Tero Oleander

Digitaalisten viivakameroiden kuvankaappaus-komponentin suunnittelu ja toteutus

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tietotekniikka

Insinöörityö

7.5.2015

Metropolia

Tekijä(t) Otsikko Sivumäärä Aika	Tero Oleander Digitaalisten viivakameroiden kuvankaappauskomponentin suunnittelu ja toteutus 54 sivua + 2 liitettä 7.5.2015
Tutkinto	Insinööri (AMK)
Koulutusohjelma	Tietotekniikka
Suuntautumisvaihtoehto	Ohjelmistotekniikka
Ohjaaja(t)	Project Manager Kari Sirén Lehtori Simo Silander

Digitaalisia viivakameroita valmistavalle JAI Oy:lle oli tarpeen kehittää uusi kameroiden tuotantoa, testausta ja tuotekehitystä tukeva ohjelmisto. Ohjelmistoprojektin kehitysympäristöksi valittiin LabView, jossa kuitenkin ilmeni puutteita. Tämän johdosta projektista erotettiin ohjelmiston tarvitseman kuvankaappaustoimintojen ja -rajapinnan toteutus.

Insinöörityön tavoitteena oli suunnitella ja toteuttaa Windows-ympäristössä käytettävä ActiveX-ohjelmistokomponentti, joka voidaan integroida kuvankaappaustoiminnallisuutta tarvitseviin asiakasohjelmiin, ensisijaisesti kehitteillä olleeseen kameroiden testausohjelmistoon. Komponentin tehtävänä on ohjata PCIExpress-väylään liitettäviä kuvankaappauskortteja, vastaanottaa sen kautta kameralta tulevaa kuvadataa, suorittaa datalle esikäsittelyä ja lopuksi välittää käsitellyt kuvadatat asiakasohjelmalle jatkokäsittelyä varten. Komponentin on kyettävä suoriutumaan hallitusti nopeataajuuksisestakin kuvankaappauksesta.

Työssä perehdyttiin ensin väriviivakameran toimintaperiaatteeseen ja kehitteillä olevaan testausjärjestelmään, minkä pohjalta aloitettiin kuvankaappauskomponentin määrittely ja tekninen suunnittelu. Asiakasohjelmarajapinta, kuten myös sisäiset rajapinnat ja rakenne pyrittiin suunnittelemaan mahdollisimman yksinkertaisiksi helpon käytettävyyden ja laajennettavuuden vuoksi. Toteutuksessa päädyttiin käyttämään rinnakkaislaskentaa ja säikeitä tehokkuuden parantamiseksi. Säikeet ovat tarpeen myös odoteltaessa asynkronisia signaaleja, joita käytetään mm. uusista kaapatuista kuvista ilmoitettaessa.

Insinöörityön tuloksena syntyi TVIGC-kuvankaappauskomponentti, joka saavutti asetetut tavoitteet. Suorituskyvyssä ei kaikissa tilanteissa päästy aivan reaaliaikaiselle tasolle, mutta toiminta on silti hallittua, eli kuvia kaapataan siinä tahdissa kuin niitä ehditään käsitellä komponentissa ja sitä käyttävässä asiakasohjelmassa. Komponentti integroitiin JAI Oy:n uuteen testausohjelmistoon, josta on tullut yrityksen tärkeimpiä ohjelmistotyökaluja. Komponenttia on käytetty myös muissa projekteissa, ja näiden käyttökokemusten perusteella se on osoittautunut käyttökelpoiseksi ja nopeaksi välineeksi saada kuvankaappaustoiminnot käyttöön sovellusohjelmissa.

Avainsanat	viivakamera, kuvankaappauskortti, ActiveX, ohjelmistokomponentti, rinnakkaislaskenta

Author(s) Title Number of Pages Date	Tero Oleander Design and Implementation of an Image Grabbing Component for Digital Line Scan Cameras 54 pages + 2 appendices 7 May 2015
Degree	Bachelor of Engineering
Degree Programme	Information Technology
Specialisation option	Software Engineering
Instructor(s)	Kari Sirén, Project Manager Simo Silander, Senior Lecturer

New camera testing software was needed for production and development of digital line scan cameras at JAI Oy. LabView was chosen as development environment for the software project. During the development some deficiencies came up with LabView, which led into separation of the implementation of the image capturing functions and interface.

The aim of this thesis was to develop the ActiveX software component for Windows which can be integrated with client applications, especially the new camera testing software, that need the image grabbing functionality. The purpose of the component is to control the PCIExpress-bus connected frame grabbers, receive image data from camera via a frame grabber, pre-process the captured image data and finally pass the image data to the client application for further handling. The component has to be capable to manage also the high frequency grabbing processes.

At first the principles of colour line scan camera and the new testing software under development were familiarized. Based on that the design of the image grabbing component was started. The client interface as well as all internal interfaces and architecture of the component were aimed to be as simple as possible for easy usability and expandability. Parallel computing and threads were chosen during the implementation phase to improve the performance and for waiting the asynchronous signals which were used for example to notify the system of new captured frames.

As a result of this thesis new software component named TVIGC was created which achieved the set targets. The real time performance was not reached in all of the cases, but still the operation of the component stays well under control so that the images are grabbed at the same rate than they can be processed by the component and the client application. The component was integrated with the new camera testing software which has become one of the most important software tools at JAI Oy. The component has been used also with other software projects. With that experience the TVIGC component has been proved to be a practical and fast way to get the image grabbing functionality added into applications.

Keywords	line scan camera, frame grabber, ActiveX, software component, parallel computing

Sisällys

Lyhenteet ja määritelmät

ActiveX	Microsoftin kehittämä ohjelmistokomponenttitekniikka, jota voidaan käyttää Windows-sovellusten laajentamiseen tai toiminnallisena elementtinä internetsivuilla.
API	*Application Programming Interface.* Ohjelmointirajapinta, joka tarjoaa liitynnän ohjelmistoon tai ohjelmistokomponenttiin muiden sovellusten käytettäväksi. Se koostuu tyypillisesti dynaamisesti linkitettävistä DLL- tai staattisista LIB-ohjelmakirjastoista.
DLL	*Dynamic Link Library.* Dynaamisesti ohjelmaan ajon aikana linkitettävä funktiokirjasto.
LabView	National Instrumentsin kehittämä graafinen ohjelmointiympäristö.
MFC	*Microsoft Foundation Classes.* Microsoftin kehittämä Windows API:n kapseloiva sovelluskehys.
SDK	*Software Development Kit.* Ohjelmistonkehityspaketti, joka sisältää ohjelmointirajapinnan (API) lisäksi yleensä vähintään dokumentaation ja esimerkkisovelluksia.

1 Johdanto

Helsingissä toimivassa viivakameroita valmistavassa yrityksessä, JAI Oy:ssä, oli jo jonkin aikaa suunniteltu kameroiden tuotannossa tarvittavien ohjelmistotyökalujen uusimista. Kameroiden kokoonpanoa suoritetaan osittain tietokoneavusteisesti, ja kameraa testataan ja kalibroidaan useammassa vaiheessa. Näissä kaikissa tuotantovaiheissa on ennen kaikkea tarve nähdä ja analysoida kameran tuottamaa kuvadataa. Lisäksi erityisesti kameroiden tuotantoprosessin viimeisen vaiheen, lopputestauksen, rutiininomaisia toimintoja olisi tarpeen automatisoida ja mahdollisesti saada joitakin vertailukelpoisia parametreja kameran optisen suorituskyvyn suhteen. Toinen lähtökohta oli, että samaa ohjelmistoa tulisi voida käyttää kaikkien kameran tuotantoprosessin vaiheiden lisäksi kameroiden tuotekehitys- ja tutkimusprojekteissa.

Digitaalinen prismaperustainen väriviivakamera koostuu kolmesta tai useammasta viivasensorista, joihin kohdistetaan eri aallonpituuksista valoa. Tyypillisesti valo jaetaan prismaa käyttäen kolmeen eri aallonpituusalueeseen, näkyvän aallonpituusalueen punaiseen, vihreään ja siniseen valoon, joista muut värisävyt voidaan muodostaa. Lisäksi näkyvän valon aallonpituusalueen ulkopuolista säteilyä, eli infrapuna- tai ultravioletisäteilyä, voidaan erotella. Viivasensorit koostuvat vaihtelevasta määrästä yhdessä rivissä olevia kuvapisteitä eli pikseleitä. Kameralla on tarkoitus kuvata liikkuvia kohteita (tai itse kameraa on liikutettava), jolloin nopeassa tahdissa valotettavia pikselirivejä yhdistettäessä voidaan muodostaa normaaleja kaksiulotteisia kuvia. Näin ollen kamerat soveltuvat parhaiten nopeiden liikkuvien kohteiden, kuten liukuhihnoilla liikkuvien esineiden kuvaamiseen vaikkapa laadunvalvontatarkoituksessa tai esimerkiksi maalikameroiksi rataurheilulajeihin.

Käynnistetyn testausohjelmistoprojektin toteutusympäristöksi päätettiin valita erityisesti testausjärjestelmissä paljon käytetty National Instrumentsin LabView. LabView'n etuina ovat suhteellisen helppo opittavuus ja nopea ohjelmistonkehitys havainnollisen graafisen ohjelmointikielensä ansiosta sekä laaja tuki erilaisille oheislaitteille. Koska yrityksessä ei ollut kuitenkaan ketään todellista LabView-osaajaa ja huomattiin, että näennäisestä helppoudesta huolimatta ohjelmointiympäristön todellinen oppiminen olisi vaatinut paljon resursseja, päätettiin antaa LabView-osuus sitä käyttämään erikoistuneelle kotimaiselle insinööritoimistolle.

