelijat ovat

yksi tietokonenäössä käytetty menetelmä objektien tunnistamiseen. Haar-kaltaiset luokittelijat kuvaavat objektin osa-alueiden ominaisuuksiin perustuvan tietosisällön, jonka avulla objektin tunnistaminen on laskennallisesti merkittävästi nopeampaa kuin sen yksittäisten kuvapisteiden väriarvojen vertailulla. OpenCV sisältää valmiiksi opetettuja haar-tunnistetietoja xml-tiedostoina joidenkin objektien, kuten kasvojen, tunnistamiseen. Jos MotionKit-kirjaston käyttäjä haluaa käyttää omia objektityyppejään, hänen tulee kouluttaa omat haar-tunnistetietonsa käyttäen MLL-moduulin palveluita tai OpenCV:n mukana tulevaa sovellusta.

HighGUI sisältää I/O-rutiineja ja funktioita videoiden ja kuvien tallentamiseen ja lukemiseen. Se tarjoaa myös rajapinnan esimerkiksi kameran asetusten käsittelyyn ja kuvien tai video ottamiseen. MotionKit käyttää HighGUI-moduulia kuvien lukemiseen videolaitteista ja videotiedostoista. Videotiedostojen tapauksessa tuettujen formaattien tuki rajoittuu MPEG- ja AVI-formaatteihin ja lukunopeus on rajoitettu videon toistonopeuteen.

CXCore sisältää alkeellisia datarakenteita ja sisältöä OpenCV:n tietotyyppien käytön helpottamiseksi. MotionKit käyttää useita CXCore:n tarjoamia tietotyyppejä, joista oleellisimmat ovat kuvan käsittelyyn käytetty IplImage ja kaksiulotteisten objektien sijainnin käsittelyyn käytetty CvBox2D. CvBox2D sisältää sijaintitiedot liukulukuina ja tarjoaa nopeita laskutoimituksia niiden käsittelyyn.

Viides, kaaviosta puuttuva pääkomponentti on *CvAux*, joka sisältää toimimattomia ja kokeellisia algoritmeja, joita ei ole tarkemmin dokumentoitu. Niitä ei suositella käytettäväksi muussa kuin itse OpenCV-kirjaston kehityksessä.

4.3 Kuvan käsittely

OpenCV käyttää kuvan esitykseen *CXCore*-komponentin tarjoamaa IplImage-kuvamuotoa. Videolähdettä tai videotiedostoa käsiteltäessä jokainen yksittäinen kuva poimitaan videosta IplImage:na, jota voidaan sitten analysoida muiden OpenCV:n toimintojen avulla. Sovellukset toteuttavat tavallisesti konversion IplImage:sta sovellusympäristön omaan natiiviin formaattiin vasta piirrettäessä kuvia näytölle.

Jos IplImage:a halutaan aluksi käsitellä ja sitten analysoida OpenCV:llä, ei yleensä ole tarpeellista tehdä välikonversiota sovelluksen omaan formaattiin käsittelyn vuoksi, sillä OpenCV:n oma HighGUI-komponentti tarjoaa tarkoitukseen kattavan valikoiman kuvankäsittelyalgoritmeja. Niiden toteutus on optimoitu suorityskykyä ajatellen, ja ne ovatkin pääsääntöisesti nopeampia kuin vastaavat toiminnot OpenCV:tä käyttävässä kehitysympäristössä.

Kuvankäsittelyalgoritmeilla on oleellinen merkitys tietokonenäön sovelluksissa, sillä oikein käytettynä niillä on mahdollista säästää merkittävästi laskenta-aikaa sekä parantaa toimintavarmuutta ja tehokkuutta. Seuraavissa kappaleissa on kuvattu sanallisesti oleellisimmat tietokonenäön soveltamisessa käytettävät OpenCV:n tarjoamat kuvankäsittelytoiminnot ja nimetty toiminnot toteuttavat algoritmit.

Kuvan pehmennystä voidaan käyttää esimerkiksi kuvan häiriöiden poistamiseen tai digikuvien/videolähteen häiriöartifaktien korjaamiseen ennen kuvan käsittelyä raskaammilla algoritmeilla. HighGUI tarjoaa kuvan pehmennykseen eri käyttötapoihin soveltuvia algoritmeja, kuten *median*-, *gaussian*- ja *bilateral*-pehmennykset. (Bradski & Kaehler 2008, 109 - 110.)

Kuvan morfologista muunnosta käytetään kuvien häiriöiden poistamiseen, osien eristämiseen muusta kuvasta ja väriliukumien löytämiseen. Tähän tarkoitukseen HighGUI tarjoaa *dilation*, *erosion*, *top hat* ja *black hat* -algoritmit. (Bradski & Kaehler 2008, 115 - 123.)

Kuvan täyttöä käytetään kuvan osien eristämiseen ennen kuvan analysointia kehittyneemmillä algoritmeilla. Sen avulla päästään eroon merkityksettömästä osasta kuvaa ja voidaan siten nopeuttaa kuvan analysointia esimerkiksi objektien tunnistamiseksi. Tähän HighGUI tarjoaa *flood fill* -algoritmin (Bradski & Kaehler 2008, 124).

Kuvan koon muutos on tarpeellisimpia kuvankäsittelyn toimintoja erityisesti monivaiheisessa kuvankäsittelyssä, sillä osa vaiheista voidaan yleensä toteuttaa myös pienemmällä versiolla alkuperäisestä kuvasta, mikä nopeuttaa prosessointia enemmän kuin mitä skaalaus hidastaa. Kuvan skaalaukseen HighGUI tarjoaa *nearest neighbor*, *bilinear*,

pixel-area resampling ja *bicubic interpolation* -algoritmit (Bradski & Kaehler 2008, 130).

Kuvapyramidiksi kutsutaan tietokonenäössä hyödynnettävää algoritmia, jossa kuvaa pienennetään neljäsosaan jokaisella prosessointikerralla. HighGUI sisältää *gaussian* ja *laplacian* -variaatiot kuvapyramidille (Bradski & Kaehler 2008, 132).

HighGUI tarjoaa myös *threshold* -algoritmeja, joita voidaan hyödyntää suodattamaan kuvainformaatiosta osa pois tiettyjen raja-arvojen perusteella, jotta merkityksettömän kuvainformaation määrää saadaan vähennettyä prosessoinnin nopeuttamiseksi (Bradski & Kaehler 2008, 136).

4.4 Kuvan muunnos

Kuvan muunnoksia käytetään tavallisesti kuvan prosessoinnin päätösvaiheessa ennen kuvan piirtämistä. Muunnokset eroavat tavallisista kuvankäsittelyalgoritmeista siinä, että niillä ei pyritä tehostamaan kuvan analysoimista kehittyneemmillä algoritmeilla, vaan niillä tehdään kuvan ulkomuotoon pysyvämpi muutos tietyn lopullisen esitystavan saavuttamiseksi, esimerkiksi muutettaessa kaksiulotteisen kuvan perspektiiviä haluttaessa esittää se osana kolmiulotteista näkymää. Kuvan muunnoksia ei yleensä käytetä tietokonenäön sovelluksissa, joten OpenCV:n muunnosalgoritmit käsitellään tässä luvussa vain pintapuolisesti. Toisinaan niille on kuitenkin käyttöä, kuten esitettäessä tietokonenäön algoritmeilla tutkittu kuva osana käyttöliittymää. Näistäkin algoritmeista annetaan lyhyt sanallinen yleiskuvaus ja mainitaan algoritmit nimeltä.

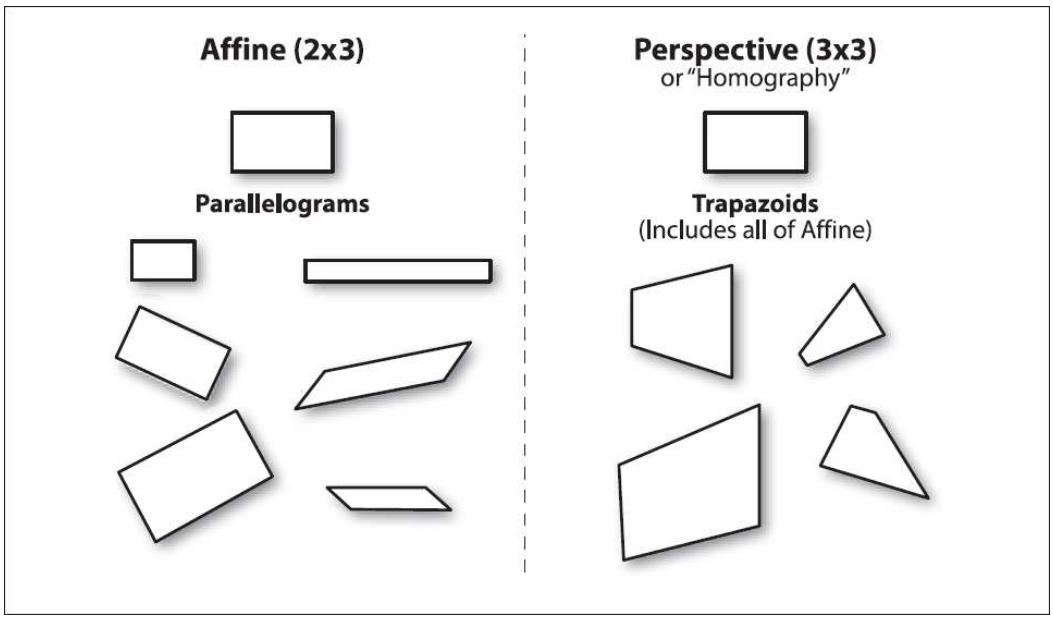
Convolution mahdollistaa kuvan jatkamisen sen reunoilta, jolloin algoritmi muodostaa kuvaan uutta keinotekoista sisältöä (Bradski & Kaehler 2008, 147). Näin muodostettu esitys ei näytä täysin luonnolliselta, mutta kuitenkin selvästi paremmalta kuin käytettäessä yksittäistä taustaväriä. Kuvan jatkaminen on hyödyllistä esimerkiksi haluttaessa piirtää kuva alkuperäistä kokoaan suuremmalle alueelle ilman skaalausta.

Sobel, *scharr*, *laplace* ja *canny* ovat algoritmeja, joiden avulla kuvasta voidaan laskea ja erotella kuvassa esiintyvien hahmojen ääriviivat (Bradski & Kaehler 2008, 148 - 152).

Ääriviivojen erottelu voi olla hyödyllistä vertailtaessa kuvia toisiinsa niissä esiintyvien hahmojen muotojen perusteella tai korostettaessa muotoja visuaalisesti.

Hough-algoritmi etsii ja piirtää kuvan perusmuodot, kuten viivat ja ympyrät (Bradski & Kaehler 2008, 154 - 159). Viivojen tunnistamista voidaan käyttää selvitettäessä kuvassa esitettyä perspektiiviä, sillä se on laskettavissa esimerkiksi rakennuksen linjoja myötäilevistä viivoista. Käytännössä algoritmin käyttö edellyttää kuitenkin kuvamateriaalin tuntemista ennalta, mikä rajoittaa sen hyödyntämistä.

Stretch, *shrink*, *warp* ja *rotate* mahdollistivat kuvan geometriset manipulaatiot. Geometriset manipulaatiot ovat pääsääntöisesti nopeammaksi optimoituja OpenCV:ssä kuin sitä käyttävissä kehitysympäristöissä, joten niitä kannattaakin käyttää jo OpenCV:ssä ennen kuvan muunnosta piirrettäväksi. Geometriset manipulaatiot voidaan toteuttaa kahdella tavalla riippuen siitä, halutaanko muuttaa myös perspektiiviä, kuten alla oleva Kuvio 6 esittää (Bradski & Kaehler 2008, 164).



Kuvio 6: *Affine* ja *Perspective* (Bradski & Kaehler 2008, 165)

Log-polar laskee kuvasta projektion, jossa jokin kuvan piste valitaan uudeksi keskipisteeksi ja kuvan muu sisältö skaalataan sen mukaan (Bradski & Kaehler 2008, 174). Algoritmi voi olla hyödyllinen esimerkiksi muodostettaessa kuvasta projektio, jota halutaan myöhemmin hyödyntää tekstuurina ennalta tiedetyn muotoisen 3D-objektin päälle piirrettynä.

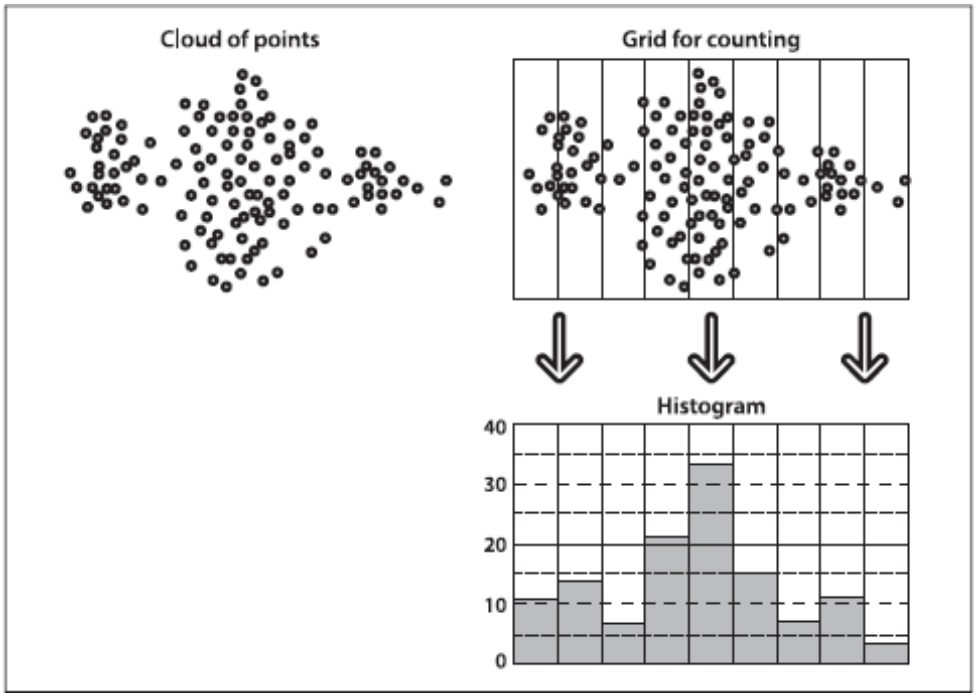
DFT, *Spectrum Multiplication*, *DCT* ja *Integral Images* ovat kuvan muunnoksen matemaattisia algoritmeja, joiden avulla on mahdollista selvittää kuvien ominaisuuksia, laskea haar-ominaisuuksia ja tehdä kuvalle pehmennyksiä ja väriliukumia (Bradski & Kaehler 2008, 177 - 184).

Distance Transform mahdollistaa kuvatun kohteen ja kameran välisen etäisyyden keinotekoisen muunnoksen. Sitä käytetään usein kuvan ääriviivojen tunnistamisen jälkeen tuomaan hahmo lähemmäksi tai kauemmaksi (Bradski & Kaehler 2008, 185).

Histogram Equalization mahdollistaa kuvan väriarvojen dynaamisuuden laajentamisen yli- tai alivalottumisen korjaamiseksi (Bradski & Kaehler 2008, 186).

4.5 Histogrammi

Histogrammeilla on keskeinen merkitys tietokonenäön toteutuksissa. Histogrammit tarkoittavat käytännössä kuvassa esiintyvien merkityksellisten pisteiden, värien tai kulmien lukumäärän laskemista sen eri osista vertailtavaksi myöhemmin otettavien kuvien histogrammeihin. Histogrammeja hyödynnetään erityisesti objektin liikkeen seurantaan liittyvissä algoritmeissa. Alla oleva kaavio (Kuvio 7) esittää histogrammin muodostamista joukosta kuvan pisteitä.



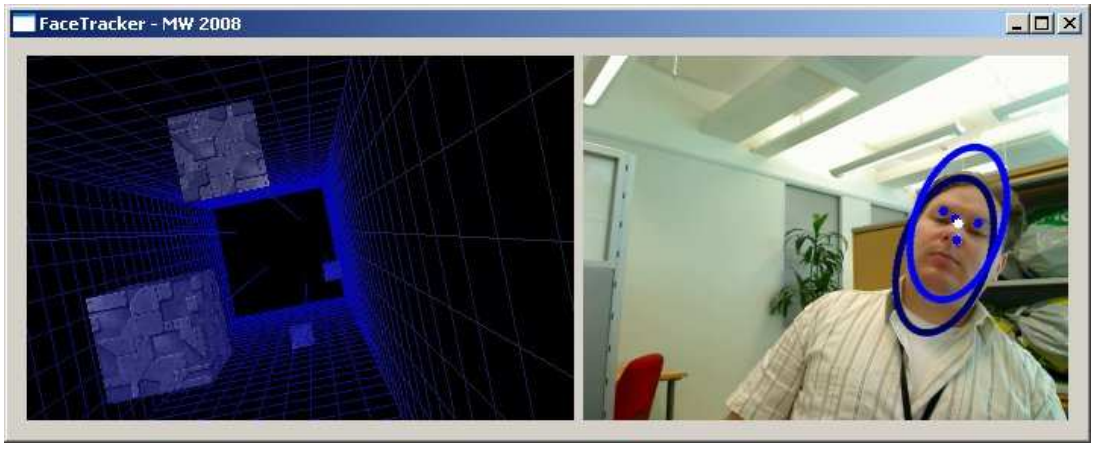
Kuvio 7: Histogrammin muodostaminen (Bradski & Kaehler 2008, 195)

OpenCV tarjoaa histogrammien käsittelyyn oman tietotyypin, jonka avulla niitä voidaan esittää yhdessä tai useammassa ulottuvuudessa, sekä operaatioita histogrammien käsittelyyn, kuten histogrammien vertailun keskenään (Bradski & Kaehler 2008, 194).