Pian huomattiin kuitenkin käytännössä eräitä LabView'n rajoituksia: suorituskyky on heikompi kuin C++-ohjelmissa ja näennäisesti hyvä laitteistotuki edellyttää, että laitteistovalmistaja on tehnyt LabView'tä varten ajurit. Tässä projektissa käytettiin kuvadatan siirtämiseksi kameralta PC-tietokoneelle PCIExpress-väylään liitettävää BitFlow'n valmistamaa kuvankaappauskorttia. Sen ongelmana olivat erityisesti puutteelliset LabView-ajurit. Kortin valmistaja oli panostanut enemmän C++-ohjelmointirajapinnan kehittämiseen. LabView'ssä on kuitenkin varsin hyvä tuki erilaisille ohjelmistokomponenteille, kuten DLL:lle (Dynamic Link Library) tai ActiveX:lle. Tämän seurauksena projekti päätettiin toteuttaa siten, että kuvankaappauskortin ohjaus hajautetaan omaan ohjelmistokomponenttiin, joka integroidaan varsinaiseen testausohjelmistoon. Tällöin testausohjelmistosta tulee samalla täysin riippumaton mistään tietystä kaappauskortista, ja toisaalta uusien korttimallien käyttöönotto helpottuu. Lisäksi kaappauskomponentti mahdollistaa kaappaustoimintojen helpon käyttöönoton myös muissa sovelluksissa.

Tämän työn tavoitteena oli suunnitella ja toteuttaa Windows-ympäristössä toimiva ActiveX-ohjelmistokomponentti helpottamaan kuvankaappauslaitteiston integrointia haluttuun kuvadatankäsittely- ja kaappausohjelmistoon. Pyrkimyksenä oli suunnitella komponentin sisäinen rakenne ja rajapinnat siten, että kuvankaappauskorttien integrointi komponenttiin ja toisaalta komponentin integrointi kaappausohjelmistoon olisivat mahdollisimman yksinkertaisia toimenpiteitä. Myös tehokkuuteen tuli kiinnittää huomiota. Tehokkuuden parantamiseksi komponenttiin sijoitettiin kuvadatan esikäsittelyoperaatioita, joiden oletettiin olevan C++-kielellä toteutettuina nopeampia.

Työssä esitellään ensin projektin syntyä ja taustoja, jonka jälkeen perehdytään prismaperustaisen väriviivakameran toimintaperiaatteeseen sekä kuvadatan siirtotekniikoihin kamerasta PC:lle. Tämän jälkeen tutustutaan projektissa käytettäviin ja tarvittaviin ohjelmistotekniikoihin. Lopuksi käydään läpi komponenttiprojektin toteutusta ja ratkaisuja sekä työn tuloksena syntyneen ohjelmistokomponentin rakennetta, toimintaperiaatetta ja käyttöä.

2 Kameratestausohjelmistoprojektin esittely

Kasvaneet tuotantomäärät sekä tarve kehittää tuotantoprosessia ja -menetelmiä entistä tarkempien ja laadukkaampien kameroiden valmistamiseksi ovat aiheuttaneet tarpeen uudenlaisten mittareiden sekä tuotantoa ja tuotekehitystä tukevan ja tehostavan atk-pohjaisen järjestelmän kehittämiselle. Vanhat ohjelmistot olivat jo tulleet tiensä päähän, eikä niitä ollut enää mielekästä lähteä päivittämään.

Yrityksen tavoitteena on ollut pitää huoltotapaukset mahdollisimman vähäisinä. Huoltotapauksia aiheuttavat varsinaisten laitevikojen lisäksi myös kameroiden yleisten ja asiakaskohtaisten asetusten konfiguroinnissa sattuneet huolimattomuusvirheet. Käsin tehtävissä operaatioissa huolimattomuusvirheiden riski on suuri, jota voitaisiin automatisoinnin myötä huomattavasti pienentää.

Uudessa järjestelmässä yhdistyisivät tarkempi mittaustekniikka, monipuolisemmat mittausmenetelmät ja automaattinen testaus ja -konfigurointi.

2.1 JAI Oy

JAI Oy toimii konenäköalalla ja suunnittelee, kehittää, valmistaa ja myy lähinnä konenäköjärjestelmiin teollisuuden tarpeisiin digitaalisia, mustavalkoviivakameroita tai prismaperustaisia väriviivakameroita. Tyypillisiä käyttökohteita ovat suurta nopeutta ja erottelukykyä vaativat sovellukset, kuten laadunvalvonta- ja lajittelujärjestelmät, mutta toisaalta kamerat soveltuvat mainiosti myös mm. rataurheilulajien maalikameroiksi tai vaikkapa elokuvafilmien digitointiin. Yrityksen kilpailuvaltteina ovat korkean laadun lisäksi tuotteiden räätälöitävyys asiakkaan tarpeiden mukaan sekä nopeat toimitusajat. Markkina-alueena toimii koko maailma: suurin osa tuotannosta menee vientiin. Yritys on toiminut alalla jo vuodesta 1986 noin 10 henkilön voimin. JAI Oy:n (aiemmin TVI Vision Oy) on omistanut vuodesta 2012 tanskalainen JAI A/S, jonka päätoimiala on konenäkö ja erityisesti konenäkökameroiden kehitys ja valmistus. JAI Oy toimii Finntemet-konsernin tiloissa Helsingin Herttoniemessä.

2.2 Aiempi järjestelmä ja aiemmat metodit

Yrityksessä on aiemmin ollut käytössä useita erilaisia järjestelmiä eri käyttötarpeisiin. Optinen kokoonpano on suoritettu vanhaa DOS-pohjaista ohjelmaa apuna käyttäen, jonka toiminnot ovat ajan myötä käyneet puutteellisiksi. Kameroiden testauksessa, tuotekehityksessä ja kameroiden esittelemisessä käytetty Windows-ohjelmisto on myös jäänyt ajastaan jälkeen, ja uusia ominaisuuksia ja toimintoja kaivattiin. Ohjelmiston laajennettavuus on heikko, se on sidottu tiukasti tietynlaiseen laitteistoon (kuvankaappauskorttiin) ja lisäksi ohjelmisto oli todettu jossain määrin epävakaaksi. Ohjelmisto ei myöskään mahdollistanut minkäänlaista toimintojen automatisointia. Yrityksen tarpeisiin on myös kehitetty toiminnanohjausjärjestelmä, johon talletetaan keskitetysti muun muassa tuotannon ja huollon raportit sähköisesti, mikä on tehty käsin tähän asti.

2.3 Tarvekartoitus

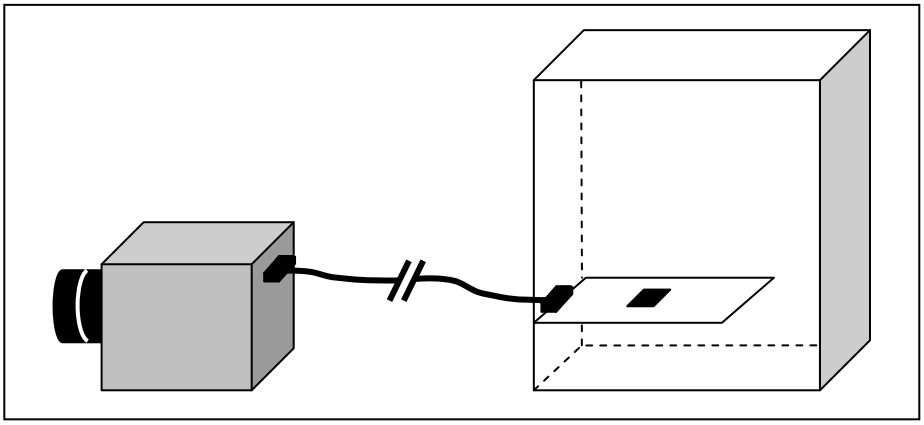
Jo lähtökohtaisesti voidaan todeta, että tarvetta vastaava ohjelmisto on luonteeltaan sellainen, että valmista ohjelmistoa ei ole olemassa, vaan se on kehitettävä räätälöidysti. Aiemmin käytössä olleet ohjelmistot ovat niin ikään räätälöidysti tehtyjä ja pääosin yrityksen omaa tuotantoa. Niiden muuttaminen ja laajentaminen nykyisiä tarpeita vastaaviksi todettiin epämielekkääksi ratkaisuksi; suurin osa koodista olisi pitänyt joka tapauksessa kirjoittaa uusiksi. Tavoitteena oli kehittää järjestelmä, joka tulisi sekä tuotannon, tuotekehityksen että tuote-esittelijöiden käyttöön. Ohjelmiston tulisi myös linkittyä yrityksen toiminnanohjausjärjestelmään mahdollistaen raporttien viemisen automatisoidusti.

2.4 Työnjako

Projektin laajuudesta ja erikoisosaamista vaativista osista johtuen projekti päätettiin jakaa useamman tahon kesken. Labview-ohjelmointiosuuden sekä osan algoritmien suunnittelun ja toteutuksen suoritti niihin erikoistunut kotimainen insinööritoimisto. JAI Oy:ssä keskityttiin järjestelmän toimintojen, käyttöliittymän ja käytettävyyden määrittelyyn sekä kuvankaappausrajapinnan suunnitteluun ja toteutukseen.

3 Kuvankaappauslaitteisto

Projektiin kuuluu oleellisena osana laitteistoa, joka koostuu kuvadataa tuottavasta kamerasta ja dataa vastaanottavasta laitteistosta. Tässä luvussa esitellään yleisesti digitaalikameroiden sekä viivakameran toimintaperiaatteet ja kuvankaappausjärjestelmän toimintaperiaate. Kuva 1 havainnollistaa käytettävää laitteistoa.



Kuva 1. Tyypillisen minimilaitteiston osat. Vasemmalla on kamera, joka on kytketty PC-tietokoneen sisälle asennettuun kuvankaappauskorttiin.

3.1 Väriviivakamera

Digitaalikameroiden kuvanmuodostus perustuu sähköiseen valoherkkään kennoon, joka koostuu tietystä määrästä kuvapisteitä eli pikseleitä. Kennon valottamisen aikana pikselit varautuvat sähköisesti fotoneista niihin kohdistuvan valon määrän eli intensiteetin ja valotusajan mukaisesti. [1.] Pikseleiden käyttäytymistä voidaan pitää melko lineaarisena, eli varaus kasvaa suorassa suhteessa valon määrän kasvaessa. Kun pikseli on kerännyt maksimimäärän valoa, se saavuttaa niin sanotun saturaatiotason, eikä se tämän jälkeen pysty keräämään enempää fotoneita valotuksen lisääntymisestä huolimatta. Kennon varauskykyä eli pimeän ja saturaation välistä eroa kutsutaan dynamiikaksi. [2.]