4.6 Objektin tunnistus ja seuranta

Objektin tunnistus ja seuranta videolähteestä toteutetaan tietokonenäön sovelluksissa yleensä siten, että aluksi kuvalähteestä etsitään tiettyä objektia kuva kuvalta. Etsintä voidaan toteuttaa esimerkiksi kuvan osien haar-kaltaisia ominaisuuksia tutkivalla algoritmilla. Kun objekti löydetään kuvasta, voidaan objektia alkaa seurata myöhemmistä kuvista histogrammiin pohjautuvalla algoritmilla, joka tunnistaa myös objektin kallistuman. Jos objektin seurannan tarkkuutta halutaan parantaa, voidaan objektin alueelta tunnistaa lisäksi haluttu määrä kiinnostavia pisteitä, eli vahvoja kulmapisteitä, joiden sijaintia seurataan vielä erikseen toisella algoritmilla tarkentamaan objektin sijaintia. Kulmapisteiden etäisyyksien muutoksista toisiinsa ja niiden keskipisteeseen on myös mahdollista laskea objektin kallistumia x-, y- ja z-akselilla. Näin objektien tunnistus on toteutettu esimerkiksi MotionKit-kirjastossa.

Alla oleva ruudunkaappaus (Kuvio 8) on opinnäytetyön pilottiprojektista, jossa tunnistetun objektin (kasvojen) sijaintia jäljitetään tarkennettuna kulmapisteiden sijainnin (sinisten pisteiden) avulla ja projisoidaan näin tarkentunut kasvojen sijainti (valkoinen piste) ohjaamaan vasemmalla olevan 3D-näkymän kameraa reaaliaikaisesti.



Kuvio 8: FaceTracker-pilotti objektin tunnistuksesta ja seurannasta (http://www.wkoti.com/qt/ft.html)

Objektin tunnistukseen OpenCV tarjoaa haar-kaltaisten luokittelijoiden laskemiseen perustuvan algoritmin, jota voidaan käyttää cvHaarDetectObjects()-funktiolla (Bradski & Kaehler 2008, 513). Funktio haluaa parametrikseen ennakkoon koneellisella oppimisella muodostetut haar-tunnistetiedot, jotka vastaavat etsittävää objektityyppiä. Algoritmin avulla kuvasta on mahdollista tunnistaa yksi tai useampi yksilöllinen objekti, kuten kasvot, kämmen tai silmä.

Kun objekti on kerran tunnistettu, sitä voidaan alkaa jäljittää tunnistetun objektin histogrammia hyödyntävällä cvCamShift()-funktion tarjoamalla algoritmilla (Bradski & Kaehler 2008, 341). Algoritmi palauttaa tiedon paitsi objektin sijainnista myöhemmissä kuvissa, myös tiedon objektin x-akselin rotaation muutoksesta. Jos jäljitykseen halutaan erityistä tarkkuutta, voidaan objektin alueelta laskea vahvat kulmapisteet cvGoodFeaturesToTrack()-funktiolla (Bradski & Kaehler 2008, 318). Näiden pisteiden sijaintia voidaan seurata cvCalcOpticalFlowPyrLK()-funktiolla (Bradski & Kaehler 2008, 329). MotionKit käyttää vahvoja kulmapisteitä objektin sijainnin tarkentamiseen.

4.7 Koneellinen oppiminen

Koneellisen oppimisen tavoitteena on datan muuntaminen informaatioksi. Tällä tarkoitetaan esimerkiksi kuvan sisältämän datan muuntamista tietokoneelle merkitykselliseksi tiedoksi. Datan opettelun jälkeen tietokoneen tulisi osata vastata kuvaan liittyviin kysymyksiin, esimerkiksi mikä muu data vastaa parhaiten tätä dataa tai onko kuvassa autoa. (Bradski & Kaehler 2008, 459.)

OpenCV:n yhteydessä koneellista oppimista käytetään objektien tunnistamisessa käytettävien haar-tunnistetietojen muodostamiseksi. Koneellisen oppimisen avulla voidaan prosessoida esimerkiksi tuhansia kuvia kasvoista, minkä jälkeen koneellisen oppimisen tuloksena syntyneiden haar-tunnistetietojen avulla voidaan ohjelmallisesti tunnistaa kasvot uusistakin kuvista.

OpenCV:n mukana tulee valmiiksi koulutetut haar-tunnistetiedot esimerkiksi kasvojen ja silmien tunnistamiseksi, ja tarvittaessa lisää voi kouluttaa itse OpenCV:n mukana tulevalla sovelluksella tai käyttäen OpenCV:n ML-moduulin tarjoamia palveluita.

5 MotionKit

Tässä luvussa kuvataan opinnäytetyön tärkeintä konstruktiivista osuutta: tietokonenäön palveluita tarjoavaa MotionKit-kirjastoa. Kuvaus aloitetaan käymällä läpi kirjaston yleisiä ominaisuuksia, ja jatketaan tutkimalla kirjaston sisäistä rakennetta luokkakaavion avulla sekä kirjaston toimintaa toimintakaavion avulla ja päädytään toimintakokonaisuuksien sanallisiin kuvauksiin. Kirjaston käyttörajapintaan liittyvät toimintakokonaisuudet, kuten kirjaston konfigurointi muodostimessa ja kirjaston ajonaikainen tiedonvälitys sovellukselle, käydään läpi yksityiskohtaisemmin esitellen myös niissä käytettävät tietorakenteet.

5.1 Yleistä MotionKit:istä

MotionKit-kirjasto on tarkoitettu käyttöliittymäohjelmoijalle, joka haluaa hyödyntää tietokonenäön tarjoamia mahdollisuuksia ilman syvällisempää uppoutumista sen edellyttämiin algoritmeihin.