Kuvanoton jälkeen pikseleiden varaukset muunnetaan A/D-muunninpiiriä käyttäen digitaalisiksi arvoiksi. Muuntimesta riippuu, kuinka monta eri jännitetasoa se pystyy erottelemaan, toisin sanoen, kuinka monta eri valotasoa eli sävyä on mahdollista erotella.

Tästä käytetään myös nimitystä bittisyvyys. Esimerkiksi 12-bittinen muunnin pystyy erottelemaan 2^{12} = 4096 eri jännitetasoa. [2.]

Värikamerassa jokaiselle värikanavalle on oltava omat pikselinsä. Kameralle tulevasta valosta ohjataan tietyt aallonpituudet tietyille pikseleille käyttäen suodattimia tai valonjakoprismaa. Tyypillinen värikamera on RGB-värimallin mukainen, eli mittaa näkyvän aallonpituuden punaista, vihreää ja sinistä valoa, joista voidaan muodostaa erilaisia värisävyjä. Kamera voi kerätä säteilyä myös IR- (infrapuna) tai UV (ultravioletti) -alueilta. Eroteltavissa olevien värisävyjen määrä n_c voidaan laskea kaavalla 1.

$$n_c = \left(2^b\right)^{n_{ch}} \quad (1)$$

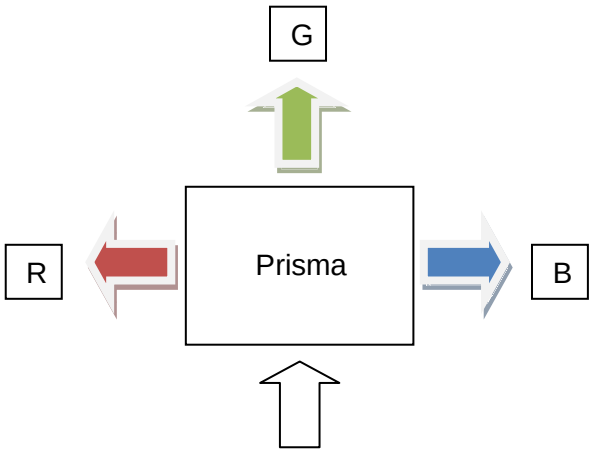
b on värikanavan bittisyvyys

n_{ch} on värikanavien lukumäärä

Esimerkiksi kameralla, jossa on kolme värikanavaa ja kunkin kanavan bittisyvyys on 8 bittiä, pystytään erottelemaan yhteensä noin 16,7 miljoonaa sävyä. [2.]

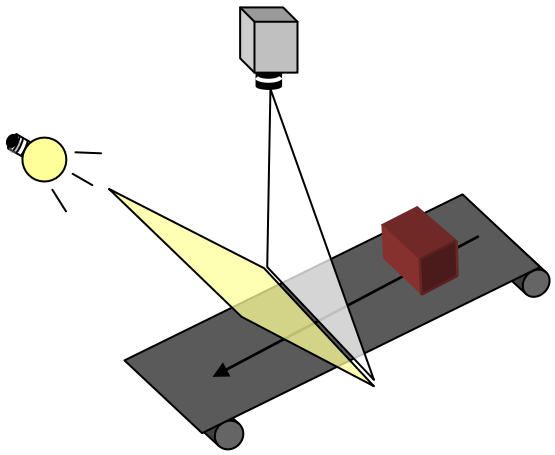
Digitaalikamerat voidaan jakaa kennonsa perusteella kahteen ryhmään: matriisi- ja viivakameroihin. Nimensä mukaisesti matriisikameroiden kenno on 2-ulotteinen ja sisältää (X·Y) -määrän pikseleitä. Viivakameroiden "kennossa" on pikseleitä ainoastaan yhdessä rivissä. Tästä johtuen viivakameroiden "kennoa" kutsutaankin mieluummin vektoriksi tai sensoriksi. Kennossa olevien pikseleiden määrästä käytetään termiä resoluutio eli erottelukyky. Viivakameroiden tyypillinen pikselimäärä on noin 500…4000 kpl. Tässä työssä keskitytään prismaperustaisiin väriviivakameroihin, koska JAI Oy:n kamerat käyttävät nimenomaan tätä tekniikkaa. [3.]

Prismaperustaisessa väriviivakamerassa valo tulee prisman etupintaan ja jakautuu siitä eri värien mukaisiin kanaviin kuten kuvassa 2. Värikanavien päissä on omat sensorit mittaamassa valon määrää kullekin värille eli aallonpituusalueelle. Tärkeätä on, että kohteesta tuleva valo tulee kullekin pikselin osavärille tismalleen samasta pisteestä, samassa kulmassa ja samanaikaisesti. Valon jakaminen osaväreihin valonjakoprismaa käyttäen on eräs tarkimmista tekniikoista. [4.]



Kuva 2. Valonjakoprisman toiminta. Valo tulee etupintaan ja jakautuu osavärikanaviin.

Viivakameran (engl. line scan camera) toimintaperiaate perustuu kohteen skannaukseen, joten kamera on tarkoitettu käytettäväksi järjestelmissä, joissa joko kamera tai kohde liikkuu. Tyypillisesti viivakameralla voidaan kuvata esim. tehtaan tuotannon liukuhihnaa, jota on havainnollistettu kuvassa 3. Kamera kaappaa suurella nopeudella viivoja, joista voidaan koostaa kokonainen kuva. Täten muodostetun kuvan (engl. frame) koko on sensorin pikselimäärä · kaapattujen viivojen määrä. Kameran viivojenvalotustaajuus asetetaan kohteen liikenopeuden mukaiseksi. Yhteen kuvaan haluttujen viivojen määrä valitaan kohteeseen sopivasti, esim. siten, että liukuhihnalla kulkeva yksittäinen tuote mahtuu kuvaan.



Kuva 3. Tyypillinen viivakamerasovellus, jossa kuvataan liukuhihnaa pitkin kulkevia esineitä. Valaisuksi riittää kapea viivamainen valonlähde. [5.]

3.2 Kuvankäsittelylaitteisto

Kameran tärkein tehtävä on tuottaa mahdollisimman laadukasta ja tarkkaa kuvadataa, toisin sanoen pyrkiä toistamaan kohde mahdollisimman identtisenä kohteen muotoja ja värejä vääristämättä.

Muun kuvankaappauslaitteiston tehtävä on vastaanottaa ja käsitellä kameran tuottamaa kuvadataa, esimerkiksi analysoida liukuhihnalla kuvattuja tuotteita laadunvalvontatarkoituksessa. Lisäksi laitteiston avulla voidaan säädellä kameran toimintaa, kuten määrittää viivojenvalotustaajuutta ja valotusaikaa sekä muuttaa kameran ohjelmoitavia parametreja.

Laitteisto koostuu yleensä PC-tietokoneesta, johon kamera jotakin liitäntätapaa käyttäen kytketään. Tyypillisesti käytetään markkinoilla olevia konenäkökäyttöön suunniteltuja PCIExpress-väylään liitettäviä kuvankaappauskortteja. Korttien tehtävänä on tarjota nopea liityntä kameran ja PC:n välille jonkinlaista väylätekniikkaa käyttäen. Joissain tapauksissa voidaan käyttää myös Ethernet-liitäntää tai periaatteessa muitakin PC:n liitäntöjä, mutta useimmiten näiden käyttöä rajoittaa alhainen väylän nopeus. [6.]

Tiedonsiirtoon on käytössä valmistajakohtaisia sekä standardeja tekniikoita. Kokemuksen perusteella viime vuosina konenäkökäytössä liityntätekniikoista ovat vakiintuneet erityisesti Camera Link -standardi, Ethernet-väylää ja IP-protokollaa käyttävä GigE Vision -standardi, sekä uusimpana liityntätekniikkana CoaXPress-standardi, jossa kaikki tiedonsiirto tapahtuu yhtä tai useampaa tavanomaista koaksiaalikaapelia pitkin. Myös LVDS-signaaliin perustuvia epästandardeja liityntöjä on käytössä. Tällä hetkellä JAI Oy:n tuotannossa olevissa kameroissa on käytössä Camera Link- ja CoaXPress-standardien mukaiset liityntätekniikat. [6.]

Liityntästandardit määrittelevät tyypillisesti ainakin datan siirtoon käytetyt protokollat, dataformaatit, siirtonopeudet, mekaaniset liittimet ja kaapelit. Kaapelissa kulkee kuvadatan lisäksi kameran ohjaukseen ja konfigurointiin käytettävät signaalit.

Normaalisti korttien mukana toimitetaan kuvankaappaussovellusten tekoa varten korttivalmistajan tekemä ohjelmistonkehityspaketti eli SDK (Software development kit). SDK sisältää yleensä vähintään ohjelmointirajapinnan ja sen käyttöönottamiseksi tarvittavat aputiedostot, kuten otsikkotiedostot tai tyyppikirjastot, dokumentaation ja esimerkkiso-

velluksia. Ohjelmointirajapinta (API, Application programming interface) tarjoaa liityntänän ohjelmistoon tai ohjelmistokomponenttiin muiden sovellusten käytettäväksi. Se koostuu tyypillisesti dynaamisesti linkitettävistä DLL- tai staattisista LIB-ohjelmakirjastoista. [7.]

4 Käytetyt ohjelmistotekniikat

Tässä luvussa esitellään pääasiassa kaappauskomponentin toteutuksessa käytettyjä ohjelmistotekniikoita. Luvun lopussa esitellään LabView-ohjelmointiympäristö, jota käytettiin varsinaisen testausohjelmiston toteutuksessa. Koska kaappauskomponentti suunniteltiin kuitenkin ensisijaisesti tämän testausohjelmiston kanssa käytettäväksi, tuli suunnittelussa ottaa huomioon LabView-ympäristö ja sen rajoitteet.