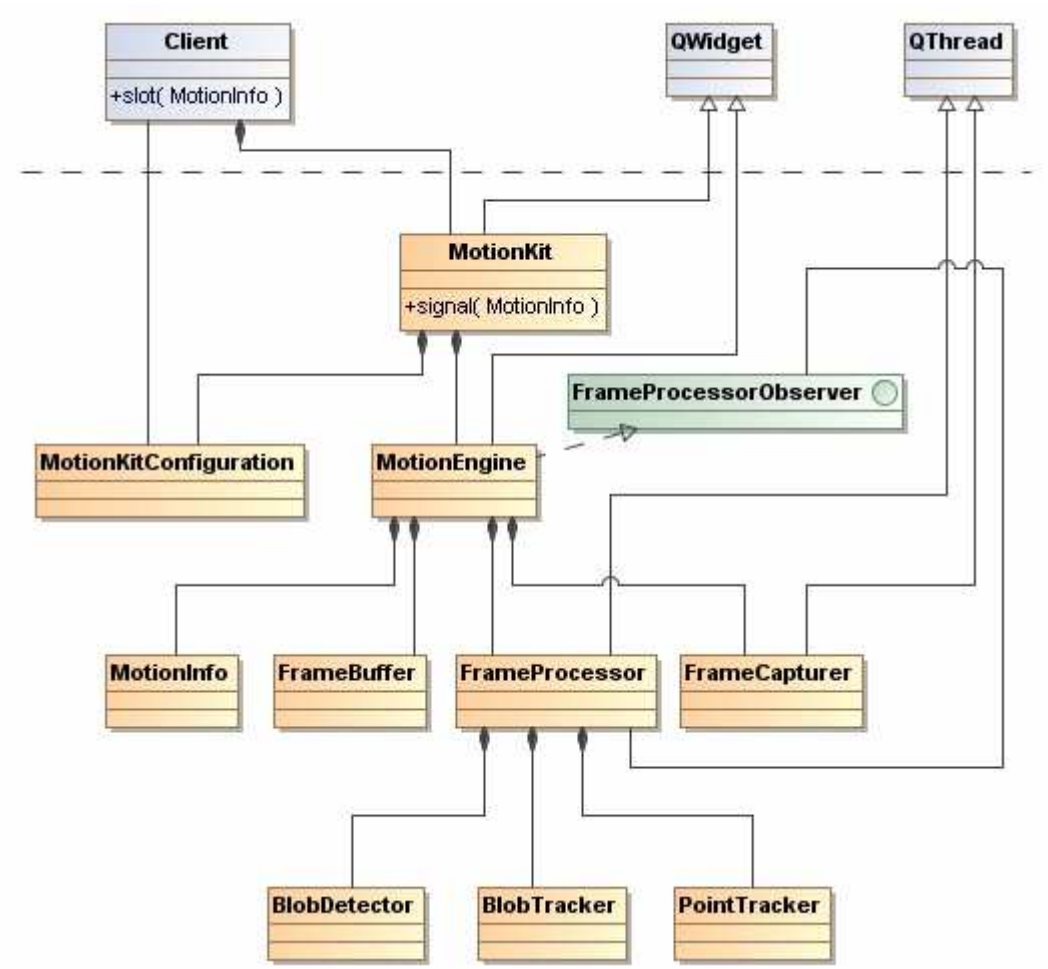
Kirjaston suunnittelun lähtökohtana oli säilyttää yksinkertaisuus käyttäjäsovelluksen näkökulmasta: mitään tietokonenäön tai kuvankäsittelyn algoritmeja ei tarvitse implementoida tai käyttää sovellustasolla. Toisena lähtökohtana oli OpenCV-sidonnaisen koodin kapseloiminen MotionKit-kirjaston sisälle, jotta käyttäjäsovelluksen lähdekoodissa ei tarvitsisi myöskään käsitellä muita kuin Qt:n omia luokkia ja muuttujatyyppejä.

MotionKit-kirjaston avulla havaitsevan käyttöliittymän ohjelmoija voi käyttää videokuvan lähteenä tietokoneeseen kytkettyä laitetta tai paikallista videotiedostoa, tunnistaa siinä esiintyviä ennalta määriteltyjä tai käyttäjän itse määrittelemiä objekteja ja seurata tunnistetun objektin liikkeitä ja rotaation muutoksia reaaliaikaisesti.

MotionKit-kirjasto käyttää OpenCV:n tarjoamia tietokonenäön algoritmeja keskeisiin tietokonenäön tehtäviin. Kirjasto toteuttaa myös itse algoritmien tuottamien tulosten jatkokäsittelyyn ja tarkentamiseen tarvittavaa logiikkaa, kuten vahvojen kulmapisteiden keskiarvojen laskua häiriöiden vähentämiseksi. MotionKit-kirjaston suorituskyky riittää ongelmitta reaaliaikaisen videolähteen käyttöön.

5.2 MotionKit:in rakenne

Seuraavassa luokkakaaviossa (Kuvio 9) kuvataan MotionKit-kirjaston sisältämät keskeisimmät luokat ja niiden väliset suhteet. Yksittäisten luokkien attribuutteja ja operaatioita ei tässä yhteydessä esitellä, sillä tarkoituksena on tutkia kirjaston rakennetta yleisemmällä tasolla sen toiminnan hahmottamiseksi.



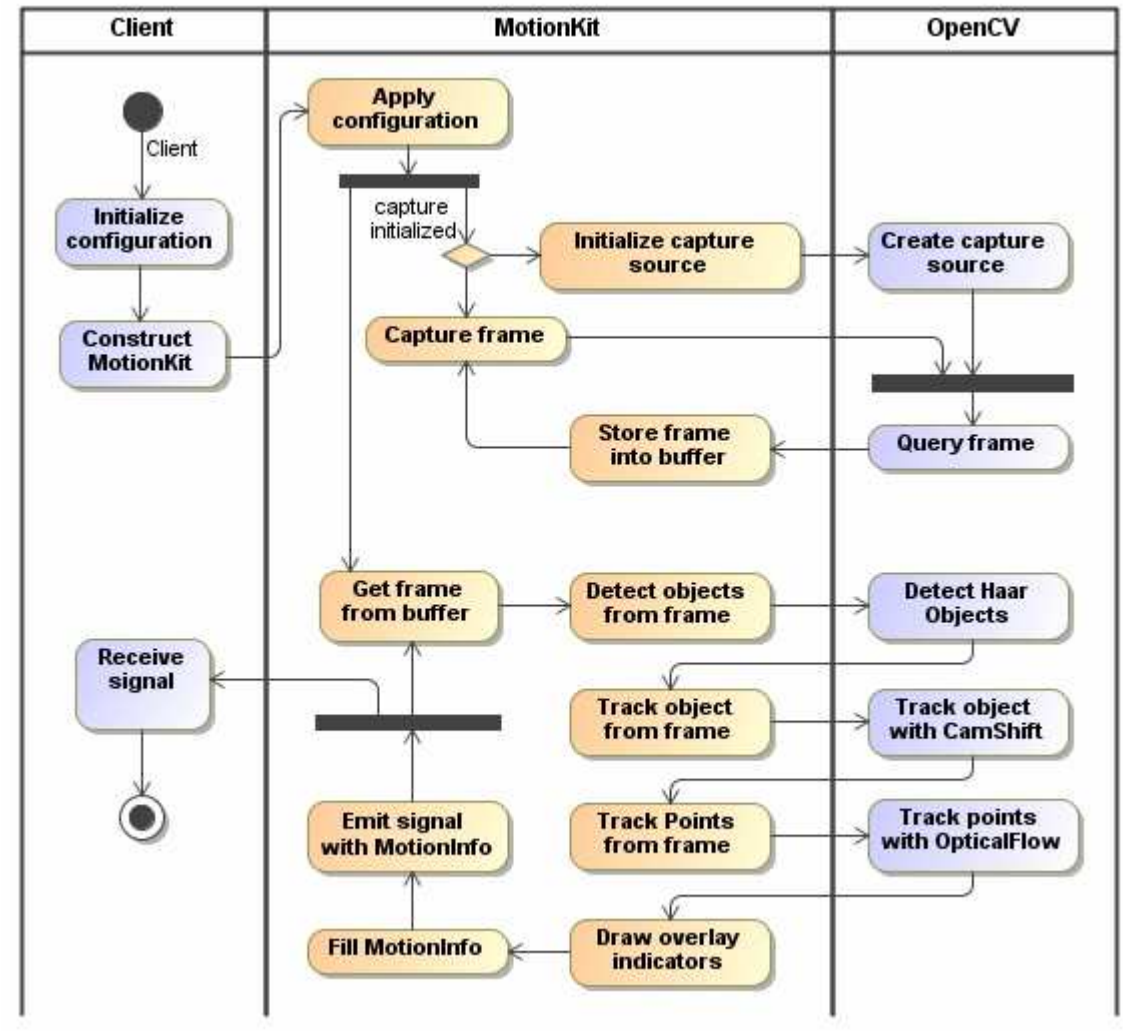
Kuvio 9: MotionKit-kirjaston yleinen luokkakaavio

Yksisäikeiset GUI-sovellukset suorittavat vain yhden toiminnon kerrallaan, ja aikaa vievät toiminnot hidastavat siten käyttöliittymää suorituksen aikana. Ongelma voidaan kiertää Qt:ssä joko antamalla sovelluksen purkaa tapahtumia jonosta kesken pitkäaikaisen toiminnon kutsumalla säännöllisesti *QApplication::processEvents()*-funktiota, tai käyttämällä säikeitä, jolloin hyödytään myös moniydinprosessorien kyvystä suorittaa useita säikeitä samanaikaisesti (Blanchette & Summerfield 2008, 339).

Yksittäisten luokkien tehtäviä kuvataan sanallisesti seuraavan luvun kirjaston toiminnan kuvauksessa, mutta yleisesti voidaan todeta kirjaston ajon tapahtuvan kolmessa eri säikeessä. *MotionKit*- ja *MotionEngine*-widgetit toimivat käyttäjäsovelluksen omassa säikeessä, mutta suoritusintensiivisimmät toiminnot, eli kuvien kaappaus (*FrameCapturer*) ja kuvien analysointi tietokonenäön algoritmeilla (*FrameProcessor*) suoritetaan kumpikin omissa säikeissään. Säikeiden välinen synkronointi tapahtuu *FrameBuffer*-luokassa, joka samalla toteuttaa kuvien puskuroinnin.