4.1 MFC-sovelluskehys

MFC-sovelluskehys (Microsoft Foundation Classes Framework) on Microsoftin kehittämä C++-luokkakirjasto, joka on tehty helpottamaan ja nopeuttamaan Windows-sovellusten kehitystä. Se kapseloi Windows API:n, joka on Windowsin matalan tason ohjelmointirajapinta. Windows API:n funktioita voidaan kuitenkin kutsua suoraan myös MFC:tä käyttävistä sovelluksista. MFC helpottaa monimutkaisten ohjelmistotekniikoiden käyttöönottoa ja ohjelmointia, kuten käyttöliittymät tai ActiveX, tarjoamalla rungon, jonka päälle sovellusta voi lähteä kehittämään, asettamatta kuitenkaan rajoituksia kehitystyölle. Lisäksi se tarjoaa runsaasti apuluokkia muun muassa merkkijonojen käsittelyyn. MFC-sovelluksia voidaan ohjelmoida Microsoftin Visual Studio -sovelluskehittimellä. [8, s. 74-76; 9.]

MFC:n kirjasto-DLL:t voidaan linkittää sovellukseen joko staattisesti tai dynaamisesti. Mikäli MFC-kirjasto linkitetään sovellukseen dynaamisesti, tulee kirjaston saman version, jota sovelluskehityksessä käytettiin, löytyä asennettuna koneelta, jolla sovellusta ajetaan. Staattisesti linkitettäessä kirjasto liitetään osaksi sovellustiedostoa, mikä kasvattaa tiedoston kokoa. Tässä projektissa päädyttiin käyttämään dynaamista linkitystä. [8, s. 81.]

4.2 ActiveX-kontrollit

ActiveX-kontrollit ovat COM-tekniikkaan perustuvia ohjelmistokomponentteja, jotka on suunniteltu upotettaviksi niin sanottuun kontrollisäilöön, kuten Windows-sovelluksen käyttöliittymään tai internetsivulle tarjoten näille palveluja julkistamiensa rajapintojen kautta. Tässä työssä kontrollia käyttävästä säilösovelluksesta käytetään termiä asiakasohjelma, joka viittaa asiakas-palvelin-malliin (client-server). Kontrolli voidaan nähdä palvelimena, jota kontrollia käyttävä sovellus (asiakas) kutsuu. COM (Component Object Model) on Microsoftin kehittämä kieli- ja laiteriippumaton standardi, joka määrittelee, kuinka oliot kommunikoivat keskenään yhteistä protokollaa käyttäen. Käytännössä ne kuitenkin toimivat lähinnä vain Windows-järjestelmissä. Kontrollille toteutetaan yleensä käyttöliittymä, jolloin se voi toimia osana asiakasohjelman käyttöliittymää. Täten kontrollit vastaavat tavanomaisia Windows-kontrolleja, kuten mm. painikkeet ja tekstikehykset. Kontrollit voidaan kuitenkin toteuttaa myös ilman käyttöliittymää. [8, s. 341-345.] Kontrolleilla on rajoitukseton pääsy käyttöjärjestelmän palveluihin, mikä mahdollistaa niiden tekemisestä hyvin monipuolisia. Tästä syystä kontrollit ovat internetkäytössä varsin alttiita tietoturvauhille. Tässä projektissa tietoturvaan ei kiinnitetty erityistä huomiota, sillä toteutettua komponenttia on tarkoitus ajaa vain paikallisesti eikä verkon yli, ja siihen toteutettiin joka tapauksessa varsin laajasti koko järjestelmää käyttävää koodia. [10.]

ActiveX-kontrollit tukevat rajapintansa kautta käytettäviä ja muutettavia ominaisuuksia (properties), metodeja (methods) ja tapahtumia (events), joiden avulla kontrollin toimintaa ohjataan [8, s. 371]. Ominaisuudet ovat kontrollissa sijaitsevia jäsenmuuttujia, joita asiakasohjelma voi lukea ja kirjoittaa suoraan rajapinnan tarjoamilla get- ja set-metodeilla. Kontrolli voi toteuttaa ominaisuuksien asettamista varten graafisen ominaisuussivun (Property Page), jota tässä työssä toteutetussa komponentissa ei kuitenkaan käytetty. Metodit ovat kontrolliin toteutettuja jäsenfunktioita, joilla voi olla parametreja ja jotka voivat suorituksesta palatessaan palauttaa arvon asiakasohjelmalle. Kontrolli voi viestirajapintansa kautta laukaista tapahtumia, jotka vastaanotetaan ja käsitellään asiakasohjelmassa samaan tapaan kuin tavanomaisten Windows-kontrollien tapahtumat. [8, s. 433.]

Kontrollin koodi käännetään ocx-päätteiseen DLL-tiedostoon. Kontrolleille luodaan sen maailmanlaajuisesti yksilöivä ID-tunnus. Ennen käyttöönottoa kontrolli tulee rekisteröidä Windowsin järjestelmärekisteriin käyttäen Windowsin apuohjelmaa RegSvr32.exe.

Rekisteriin kirjoitetaan muun muassa kontrollin tunniste ja suoritettavien tiedostojen sijainti, joiden avulla sovellukset niitä ajettaessa ja kehitettäessä kontrollit löytävät. [8, s. 591-592.]

4.3 LabView-ohjelmointiympäristö

LabView (Laboratory Virtual Instrumentation Engineering Workbench) on National Instruments -yhtiön kehittämä graafinen ohjelmointiympäristö, joka perustuu yhtiön kehittämään G-kieleen. Se on suunniteltu pääasiassa mittaus-, testaus- ja automaatiosovellusten kehittämistä varten. LabView'ssä ei tuoteta ohjelmakoodia lainkaan kirjoittamalla, kuten perinteisillä ohjelmointikielillä, vaan ohjelmat rakennetaan graafisten objektien avulla, joiden välille luodaan yhteyksiä erilaisilla objektien välisillä johtimilla. LabView sisältää runsaasti valmiita komponentteja erilaisten ulkoisten laitteiden käyttämiseksi sekä käyttöliittymien rakentamiseksi. LabView-ohjelmia voidaan laajentaa käyttämällä muun muassa C++:lla kirjoitettuja ohjelmistokomponentteja, kuten .NET-komponentteja, DLL-kirjastoja ja ActiveX-kontrolleja. LabView-ohjelmien suorituskyky on jonkin verran C++-ohjelmia heikompi. [11.]

5 Kaappauskomponenttiprojektin toteutus

Tässä luvussa käsitellään JAI Oy:lle toteutetun testausjärjestelmän kuvankaappausrajapinnan suunnittelu ja toteutus. Toteutus tuli olemaan erillinen ohjelmistokomponentti, joka integroitiin varsinaiseen järjestelmään. Komponentin toteutusta voitiin pitää erillisenä osaprojektina.

Projektin toteutuksessa ei pitäydytty tiukasti missään tietyssä vaihejako- tai ohjelmistonkehitysmallissa, vaan eri malleja sovellettiin tilanteen mukaan. Pääasiassa projektin toteutus eteni siten, että aluksi komponentista toteutettiin erittäin yksinkertainen prototyyppi, jolla pyrittiin varmistamaan erityisesti datan siirtämisen toimivuus komponentin ja asiakassovelluksen välillä. Kun prototyypin avulla voitiin todeta, että datan siirtäminen toimii riittävän tehokkaasti, aloitettiin varsinaisen komponentin toteutus ketterästi vesiputousmallia mukaillen. Ensin määriteltiin komponentin tarkoitus, tietosisältö ja toiminnot, sitten suunniteltiin sen arkkitehtuuri, rajapinnat ja moduulit. Määritelmien valmistuttua ne katselmoitiin yrityksen sekä projektiin kuuluvan insinööritoimiston kans-

sa. Näitä määritelmiä luonnollisesti jouduttiin tarkentamaan projektin edetessä. Kun komponentti oli pääpiirteittäin suunniteltu, aloitettiin sen ensimmäisen version ohjelmoiminen.

5.1 Prototyypin toteutus

Ensimmäisen prototyypin ohjelmointia varten tuli päättää, millä ohjelmistotekniikalla lopullinen komponentti tultaisiin todennäköisimmin toteuttamaan. Reunaehtoina olivat LabView-ohjelmointiympäristön tuki, kuvankaappauskorttien ohjelmistorajapinnan käytön mahdollisuus sekä mahdollisimman hyvä tehokkuus. Lisäksi koettiin hyödylliseksi myös tapahtumapohjaisen viestinvälitysmekanismin mahdollisuus komponentilta asiakasohjelmalle päin, sillä komponentin toiminta on luonteeltaan asynkronista ja sen tulisi informoida asiakasohjelmaa muun muassa valmistuneista kuvista. Käyttöjärjestelmäalustana tulisi olemaan Microsoft Windows 7.

Kokemuksen perusteella tiedettiin, että kuvankaappauskorttien ohjelmointirajapinnat eli API:t on useimmiten kirjoitettu ensisijaisesti C++-kielellä Microsoft Visual C++-ympäristössä. Näin on myös erityisesti tätä projektia varten valitun BitFlow'n kaappauskortin API. API saattaa olla tarjolla myös muilla ohjelmistotekniikoilla toteutettuna, kuten BitFlow'n API oli myös LabView'llä, mutta koska se osoittautui puutteelliseksi, haluttiin nimenomaan käyttää C++-toteutusta. Tällöin komponentti olisi varmemmin hyödynnettävissä myös muissa kuin LabView-järjestelmissä.

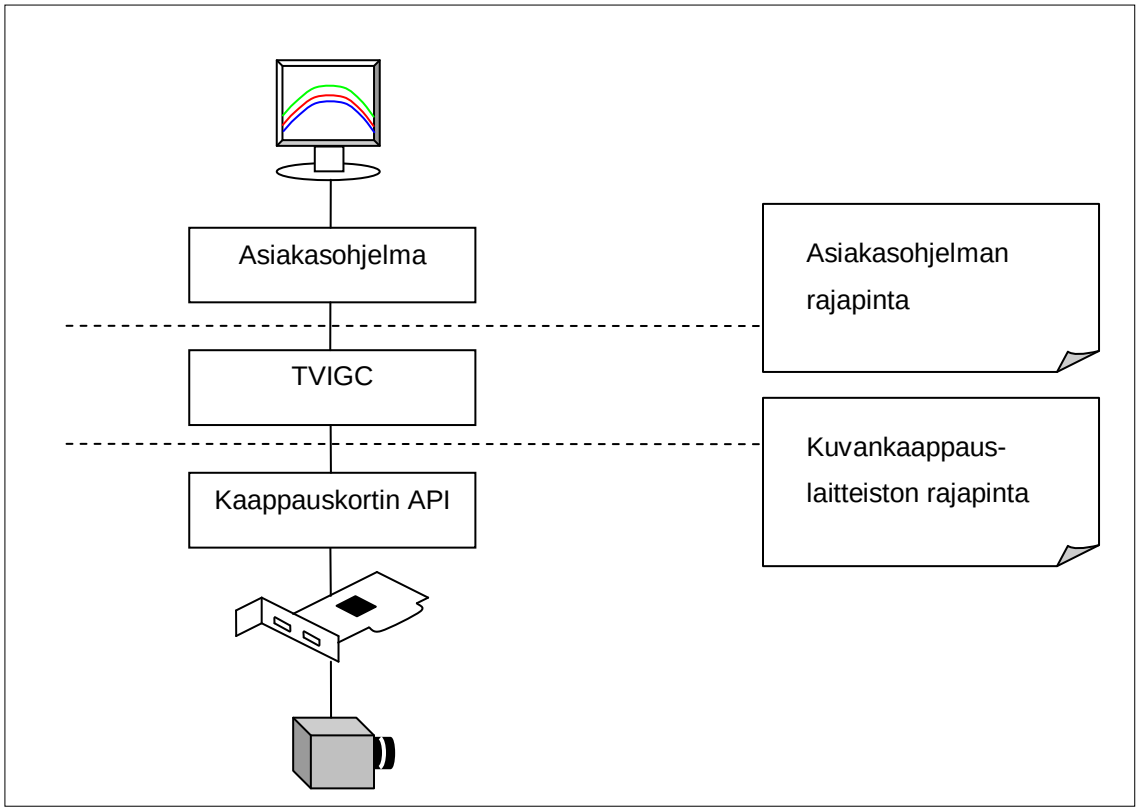
Potentiaalisten komponenttitekniikoiden tutkimisen jälkeen päädyttiin kokeilemaan Microsoftin ActiveX-komponentteja, jotka vaikuttivat täyttävän edellä mainitut reunaehdot. Lisäksi ActiveX-komponentteja voidaan kirjoittaa Microsoftin MFC-sovelluskehystä käyttäen, joka oli yrityksessä jo ennestään tuttu ja siten nopeutti alkuun pääsyä. Prototyypin kehitys alkoi kuitenkin tutustumisella ohjelmistotekniikoihin tai niiden osa-alueisiin, jotka eivät olleet tuttuja, kuten ennen kaikkea ActiveX-komponenttiteknologiaan. Nämä tekniikat on esitelty tarkemmin luvussa 4.

5.2 Kaappauskomponentin määrittely

Komponentin määrittelyvaiheessa määriteltiin toiminnot pseudotasolla, jonka tuloksena syntyi toiminnallinen määrittely.

5.2.1 Komponentin tarkoitus

Suunnitellun ja toteutetun ohjelmistokomponentin tarkoituksena on ohjata PC-tietokoneeseen liitettävää kuvankaappauskorttia tai -laitteistoa ja vastaanottaa siltä kuvadataa sekä käsitellä dataa tai suorittaa datalle laskentaa halutusti ja mahdollisimman tehokkaasti. Laskennat pyritään suorittamaan rinnakkaislaskentaa hyödyntäen tehokkuuden maksimoimiseksi.



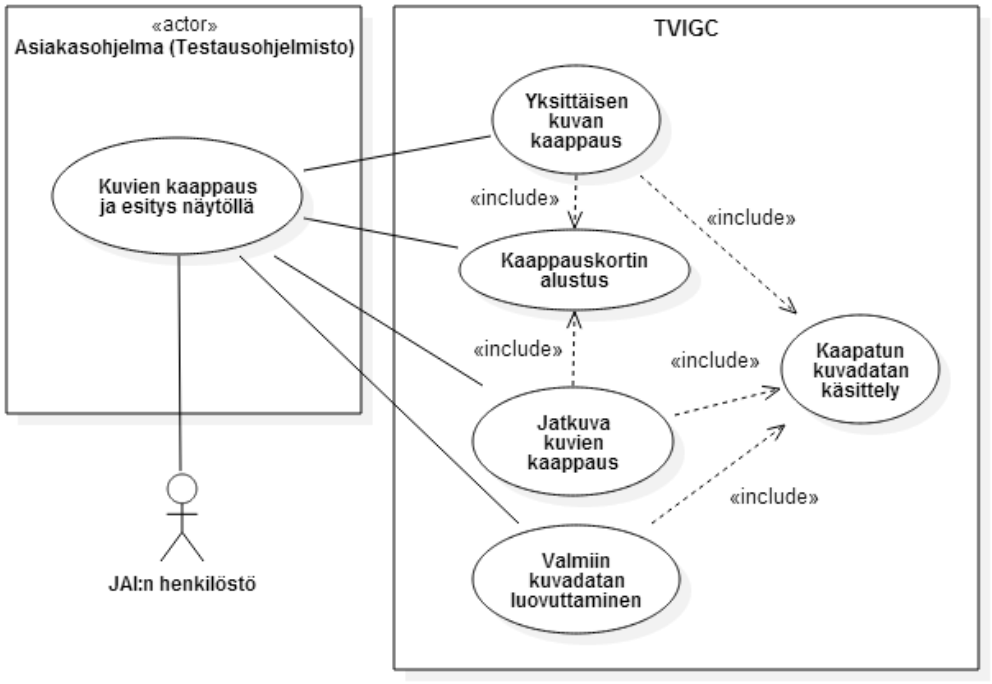
Kuva 4. TVIGC-komponentin liittyminen ympäristöönsä.

Komponentin ei ole tarkoitus toimia itsenäisesti, vaan se vaatii aina varsinaisen asiakasohjelman, joka toimii käyttöliittymänä ja jonka kautta komponenttia käytetään. Komponentin tuottama data on asiakasohjelman käytössä jatkokäsittelyä, esimerkiksi näytölle piirtämistä varten. Komponentti toimii ikään kuin kuvankaappauskortin ja varsinaisen kuvankaappausohjelmiston välissä (kuva 4). Komponentti oli tarkoitus toteuttaa modulaarisesti siten, että se on helposti laajennettavissa. Komponentille annettiin nimeksi *TVIGC* (TVI Grabbing Component). Tässä dokumentissa käytetään kuitenkin jatkossakin pelkästään nimitystä *komponentti*, koska ilmeistä sekaannuksen vaaraa ei ole.

Komponentti on tarkoitettu käytettäväksi erilaisissa kuvankaappausjärjestelmissä. Pääpaino oli JAI Oy:lle suunnitellussa tuotantokameroiden testausjärjestelmässä, joka esiteltiin luvussa 2. Samaa testausympäristöä voidaan käyttää myös muihin käyttötarkoituksiin, kuten esimerkiksi valolähteiden ja optiikoiden tutkimiseen, viallisten kameroiden vianhakuun, kameroiden optisen kokoonpanon apuvälineenä tai kameroiden esittelyyn messuilla.

5.2.2 Komponentin toiminta

Komponentille annetaan tiedot kuvan koosta, bittisyvyydestä, puskureiden määrästä jne., joiden perusteella se varaa muistia ja alustaa kuvankaappauskortin kaappausvalmiiksi. Lisäksi voidaan valita, minkälaista käsittelyä komponentin halutaan datalle tekevän. Komponentti kaappaa kuvia sarjassa, suorittaa jokaisen kaapatun kuvan jälkeen kuvadatalle valitut laskennat ja käsittelyt sekä tallentaa muokatut kuvadatat asiakasohjelman käytettäväksi. Asiakasohjelma suorittaa omat käsittelynsä kuvadatoille niin ikään jokaisen kaappauksen välissä. Virhetilanteen sattuessa virheestä pyritään toipumaan mahdollisuuksien mukaan sekä annetaan ilmoitus käyttäjälle suoraan tai asiakasohjelman kautta. Kuvassa 5 on esitetty komponentin tärkeimmät käyttötapaukset.



Kuva 5. Komponentin tärkeimpiä käyttötapauksia käyttötapauskaaviossa.

Järjestelmää voidaan pitää reaaliaikajärjestelmänä, sillä sen odotetaan suoriutuvan kameralta ja kaappauskortilta tulevien kuvien käsittelystä reaaliajassa ja sen toimintaan vaikuttaa asynkronisia tapahtumia kuten uusien kuvien valmistuminen. Komponentin sisäinen toiminta on myös asynkronista rinnakkaisuuden vuoksi. Järjestelmälle ei kuitenkaan asetettu tiukkoja vasteaikavaatimuksia, joten se voidaan luokitella pehmeäksi reaaliaikajärjestelmäksi. Näin ollen komponentista pyrittiin vain tekemään mahdollisimman tehokas.

5.2.3 Kuvankaappauksen tahdistus

Kuvankaappaus alkaa kaappauskortin lähettäessä kameralle signaalin, jolloin kamera alkaa valottaa viivasensoria ja lopuksi lähettää datan kaappauskortille. Tätä toistetaan, kunnes määritetyn kuvan pituuden verran viivoja on kaapattu. Tällöin yksi, kokonainen kuva (engl. frame) on valmis. Kuvien kaappaus toimii pääasiassa kahdessa erilaisessa toimintatilassa: jatkuvan kaappauksen tilassa (free-run) tai liipaisusignaalitilassa (trigger).