5.3 Toiminta

Alla olevassa toimintakaaviossa (Kuvio 10) esitetään MotionKit-kirjaston toimintaa yhteistyössä käyttäjäsovelluksen ja OpenCV-kirjaston kanssa.



Kuvio 10: MotionKit-kirjaston toimintakaavio

Käyttäjä antaa MotionKit-kirjastolle muodostimessa tarvittavat konfiguraatiotiedot, joiden perusteella tietokonenäkö toimii. MotionKit-kirjasto huolehtii kuvien kaappauksesta käyttäen videolähteenä järjestelmään kytkettyä laitetta, kuten web-kameraa, tai vaihtoehtoisesti AVI- tai MPEG-muotoista videotiedostoa. MotionKit lukee kuvalähdettä erillisessä säikeessä ja tallentaa kuvat kuvapuskuriin. Toinen säie ottaa kuvia kuvapuskurista ja analysoi niitä käyttäjän antamien määritysten perusteella, sekä asettaa saadut tulokset tietorakenteeseen, joka sitten välitetään käyttäjäsovellukselle Qt:n signals and slots -mekanismin avulla. Mekanismi mahdollistaa käyttäjäsovelluksen kutsumisen ilman, että sen tarvitsisi esimerkiksi toteuttaa kirjaston tarjoamaa abstraktia rajapintaa takaisinkutsulle. Kuvien kaappauksen ja analysoinnin säikeistämisellä pyritään minimoimaan käyttäjäsovelluksen ja erityisesti käyttöliittymän hidastuminen intensiivisen prosessoinnin aikana.

MotionKit itsessään nojautuu tietokonenäön toteutuksessa pääosin OpenCV:n tarjoamiin tietokonenäön algoritmeihin, jotka eivät näy käyttäjäsovellukselle asti. Algoritmien suoran käytön lisäksi MotionKit toteuttaa algoritmien soveltamisen edellyttämää logiikkaa. MotionKit-kirjaston käytön yksinkertaisuus on tietenkin mahdollista vain sen tarjoaman toiminnallisuuden kustannuksella. Monimutkaiset havaitsevan käyttöliittymän konseptit on edelleen järkevämpää toteuttaa suoraan OpenCV:n tai kokonaan omien algoritmien avulla.

5.4 Alustus

Kirjasto alustetaan antamalla *MotionKit*-widgetin muodostimen mukana *MotionKitConfiguration*, jonka asetusten mukaan tietokonenäön logiikan halutaan toimivan. Halutut asetukset määräytyvät tietokonenäköä käyttävän sovelluksen konseptin mukaan.

Alustuksen jälkeen kirjasto alkaa välittömästi lukea kuvia videolähteestä ja analysoida kuvista konfiguraation mukaisia tietoja. Sovellus ei kuitenkaan saa kirjaston palauttamia tietoja käyttöönsä, ennen kuin se rekisteröityy vastaanottamaan kirjaston lähettämän signaalin. Jos sovelluksen täytyy kesken ajon muuttaa käytettävää konfiguraatiota, MotionKit-widget täytyy poistaa ja muodostaa uudelleen.

Seuraavassa kuvassa (Kuvio 11) esitetään konfigurointiin käytettävä tietorakenne ja eri asetuksille tuetut vaihtoehdot.

```
1     enum MCaptureSource { ECaptureSourceCamera = 0,
2                           ECaptureSourceFile };
3     enum MObjectType    { EObjectTypeFace = 0,
4                           EObjectTypeEyes,
5                           EObjectTypeHand,
6                           EObjectTypeCustom };
7     enum MObjectAction  { EObjectActionDetect = 0,
8                           EObjectActionTrack,
9                           EObjectActionDetectAndTrack };
10    enum MImageType     { EImageTypeNone = 0,
11                          EImageTypeSource,
12                          EImageTypeSourceWithOverlays };
13    struct MotionKitConfiguration {
14        MCaptureSource captureSource;
15        QString captureSourceIdentifier;
16        MObjectType objectType;
17        QString objectTypeIdentifier;
18        MObjectAction objectAction;
19        MImageType imageType;
20    };
```

Kuvio 11: MotionKit-kirjaston konfiguroinnin tietorakenne

Konfiguraatiossa määritellään aluksi haluttu videolähde, josta kuvia luetaan. Asetukselle annetaan arvoksi *ECaptureSourceCamera* haluttaessa käyttää videolähteenä tietokoneeseen kytkettyä kameraa, TV-korttia tai muuta videolähdelaitetta, tai *ECaptureSourceFile*, jos kuvia halutaan lukea paikallisesta videotiedostosta. Jälkimmäisessä tapauksessa konfigurointiin on asetettava myös *captureSourceIdentifier*-merkkijonomuuttuja, jossa määritellään käytettävän videotiedoston polku.

Seuraavaksi määritellään kuvista etsittävän objektin tyyppi. Valittavissa on valmiiksi määriteltyjä objektityyppejä, kuten *EObjectTypeFace, EObjectTypeEyes ja EObjectTypeHand*. Jos kuvista halutaan etsiä jotain ennalta määrittelemätöntä objektia, voidaan käyttää asetusta *EObjectTypeCustom* ja määritellä sen lisäksi *objectTypeIdentifier*, joka on merkkijonomuuttuja ja joka sisältää polun paikalliseen haar-tunnistetiedot sisältävään tiedostoon.

Konfiguraatiossa on myös kerrottava, mitä määritellyille objekteille tarkalleen ottaen halutaan tehdä. *EObjectActionDetect*-asetuksella kuvista vain tunnistetaan määritetyt objektit, ja niiden sijaintitiedot tallennetaan palautettavaan tietorakenteeseen. Tunnistettuja objekteja voi löytyä useampia yhdestä kuvasta. Pelkkä tunnistus ei sovellu liikkeen seurantaan, sillä objektien sijainti saattaa vaihdella liikaa tai tunnistus saattaa epäonnis-

tua useita kertoja sekunnissa. Liikkeen seuranta onnistuu paremmin *EObjectActionTrack*-asetuksella, jolloin MotionKit alkaa jäljittää ensimmäisen tunnistetun objektin sijaintia käyttäen OpenCV:n kehittyneempiä algoritmeja ja MotionKit:issä toteutettua tarkennuslogiikkaa. *EObjectActionDetectAndTrack*-asetus jatkaa objektien tunnistamista myös jäljityksen ollessa käynnissä.

Kirjaston palauttamaan tietorakenteeseen voidaan haluttaessa pyytää mukaan QImage-muotoinen kuva, josta objekteja on yritetty tunnistaa tai josta objektin liikettä on yritetty seurata. *EImageTypeNone*-asetuksella kuvaa ei liitetä tietorakenteeseen lainkaan. *EImageTypeSource*-asetuksella tietorakenteen kuva on alkuperäinen videolähteestä saatu kuva konvertoituna Qt:n natiiviin QImage-muotoon. QImage on valittu käyttöön siksi, että sitä piirrettäessä käytetään Qt:n omaa sisäistä piirtäjää, joka tarjoaa identtisen tuloksen kaikissa käyttöjärjestelmissä (Blanchette & Summerfield 2008, 193). *EImageTypeSourceWithOverlays*-asetuksella kuvaan on piirretty lisäksi indikaatiot tunnistettujen objektien ympärille ja liikkeen seuraamisen tuloksista. Tätä asetusta voidaan hyödyntää esimerkiksi videolähteen kalibroinnissa.

Esimerkkejä kirjaston alustuksesta löytyy luvun 6 käyttöliittymäprototyypeistä, joiden yhteydessä esitellään käytetyt konfiguraatiot kullekin konseptille.

5.5 Kuvan lukeminen videolähteestä

FrameCapturer-luokka vastaa kuvien lukemisesta käyttäjän antaman konfiguraation määrittelemästä videolähteestä. Luokka toimii omassa säikeessään ja täyttää *FrameBuffer*-kuvapuskuria videolähteen tarjoamalla nopeudella. Mikäli videolähteeksi on valittu videotiedosto, yksittäiset kuvat kaapataan videon toistonopeudella, vaikka kirjaston suorituskyvyn puitteissa kuvat olisikin mahdollista sekä kaapata että prosessoida moninkertaisella nopeudella. Tämä ei kuitenkaan palvelisi MotionKit-kirjaston reaaliaikaisen toiminnan periaatetta.

FrameCapturer-luokka käyttää videolähteen lukemiseen OpenCV-kirjaston HighGUI-moduulin palveluita, jotka mahdollistavat videolähteiden tai videotiedostojen lukemisen kuva kerrallaan. Luettu kuva palautetaan OpenCV:n natiivissa *IplImage*-muodossa, mi-

kä mahdollistaa MotionKit:in sisäisen kuvan jatkokäsittelyn ja analysoinnin OpenCV:n algoritmeilla ilman kuvaformaatin edestakaista konvertointia.

5.6 Kuvan analysointi

Kuvien analysointi on toteutettu useassa luokassa. *FrameProcessor*-luokka ottaa säikeessään kuvia kuvapuskurista, ja käyttää sitten käyttäjäsovelluksen antaman konfiguraatiotiedon mukaisesti *BlobDetector*, *BlobTracker* ja *PointTracker*-luokkia objektien tunnistamiseen tai yhden objektin liikkeen seurantaan sekä sijainnin tarkentamiseen.

Objektien tunnistus aloitetaan lataamalla konfiguraatioasetusten mukainen haar-tunnistetieto, joka annetaan OpenCV:n HaarDetect-algortimille. Jos MotionKit on määritelty vain seuraamaan objektia (*EObjectActionTrack)*, keskeytetään objektien tunnistus heti ensimmäisen löydyttyä. Muussa tapauksessa algoritmin annetaan etsiä kaikki objektit kuvasta.

Objektin liikkeen seuranta edellyttää objektin tunnistamista, jonka jälkeen alustetaan *BlobTracker*-luokka etsimään tunnistettu objekti OpenCV:n CamShift-algoritmin avulla seuraavien kuvien histogrammien muutoksista. Tämä ei yksistään riitä luotettavaan liikkeen seurantaan, sillä toisinaan algoritmi epäonnistuu seuraamaan liikettä tai objektin paikka saattaa vaihdella liikaa videokuvan häiriöiden vuoksi. Häiriöiden minimoimiseksi käytetään *BlobTrackerin* apuna *PointTracker*-luokkaa, joka etsii *BlobTracker*in seuraaman objektin alueelta vahvat kulmapisteet, joiden väriarvot, kirkkaus tai muut ominaisuudet erityisesti poikkeavat muista objektin alueen kuvapisteistä. Vahvojen kulmapisteiden sijaintia seurataan tulevissa kuvissa OpenCV:n OpticalFlow-algoritmilla. Käyttämällä seurattujen vahvojen kulmapisteiden keskiarvoa aiemmin seuratun objektin keskipisteenä saavutetaan riittävä vakaus liikkeen seurannan toteuttamiseksi.

Kun kuva on analysoitu, *FrameProcessor*-luokka välittää valmiin *MotionInfo*-tietorakenteen abstraktin *FrameProcessorObserver*-rajapinnan kautta *MotionEngine*-luokalle, joka puolestaan lähettää sen signaalin parametrina käyttäjäsovelluksen käsiteltäväksi.

5.7 Tuloksen palautus käyttäjäsovellukselle

MotionKit lähettää signaalin jokaisen prosessoidun kuvan jälkeen, eli tavallisesti 30 kertaa sekunnissa. Signaali sisältää *MotonInfo*-tietorakenteen, joka tarjoaa Kuvion 12 mukaiset julkiset funktiot analysoitujen arvojen lukemiseksi.

```
1    QImage* image();
2    QList<QRect> detectedObjects();
3    QRectF trackedObject();
4    QList<QPointF> trackedObjectPoints();
5    double trackedObjectAngle();
```

Kuvio 12: MotionKit-kirjaston sovellukselle palauttama tietorakenne

Funktio image() palauttaa videolähteen antaman kuvan QImage-muodossa. Jos EImageTypeSourceWithOverlays-asetus on käytössä, kuvaan on piirretty myös indikaatiot tunnistetuista objekteista ja liikkeen seurannan tuloksista. Kuvan kokoa voidaan hyödyntää skaalattaessa tunnistettujen objektien tai liikkeen seurannan sijainteja toiseen kohderesoluutioon. Tietorakenteesta saatavan QImage-objektin varsinainen kuvasisältö pysyy MotionKit-kirjaston omistuksessa, ja se lakkaa olemasta välittömästi seuraavan signaalin kutsumisen jälkeen. Jos sovellus haluaa säilyttää myös aiemmista signaaleista saadut kuvat, tulee sen kopioida niiden kuvasisällöt omistukseensa viimeistään seuraavan signaalin tullessa.

Funktio detectedObjects() palauttaa listan tunnistettujen objektien sijainneista kuvassa. Listan koko kertoo tunnistettujen objektien lukumäärän. Tunnistetuista objekteista on saatavissa sijaintitiedot kokonaislukuina (QRect), mutta ei esimerkiksi kulmaa, toisin kuin seurattavasta objektista.

Funktio trackedObject() palauttaa seurattavan objektin sijaintitiedot liukulukumuodossa (QRectF). Tämä mahdollistaa sijainnin tarkemman seurannan, sillä liikettä seurattaessa pienillä resoluutiolla jokaisella pikselin kymmenykselläkin on merkitystä. Seuratun objektin keskipisteen tarkentamiseen käytetyt vahvat kulmapisteet voidaan lukea trackedObjectPoints()-funktiolla, ja seuratun objektin rotaation kulma saadaan trackedObjectAngle()-funktiosta.

6 Käyttöliittymäprototyyppejä

Tässä luvussa kuvataan MotionKit-kirjastoa hyödyntäviä käyttöliittymäprototyyppejä kirjaston avulla toteutetuista konsepteista. Toteutetut prototyypit on tehty lähinnä esittelyksi MotionKit-kirjaston käytöstä, ja niiden tarkoitus on johdatella kirjaston käyttäjää uusien käyttöliittymäkonseptien kehittelyyn.

Käyttöliittymäprototyypit esitellään kuvailemalla lyhyesti konseptin toimintaa, antamalla MotionKit-kirjaston alustuskonfiguraatio konseptille ja kuvailemalla sanallisesti, miten konseptissa käytetään MotionKit-kirjaston palauttamaa tietorakennetta. Käyttöliittymäprototyypeistä on kuvattu videoita, joiden sisältö esitellään liitteessä 1. Videotallenteet löytyvät DVD-muodossa erillisenä liitteenä.

3D-ympäristön kameran ohjaus

Tämä käyttöliittymäprototyyppi vastaa kesällä 2008 tehdyn pilottiprojektin toimintaa, mutta kypsemmän MotionKit-kirjaston avulla toteutus toimii vakaammin. MotionKit kuvaa käyttäjää web-kameralla, ja sovellus käyttää kuvattujen kasvojen sijaintia ja rotaatiota 3D-näkymän kameran ohjaamiseen. Käyttöliittymässä esitetään erikseen myös web-kameran antama kuva, johon on piirretty indikaatiot tunnistetusta objektista, sijainnista ja vahvoista kulmapisteistä. Alustukseen käytettiin seuraavaa konfigurointia (Kuvio 13).

```
1    MotionKit::MotionKitConfiguration config;
2    config.captureSource = MotionKit::ECaptureSourceCamera;
3    config.objectType = MotionKit::EObjectTypeFace;
4    config.objectAction = MotionKit::EObjectActionDetectAndTrack;
5    config.imageType = MotionKit::EImageTypeSourceWithOverlays;
```

Kuvio 13: MotionKit-kirjaston konfigurointi 3D-ympäristön kameran ohjaukseen

Sovellus käyttää MotionKit-kirjaston palauttamaa tietorakenneluokkaa image()-funktion palauttaman kuvan piirtämiseen sellaisenaan QLabel-widgetissä, ja trackedObject() sekä trackedObjectAngle() funktioiden palautteen välittämiseen QGLWidgetille, joka siirtää tietojen perusteella 3D-näkymän kameran sijaintia. Tämä prototyyppiesimerkki on myös demonstraatio siitä, että MotionKit-kirjaston säikeiden käytön ansiosta sen kanssa voidaan samaan aikaan suorittaa muitakin raskaita toimintoja, kuten 3D-grafiikkaa.