Kaappauksen ollessa jatkuvan kaappauksen toimintatilassa kuvia kaapataan asetetun viivataajuuden tahdissa tauotta. Yleensä kaappauskortti generoi kaappaussignaalin,

mutta myös kamera voi generoida signaalin, jolloin kortti vain vastaanottaa dataa kameran lähettämässä tahdissa.

Liipaisutilassa kaappauksen aloittava signaali tulee yleensä ulkoisesta signaalilähteestä, joka on liitetty kuvankaappauskorttiin.

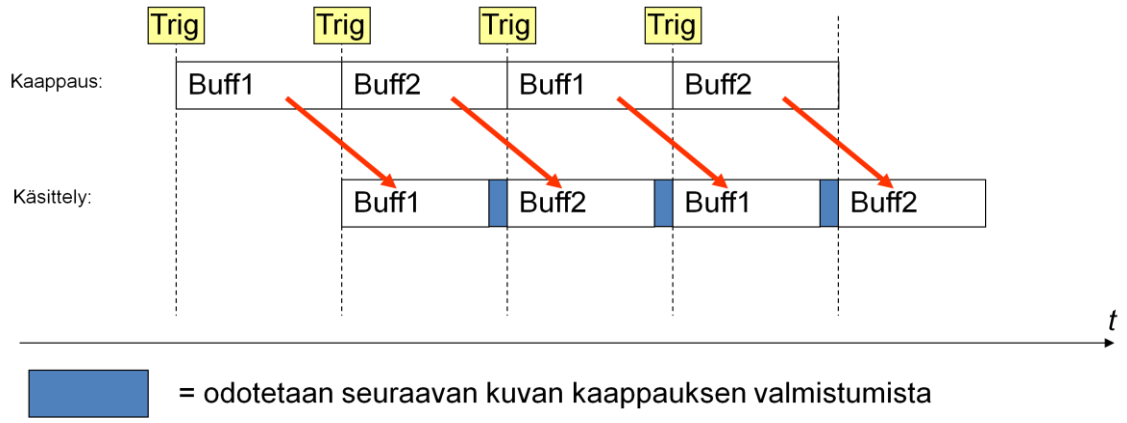
Komponentin toiminnassa oleellista on, että kuvien kaappaus on hallittua, eli kuvia kaapataan vain siinä tahdissa, kuin se käsittelyn puitteissa on mahdollista. Muuten on vaarana, että uutta kuvaa aletaan kaapata edellisen, vasta käsiteltävänä olevan tai käsiteltäväksi tulevan kuvan päälle, jolloin kuvat korruptoituvat. Tärkeätä on huolehtia myös, että kuvat tulevat käsiteltäviksi oikeassa kronologisessa järjestyksessä.

Jatkuvan- ja liipaistun kaappauksen tiloissa kaappauksen hallinta on vaikeaa, jolloin ratkaisuna on käyttää ohjelmallista liipaisusignaalia (software trigger). Komponentti päättää viime kädessä, milloin uuden kuvan kaappaus aloitetaan riippumatta siitä, mistä ja milloin todellinen signaali on annettu. Kun komponentti vastaanottaa tiedon uuden kuvan kaappaamisen aloittavasta signaalista ja edellisen kuvan kaappaus- ja käsittelyprosessi on siinä vaiheessa, että uuden kuvan kaappaaminen on mahdollista, laukaisee komponentti ohjelmallisesti liipaisusignaalin. Kuvankaappauskortti on myös ohjelmallista liipaisua käytettäessä yleensä asetettava liipaisutoimintatilaan.

Kaappauksessa voidaan käyttää kaksoispuskurointia, jolloin ensimmäisen kuvan käsittelyn aikana voidaan kaapata seuraavaa kuvaa valmiiksi. Jos toisenkaan kuvan kaappauksen jälkeen ensimmäisen kuvan käsittely ei ole vielä valmis, katkaistaan kaappaus ja odotetaan, että käsittely valmistuu. Ensimmäisen kuvan käsittelyn valmistuttua otetaan toinen, viimeksi kaapattu kuva käsittelyyn ja aletaan kaapata taas uutta kuvaa. Periaatteessa kuvapuskureita voidaan määritellä enemmänkin, mutta jos kuvien käsittely kestää systemaattisesti kauemmin kuin kaappaus, tulee raja kuitenkin ennemmin tai myöhemmin vastaan, jolloin kaappaus joudutaan keskeyttämään siksi aikaa, kunnes puskureita vapautuu. Näin ollen on selkeämpää käyttää kaappauksessa vain kahta kuvapuskuria, jolloin myös reaaliaikaisuus pysyy parempana, eli käsiteltäviksi tulevat kuvat eivät ole kovin vanhoja. Useampia kuvapuskureita voidaan silti määritellä, mutta ne toimivat lähinnä aikaisempien kuvien muistivarastoina. Lisäksi ongelmaksi tulee helposti myös ns. puskuriviive etenkin kaappausta pysäytettäessä. Tällöin on vaarana, että esimerkiksi sovellus, joka näyttää kaapattuja kuvia näytöllä, jää pysäytyksen jälkeen näyttämään aiempaa kuvaa kuin viimeisin oikeasti kaapattu kuva. Tämä saattaa

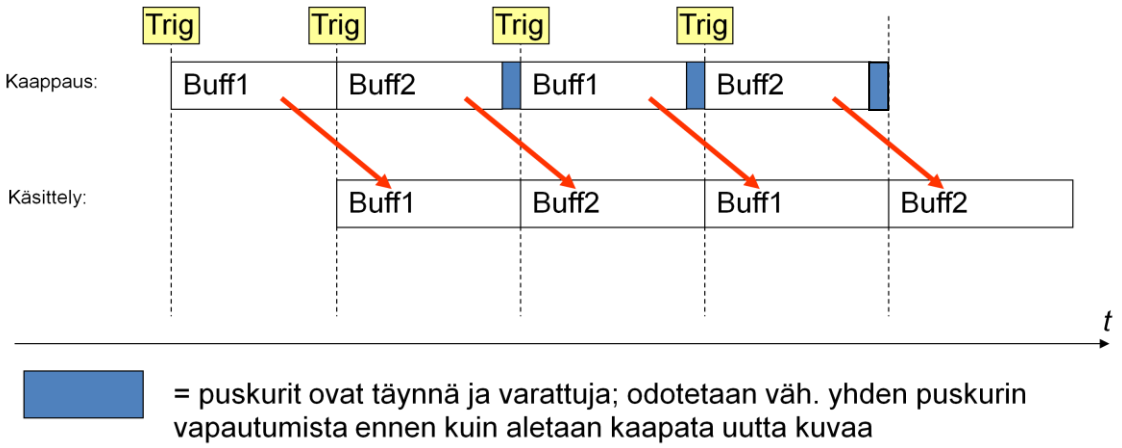
olla käyttäjän kannalta harhaanjohtavaa. Kaappauksen pysäyttämisen jälkeenkin käsittelysekvenssiä tulee suorittaa niin kauan, että kaikki puskureissa olevat kuvat on käsitelty.

Seuraavassa on esitetty kaksi erilaista tapausta kaappauksen ja käsittelyn tahdistuksesta ajoituskaavioiden muodossa. Kaavioissa kuvankaappaaminen kameralta kuvapuskuriin (Buff) alkaa aina välittömästi liipaisusignaalin (Trig) jälkeen. Kaavioiden Kaappaus-tapahtuma käsittää vain kuvan kaappaamiseen kuluvan ajan, jonka pituuteen vaikuttaa kuvan koko, vektorin pituus ja viivataajuus. Käsittely-tapahtumaan kuuluu kaikki kaappauksen jälkeen kuvalle suoritettavat operaatiot riippumatta siitä, mikä ohjelmisto tai komponentti käsittelyä suorittaa. Käsittelyn pituuteen vaikuttavat muun muassa kuvalle suoritettavan laskennan määrä, PC:n tehokkuus ja ohjelmiston tehokkuus.



Kuva 6. Tapaus 1: Kuvankaappaus kestää kauemmin kuin käsittely.

Kuvassa 6 on ideaalitilanne, jolloin kaappaus kestää käsittelyä kauemmin. Puskureita voidaan täyttää vuoronperään saumattomasti, ja puskurin käsittely alkaa välittömästi kaappauksen valmistuttua.



Kuva 7. Tapaus 2: Käsittely kestää kauemmin kuin kuvankaappaus.

Kuvassa 7 ongelmana on käsittelyn hitaus suhteessa kaappausnopeuteen. Koska molemmat puskurit on kaapattu täyteen ja vasta ensimmäistä puskuria käsitellään, joudutaan kaappauksen jatkamista odottamaan, kunnes ensimmäinen puskuri vapautuu. Tästä seuraa se, että kaappaus on katkonaista ja kuvia putoaa välistä pois.

5.2.4 Järjestelmän käyttäjät

Valmiin järjestelmän käyttäjiä ovat JAI:n testausjärjestelmän tapauksessa kameroiden tuotantohenkilöstö (kokoonpano, testaus, huolto), tuotekehittäjät ja tuote-esittelijät. Komponentin käyttäjiksi voidaan ajatella myös järjestelmäsuunnittelijat, jotka integroivat komponentin osaksi toista järjestelmää. Komponentin mukana toimitetaan ohjeet, joista selviää sen toiminta ja tarjoamat palvelut sekä integrointiohjeita.

5.2.5 Oletukset ja rajoitteet

Tässä luvussa käsitellään oletuksia ja rajoitteita, jotka vaikuttivat komponentin suunnitteluun. Erityisesti laitteistojen asettamat rajoitukset tuli huomioida.