Työpöydän näkymän siirtyminen

Tähän käyttöliittymäprototyypin konseptiin viitattiin epäsuorasti käyttöliittymää käsittelevän luvun 2.2 esimerkissä. MotionKit kuvaa käyttäjää web-kameralla ja käyttäjän kasvojen kallistuessa oikealle tai vasemmalle työpöytää laajennetaan samaan suuntaan. Työpöytä ei ole tässä esimerkissä oikea käyttöjärjestelmän työpöytä vaan työpöytää mukaileva sovelluksen näkymä. Alla olevassa kuvassa esitetään prototyypin käyttämä konfigurointi (Kuvio 14).

```
1     MotionKit::MotionKitConfiguration config;
2     config.captureSource = MotionKit::ECaptureSourceCamera;
3     config.objectType = MotionKit::EObjectTypeFace;
4     config.objectAction = MotionKit::EObjectActionDetectAndTrack;
5     config.imageType = MotionKit::EImageTypeNone;
```

Kuvio 14: MotionKit-kirjaston konfigurointi työpöydän näkymän siirtämiseen

Sovellus käyttää MotionKit-kirjaston palauttamaa tietorakenneluokkaa aktiivisen näkymän siirtämiseen oikealle ja vasemmalle trackedObjectAngle()-funktion palautteen mukaan.

Sovelluksen koon muuttaminen

Tässä käyttöliittymäprototyypissä MotionKit kuvaa käyttäjää web-kameralla ja sovelluksen koko muuttuu seuraten käyttäjän kasvojen etäisyyttä näytöstä, jos sovellus on aktiivisena. Jos kaikki sovellukset toimisivat näin, käyttäjä saisi yleisnäkymän avoinna oleviin sovelluksiin nojautumalla taaksepäin, ja näkymän vain aktiiviseen sovellukseen nojautumalla eteenpäin. Alla olevassa kuvassa esitetään prototyypin käyttämä konfigurointi (Kuvio 15).

```
1     MotionKit::MotionKitConfiguration config;
2     config.captureSource = MotionKit::ECaptureSourceCamera;
3     config.objectType = MotionKit::EObjectTypeFace;
4     config.objectAction = MotionKit::EObjectActionDetectAndTrack;
5     config.imageType = MotionKit::EImageTypeNone;
```

Kuvio 15: MotionKit-kirjaston konfigurointi sovelluksen koon muuttamiseen

Sovellus käyttää MotionKit-kirjaston palauttamaa tietorakenneluokkaa sovelluksen pääikkunan koon muuttamiseen trackedObject()-funktion palauttaman alueen koon mukaan. Jos koko ylittää tai alittaa puolet näytön koosta, asetetaan se koko ruudun tilaan tai normaaliin tilaan.

Käyttäjän läsnäoloon reagoiva näytönsäästäjä

Tässä käyttöliittymäprototyypissä MotionKit kuvaa käyttäjää web-kameralla ja sovellus asettaa näytönsäästäjätilan päälle, jos se ei havaitse lainkaan käyttäjiä. Alustukseen käytettiin seuraavaa konfigurointia (Kuvio 16).

```
1    MotionKit::MotionKitConfiguration config;
2    config.captureSource = MotionKit::ECaptureSourceCamera;
3    config.objectType = MotionKit::EObjectTypeFace;
4    config.objectAction = MotionKit::EObjectActionDetect;
5    config.imageType = MotionKit::EImageTypeNone;
```

Kuvio 16: MotionKit-kirjaston konfigurointi läsnäoloon reagoivaan näytönsäästäjään

Sovellus käyttää MotionKit-kirjaston palauttamaa tietorakenneluokkaa käyttäjien lukumäärän selvittämiseen detectedObjects()-funktion palauttaman listan koon perusteella. Jos lukumäärä ei muutu 5 sekuntiin ja tila on vaihtunut, asetetaan näytönsäästäjä pois päältä tai päälle.

7 Loppusanat

Opinnäytetyön tavoitteena oli tutkia tietokonenäön hyödyntämistä havaitsevissa käyttöliittymissä ja toteuttaa tarkoitukseen sopiva apukirjasto mahdollisimman helppokäyttöisellä rajapinnalla. Tehty toteutus tuli dokumentoida riittävällä tarkkuudella, ja kirjaston käyttömahdollisuuksia tuli esitellä prototyyppikäyttöliittymien avulla. Konstruktiivisen osuuden oli sovelluttava Qt-sovelluskehityksen itseopiskelumateriaaliksi, ja sitä piti voida käyttää ohjelmistokehityshankkeiden havaitsevien käyttöliittymien toteutuksessa sekä Qt-osaamisen referenssinä.

Opinnäytetyölle asetetut tavoitteet saavutettiin. Toteutettu MotionKit-kirjaston toiminnallisuus täyttää sille asetetut vaatimukset, vaikkei se täydellinen olekaan. Toteutuksen keskeisin osuus on kuvattu tässä opinnäytetyöraportissa. Vaikkei opinnäytetyöraportin viimeistelyn hetkellä kirjastoa vielä hyödynnetäkään sovelluksissa, löytyy se omasta CV:stäni Qt-referenssinä, eikä kalpene lainkaan formaalimman Qt-koulutuksen tai Qt-asiakashankkeiden rinnalla.

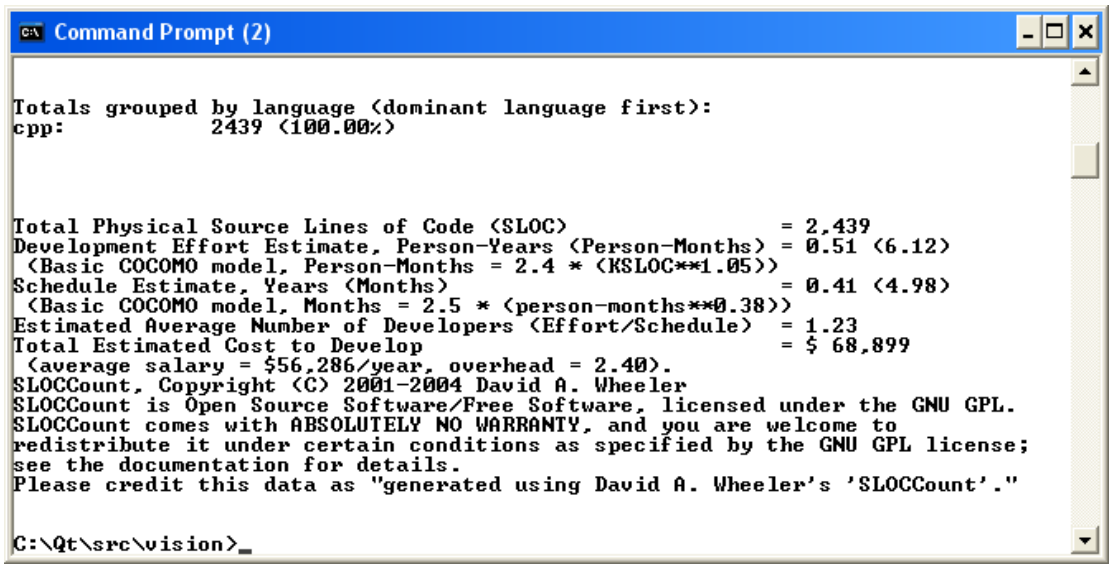
MotionKit-kirjaston toiminnallinen rajaus oli opinnäytetyön toteutuksen vaikein osuus. Kirjastoon jäi nykyisessä muodossaan tiettyjä rajoituksia, joita ilman se mahdollistaisi mielenkiintoisia konsepteja. Toisaalta rajoitukset ovat perusteltavissa käytettävissä olevan ajan ja työmääräkiintiön vuoksi. Kirjaston jatkokehitys on kuitenkin mahdollista.

Yksi puuttuvista ominaisuuksista on mahdollisuus seurata useita tunnistettuja objekteja samaan aikaan. Tämän ominaisuuden avulla olisi mahdollista toteuttaa multi-touch kosketusnäytöistä tuttuja ideoita, kuten kuvan zoomaus siirtämällä kämmeniä lähemmäksi tai kauemmaksi toisistaan. Nykyiselläkin kirjastolla tuo olisi mahdollista toteuttaa hyödyntämällä sen tarjoamaa usean objektin samanaikaista tunnistusta, mutta liikkeen seuraaminen pelkän tunnistuksen avulla vaatisi sovelluksen puolelle logiikkaa estämään satunnaisesti tunnistamatta jäävien objektien aiheuttamat häiriöt ja sijainnin epätarkkuuksiin liittyvät ongelmat.

Toinen toteuttamatta jäänyt ominaisuus on seurattavan objektin x-, y- ja z-akselin rotaation muutokset. Nykyisellään objektista tunnistetaan vain x-akselin rotaatio, mikä ei esimerkiksi riitä selvittämään, mihin suuntaan kuvattavien kasvojen nenä näyttää. Tä-

mäkin olisi mahdollista laskea objektista tunnistettujen vahvojen kulmapisteiden etäisyyksien muutoksilla toisiinsa ja objektin keskipisteeseen nähden, mutta toteutuksen oikea paikka tälle olisi kirjastossa eikä sovelluksessa.

Nykyisessä muodossaan kirjaston laajuus osoittautui kuitenkin kohtuulliseksi yhden henkilön toteutettavaksi, kuten voidaan lukea seuraavasta kuvasta, joka esittää lähdekoodeista tehdyn SLOCCount-yhteenvedon (Kuvio 17). SLOCCount esittää lähdekoodin rivimäärään perustuvia COCOMO-mallin mukaisia arvioita työmäärästä.



Kuvio 17: SLOCCount-yhteenveto MotionKit-kirjaston lähdekoodista

MotionKit-kirjaston julkaisemisesta ei ole tehty vielä opinnäytetyöraportin kirjoittamisen hetkellä päätöksiä, eikä kirjastoa siis ole julkisesti saatavilla. Se on mahdollista julkaista myöhemmin joko binäärinä tai avoimena lähdekoodina, tai se voidaan pitää suljettuna toimeksiantajan omistuksessa, jos halutaan säilyttää mahdollisuus sen kaupalliseen hyödyntämiseen. Tämä opinnäytetyöraportti soveltuu kuitenkin pohjatiedoksi niille tietokonenäöstä kiinnostuneille kehittäjille, jotka haluavat kokeilla OpenCV:n algoritmeja itse tai toteuttaa oman vastaavan kirjastonsa. Myös OpenCV sisältää tarkoitukseen hyviä esimerkkejä.

Itse sain opinnäytetyöstäni paljon irti. Pääsin hyödyntämään yhdeksän vuoden ohjelmistotyökokemusta ja opinnoista saamaani osaamista projektissa, joka kiinnosti itseäni myös henkilökohtaisella tasolla. Opinnäytetyön luonteen vuoksi luin myös paljon aihepiiriin liittyvää taustakirjallisuutta, mikä ei ole lainkaan tapaistani, mutta mikä osoittau-

tui hyödylliseksi perusteltaessa ohjelmointiteknisiä ratkaisuja tai rakennettaessa riittävää teoriapohjaa tehtävälle toteutukselle.

Tietokonenäön tarjoamien mahdollisuuksien edessä nöyrtyneenä voin kuitenkin myöntää, että MotionKit-apukirjasto vasta raottaa tietokonesovelluksen uuden näköaistin täyttä hyödyntämistä ja jättää paljon tilaa jatkokehitykselle ja uusille havaitsevien käyttöliittymien innovaatioille.

Lähteet

Abowd, Gregory D; Beale, Russell; Dix, Alan; Finlay, Janet 2004. Human-Computer Interaction, 3rd Edition. Pearson Education Limited. ISBN 0130-461091.

Blanchette, Jasmin & SummerField, Mark 2008. C++ GUI Programming with Qt 4, 2nd Ed. Westford, Massachusetts. ISBN 0-13-235416-0.

Bradski, Gary & Kaehler, Adrian 2008. Learning OpenCV, 1st Edition. [online] [viitattu 23.11.2008]. http://proquestcombo.safaribooksonline.com/9780596516130/preface

Hartley, Richard & Zisserman, Andrew 2004. Multiple View Geometry in Computer Vision. Cambridge University Press. ISBN 0-511-18711-4 [online] [viitattu 18.2.2009] [ebrary]

Norman, Donald Arthur. 1990. Why Interfaces don't work. Teoksessa Laurel, Brenda (toim.) The Art of Human-Computer-Interface Design. Addison-Wesley Publishing Company. ISBN 0-201-51797-3.

Qt 4.4 Whitepaper 2008. Nokia Corporation. [online] [viitattu 18.2.2009]. http://www.qtsoftware.com/files/pdf/qt-4.4-whitepaper

Turk, Matthew 1998. Moving from GUIs to PUIs. [online] [viitattu 18.2.2009]. http://research.microsoft.com/pubs/69680/tr-98-69.pdf

Liitteet

Liite 1: Kuvaus käyttöliittymäprototyyppien videotallenteista

Tässä kuvauksessa esitellään MotionKit-apukirjastoa hyödyntävien käyttöliittymäprototyyppien havainnollistamiseksi tehtyjen videotallenteiden sisältö sanallisesti. Itse videotallenteet löytyvät DVD-muodossa erillisenä liitteenä.

Video 1 on kuvattu kesällä 2008, ja se esittää pilottiprojektina tehdyn FaceTracker-sovelluksen. Sen käyttöliittymä koostuu kahdesta widgetistä: 3D-näkymästä ja web-kameran näkymästä. Web-kamerasta luetun kuvan päälle piirretään grafiikkaa osoittamaan tunnistettu objekti (sininen ovaali), kulmapisteet (siniset pisteet) sekä keskipiste (valkoinen piste), jonka perusteella ohjataan 3D-näkymän kameraa. Videon alussa näytetään web-kameran näkymässä, kuinka objektin seuranta reagoi pään kääntelyyn ja sivuttaisiin liikkeisiin. Tämän jälkeen esitetään liikkeen aiheuttamat vaikutukset 3D-näkymän kameran sijaintiin.

Video 2 esittää 3D-käyttöliittymäprototyyppiä, joka pohjautuu alkuperäiseen FaceTracker-pilottiin. Videon alussa näytetään, kuinka käyttöliittymän 3D-näkymä seuraa kasvojen liikkeitä aivan kuten alkuperäisessäkin versiossa. Tämän jälkeen esitetään käyttäjän kasvojen päälle piirretyt indikaatiot, joissa on uutena mukana tunnistettu objekti (keltainen neliö). Seuraavaksi demonstroidaan uutta MotionKit-kirjaston vikasietoisuutta aiheuttamalla objektin seurantaan tahallinen häiriö pyyhkäisemällä kasvojen yli kädellä. MotionKit harhautuu hetkellisesti seuraamaan kättä, mutta toipuu pian havaittuaan, että objektin keskipiste (valkoinen piste) on ajautunut tunnistetun objektin (keltainen neliö) ulkopuolelle. Sitten käyttäjä nostaa näkyviin paperin, johon on piirretty kasvot, jotka MotionKit tunnistaa seuratessaan samaan aikaan alkuperäistä objektia. Lopuksi käyttäjä peittää toisen silmänsä, jolloin MotionKit ei enää tunnista kasvoja (keltainen neliö katoaa), mutta pystyy edelleen seuraamaan käyttäjää aiemmin tunnistetun objektin tietojen perusteella.

Video 3 esittää käyttöliittymäprototyyppiä, jossa työpöytää voidaan laajentaa käyttäjän liikkeen perusteella. Videon alussa näkyy täysi työpöytä, jolla on avoimena kolme

sovellusta ja vain vähän vapaata tilaa. Käyttäjä avaa uuden sovelluksen, mutta ei saa sovitettua sitä jäljellä olevaan tilaan. Käyttäjä siirtää kasvojaan oikealle, jolloin työpöydän sisältö liukuu vasemmalle tuoden työpöydälle uutta tilaa, johon käyttäjä asemoi avaamansa sovelluksen. Käytön jälkeen sovellus suljetaan ja käyttäjä siirtää kasvonsa takaisin vasemmalle, jolloin aiemmat sovellukset liukuvat vasemmalta esiin.

Video 4 esittää käyttöliittymäprototyyppiä, jossa käyttäjän pään etäisyys vaikuttaa sovelluksen ikkunan kokoon ja tilaan. Alussa käyttäjän kasvot ovat kaukana näytöstä, ja sovellus näkyy normaalitilassa työpöydän keskiosassa. Kun käyttäjä siirtää kasvojaan lähemmäksi näyttöä, sovellus suurenee, ja menee lopulta koko ruudun tilaan. Kun käyttäjä loitontaa kasvonsa takaisin kauemmaksi näytöstä, sovellus palautuu normaalitilaan ja pienenee sitten etäisyyden yhä kasvaessa. Sovellus reagoi käyttäjän kasvojen etäisyyteen vain ollessaan valittuna.

Video 5 esittää käyttöliittymäprototyyppiä, jossa ruudunsäästäjä reagoi kasvojen näkyvyyteen. Alussa käyttäjä on ruudun ääressä ja näyttö on toiminnassa. Kun käyttäjä siirtyy kauas vasemmalle pois web-kameran näköpiiristä, eikä taustalla oleva sovellus tunnista enää kasvoja, näytön virta kytkeytyy pois päältä. Kun käyttäjä palaa takaisin ruudun ääreen, näyttöön kytkeytyy jälleen virta.