Komponentti suunniteltiin ensisijaisesti JAI Oy:n valmistamien kameroiden (Pricolor-, Colibri-, Xiimus-, Priimus- ja Sweep-tuoteperheet) kanssa käytettäväksi, jolloin se tukee muun muassa pikselimääriä ja bittisyvyyksiä lähtökohtaisesti vain näiden kameramallien mukaisesti. Vaikka osa malleista on jo poistunut tuotannosta, on tuelle tarvetta esimerkiksi huoltotapauksia varten. Tukia uusille kameramalleille voidaan lisätä myöhemmin.

Kuvankaappauskortit ja niiden ajurit tai ohjelmointirajapinnat saattavat aiheuttaa rajoitteita kuvan koon, kuvapuskureiden määrän, liipaisuasetusten ja muiden vastaavien ominaisuuksien suhteen. Siksi komponentti suunniteltiinkin mahdollisimman joustavaksi siten, että eri korttien erilaiset ominaisuudet saadaan käyttöön. Korttien asetusten muuttamiseen ja talteenottoon käytetään usein jonkinlaista asetustiedostoa, jonka avulla kaappausparametrit välitetään kortille. Komponentissa pyrittiin tällaiset konfigurointitiedostot saamaan käyttäjiltä mahdollisimman näkymättömiin joko ohittamalla niiden käyttö kokonaan, mikäli kortin ohjelmointirajapinta mahdollistaa parametrien asettamisen funktiokutsuilla, tai integroimalla ne komponenttiin, josta ne voidaan sisäisesti lukea, kuten BitFlow'n kaappauskorttien tapauksessa. Tavoite oli, että komponentille voidaan alustuksen yhteydessä vain antaa halutut parametrit (kuvan koko jne.) ja komponentti huolehtii tietojen välityksestä kortille. Toisaalta haluttiin myös mahdollistaa ulkoisten asetustiedostojen käyttö tarvittaessa.

Erityisesti pitkiä kuvia erittäin nopealla viivataajuudella kaapattaessa kuvankäsittely saattaa kestää systemaattisesti pidempään kuin kuvankaappaus. Tällöin kuvat, joita ei ehditä käsitellä, menetetään.

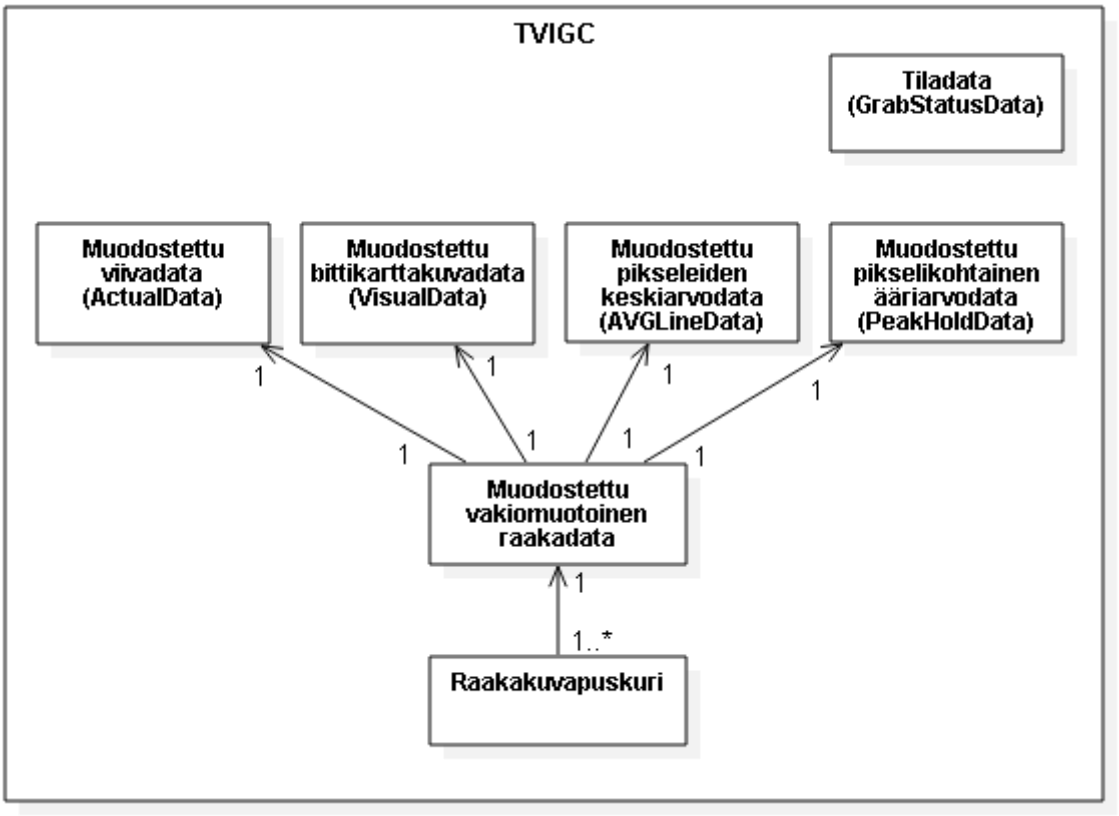
Komponentin pitäisi pystyä kaappaamaan vähintään 4096x10000-kokoisia kuvia vähintään kahteen puskuriin.

Komponenttia käytetään testausohjelmiston kanssa yrityksessä päivittäin yleensä usean tunnin ajan useilla työasemilla, joilla ei kuitenkaan ole riippuvuuksia keskenään. Komponentin on pystyttävä pitämään kaappaus käynnissä tarvittaessa useamman päivän yhteen menoon. Riittää, että se pystyy toimimaan kerrallaan yhdessä työasemassa vain yhden asiakasohjelman ja kuvankaappauskortin kanssa.

5.2.6 Komponentin tietosisältö

Komponentin tärkein käsiteltävä tieto on kameralta kaapattava kuvadata, joka tallennetaan pääasiassa tietokoneen keskusmuistiin. Riippuen kuvankaappauskortin ominaisuuksista kuvia voidaan pitää muistissa samanaikaisesti joko yksi tai useampia, mikäli käytetään puskurointia. Kaappauskortit hoitavat kuvien tallentamisen ja puskuroinnin keskusmuistiin, josta kuvia voidaan ottaa komponenttiin jatkokäsiteltäviksi osoittimien avulla.

Komponentti hakee kaapatun raakakuvadatan muistista ja laskee ja järjestelee dataa halutusti, jonka jälkeen se tallettaa tulokset omiin tietueisiinsa. Komponentti pitää yllä myös tilataulukkoa, josta on luettavissa komponentissa käytössä olevat aktiiviset parametrit kuten kuvan koko ja kaappauksen tila. Kuva 8 havainnollistaa komponentin käsittelemiä tietoja ja niiden välisiä yhteyksiä.



Kuva 8. TVIGC:n tietosisältö korkealla tasolla esitettynä.

Kaikki muodostetut kuvadatat johdetaan luonnollisesti kuvapuskureihin tallennetusta raakakuvadatasta. Puskureita voi olla useita, mutta muita tietueita vain yksi kappale kutakin lajia. Komponentti varaa muistin kaikkia kuvadatoja sekä kuvapuskureita varten, ja muisti kuuluu komponentin muistiavaruuteen. Seuraavassa on esitelty kaikki komponentin varaamat muistialueet ja -tietueet. Datan järjestys muistissa on esitetty tietohakemistomäärittelyä käyttäen. Tässä käytetyn tietohakemistonotaation merkintätavat on esitetty taulukossa 1.

Taulukko 1. Tietohakemistonotaation merkintätavat, joita on käytetty tietueiden datan järjestyksen kuvaamiseen. [12, s. 111.]

Merkki	**Merkitys**
+	ja
()	optionaalinen (voi puuttua)
{}	toisto (0...N kertaa)
n{}m	toisto (n...m kertaa)
[]	vaihtoehtoja
|	vaihtoehtojen erotin

Puskuriin kaapattu raakadata

Puskureita varten keskusmuistista varataan valmiiksi tilaa. Tilantarve riippuu käytetystä kamerasta ja asetuksista. Muistialkion koko voi vaihdella 8…64 bitin välillä. Esimerkiksi käytettäessä kolmekanavaista 2K-kameraa 12-bit tilassa ja kaapattaessa 1000 viivaa pitkää kuvaa puskurin muistialkion koon ollessa 16 bittiä, on yhden puskurin tilantarve muistista noin: (3 · 2048 · 1000 · 16 b) ≈ 94 Mb ≈ 12 MB. Datan muoto ja järjestys puskurissa riippuu käytettävästä kuvankaappauskortista. Useimmiten data on muodossa RGBRGB..., tietohakemistomäärittelyllä esitettynä:

1{ 1{R+(G+B)} [512|1024|2048|4096] }N
R/G/B = [8|10|12]-bit luku

Vakiomuotoon muunnettu raakadata

Puskuriin kaapattu raakadata muunnetaan ensin vakiomuotoon, joka tarjoillaan muut kuvadatat muodostavien ja laskevien toimintojen käyttöön. Tilantarve on sama kuin puskurin tilantarve, paitsi että muistialkion koko on aina 32 bittiä. Kerrallaan vain yhden puskurin data muunnetaan. Data on muodossa RRR…GGG…BBB..., tietohakemistomäärittelyllä esitettynä:

1{ 1{R}Line +(1{G}Line + 1{B}Line) }N
R/G/B = [8|10|12]-bittinen luku
Line = [512|1024|2048|4096]

Viivadata

Viivadata muodostetaan vakiomuotoon muunnetusta puskuriin kaapatusta raakadatasta. Datan muoto on sama kuin vakiomuotoisella datalla. Viivadatan bittisyvyys on aina sama kuin kameralta ulostulevan datan bittisyvyys.

Viivadataa varten varataan keskusmuistista tilaa asetuksesta riippuen joko yhtä viivaa tai kuvan kaikkia viivoja varten. Muistialkion koko on 32 bittiä. Muistitilantarve esimerkiksi kolmekanavaisella 2K-kameralla on yhtä viivaa tallennettaessa $(3 \cdot 2048 \cdot 32\ b) = 192\ kb = 24\ kB$.

Bittikarttakuvadata

Bittikarttakuvadata on kaksiulotteinen taulukko, joka muodostetaan vakiomuotoon muunnetusta puskuriin kaapatusta raakadatasta. Kuvadatan bittisyvyys on aina 8 bittiä kunkin värikanavan osalta. Jokainen pikseli talletetaan taulukon 32-bittisiin muistialkioihin siten, että tavut alhaalta (LSB) ylöspäin sisältävät värikanavat seuraavasti: punainen, vihreä, sininen. Ylintä tavua (MSB) eli ns. alfa-kanavaa ei käytetä.

Kuvadataa varten varataan keskusmuistista tilaa. Muistitilantarve esimerkiksi kolmekanavaisella 2K, 12-bit kameralla 1000 viivan kuvaa tallennettaessa on: $(2048 \cdot 1000 \cdot 32\ b) \approx 63\ Mb \approx 8\ MB$.

Pikseleiden keskiarvodata

Pikseleiden keskiarvodata muodostetaan vakiomuotoon muunnetusta puskuriin kaapatusta raakadatasta. Pikseleiden keskiarvodataa varten varataan keskusmuistista tilaa. Muistialkion koko on 32 bittiä. Muistitilantarve riippuu laskentaan mukaan otettavien viivojen lukumäärästä. Datan bittisyvyys on aina sama kuin kameralta ulostulevan datan bittisyvyys. Data on muodossa RRR…GGG…BBB..., tietohakemistomäärittelyllä esitettynä:

1{R}Lines +(1{G}Lines + 1{B}Lines)
R/G/B = [8|10|12]-bittinen luku
Lines = 1…*

Pikselikohtainen ääriarvodata

Pikselikohtainen ääriarvodata muodostetaan vakiomuotoon muunnetusta puskuriin kaapatusta raakadatasta. Data on muodoltaan ja kooltaan täysin vastaava kuin viivadata, mutta kaksinkertainen (maksimi- ja minimiarvot).

Tiladata

Komponentin tiladata on yksiulotteinen 32-bittinen taulukko, jossa ylläpidetään järjestelmän parametreja. Taulukkoa varten varataan keskusmuistista tilaa. Taulukossa on 21 alkiota, yksi kutakin parametria kohti. Parametrit esitellään tarkemmin luvussa 5.2.7 *Kaappausparametrien ylläpito* -toiminnon yhteydessä.

5.2.7 Komponentin toiminnot

Komponentin toiminnot koostuvat pääasiassa kaappausparametrien asettelusta, kuvankaappauskortin ohjaamisesta ja kuvankäsittelyoperaatioista. Tässä luvussa on esitelty kaikki komponentin sisältämät toiminnot.

Kuvankaappauksen alustus

Ennen kuin kuvankaappausta voidaan käynnistää, on komponentille kerrottava, millä komponentin tukemalla kuvankaappauskortilla ja -parametreilla kuvaa halutaan kaapata. Alustusoperaatiossa varataan kuvankaappauskortti sovelluksen käyttöön sekä varataan tarvittava määrä muistia kaapattavia ja käsiteltäviä kuvia varten. Alustustoiminnolle annetaan syötteinä halutut kaappausparametrit kuten viivan pituus eli pikselien lukumäärä, viivojen lukumäärä eli kuvan pituus, bittisyvyys, kuvapuskureiden lukumäärä, värikanavien lukumäärä, datan ulostuloformaatti kamerasta ja käytettävä kuvankaappauskortti. Vaihtoehtoisesti voidaan antaa nimi ja polku asetustiedostoon, jossa edellä mainitut parametrit tai osa niistä on määritelty, mikäli kyseinen kuvankaappauskortti tällaista tiedostoa tukee. Syötettyjen parametrien kelpoisuus tarkistetaan. Toiminnossa aiheutuvia virhetilanteita voi olla se, että kuvankaappauskortti on jo varattuna jollain toisella sovelluksella, muistinvaraus epäonnistuu (yleensä muisti ei riitä) tai jokin kaappausparametreista on laiton.

Kuvankaappauksen resurssien vapautus

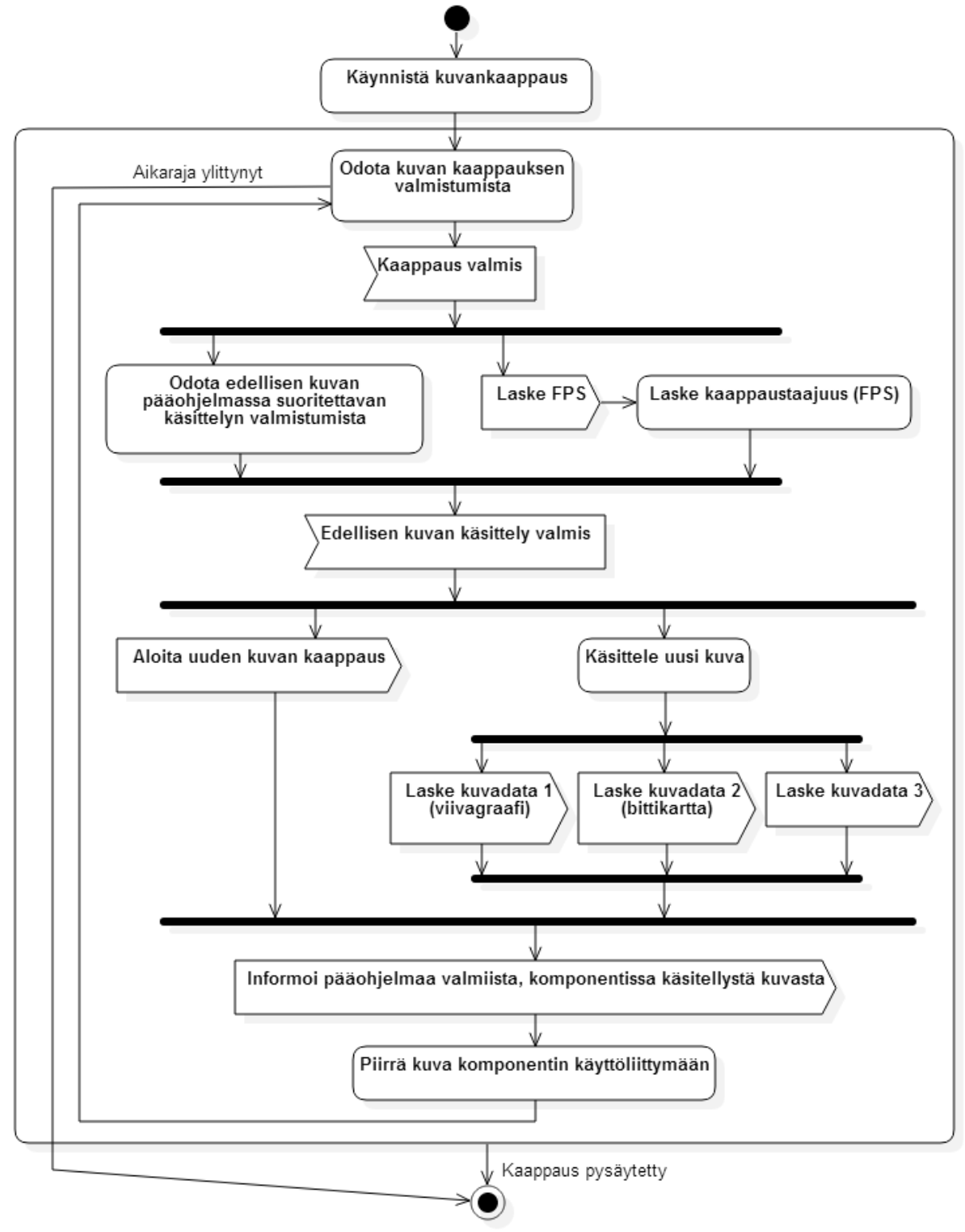
Ohjelman päättyessä tai haluttaessa muuttaa kaappausparametreja, vapautetaan varatut resurssit kuten kuvankaappauskortti ja muistivaraukset. Toiminnossa aiheutuu virhetilanne, jos vapautettavaa ei ole.

Jatkuvan kuvankaappauksen käynnistys ja pysäytys

Jatkuvan kuvankaappauksen käynnistäminen aloittaa sekvenssin, jossa kuvia kaapataan kameralta hallitusti luvussa 5.2.3 esitetyn mukaisesti ohjelmallista liipaisua käyttäen siihen asti, kunnes kaappaus pysäytetään tai se pysähtyy virhetilanteeseen. Kuvia kaapataan alustustoiminnossa varattuihin muistipuskureihin järjestyksessä, ja jokaiselle kuvalle suoritetaan kaappauksen valmistuttua käyttäjän valitsemat kuvankäsittelyoperaatiot. Muodostuneet käsitellyt kuvat talletetaan omiin muistitietueisiinsa, jotka on esitelty luvussa 5.2.6. Käsittelyiden valmistuttua asiakasohjelmaa informoidaan valmistuneesta kaapatusta ja käsitellystä kuvasta, jolloin asiakasohjelma voi hakea käsitellyt kuvat käyttöönsä. Pysäytettäessä kaappaus pysäytyshetkellä meneillään olevan kuvan kaappaus suoritetaan loppuun ja puskureissa oleville käsittelemättömille kuville suoritetaan määrätyt käsittelyoperaatiot, ja vasta tämän jälkeen kuvankaappaustila katsotaan pysähtyneeksi. Toiminnossa aiheutuu virhetilanne, jos kaappausta yritetään käynnistää kaappauksen jo ollessa käynnissä tai pysäyttää sen ollessa pysäytettynä. Virhetilanne syntyy myös, jos kuvankaappaus kestää odottelulle määritettyä aikaylärajaa (timeout) kauemmin. Tämän aikakatkaisun tapahtuessa kaappaus menee pysäytetty-tilaan ja kaappaus voidaan käynnistää uudelleen. Kaappaussekvenssi on esitetty kuvan 9 aktiviteettikaaviossa.

Yksittäisten kuvien kaappaus

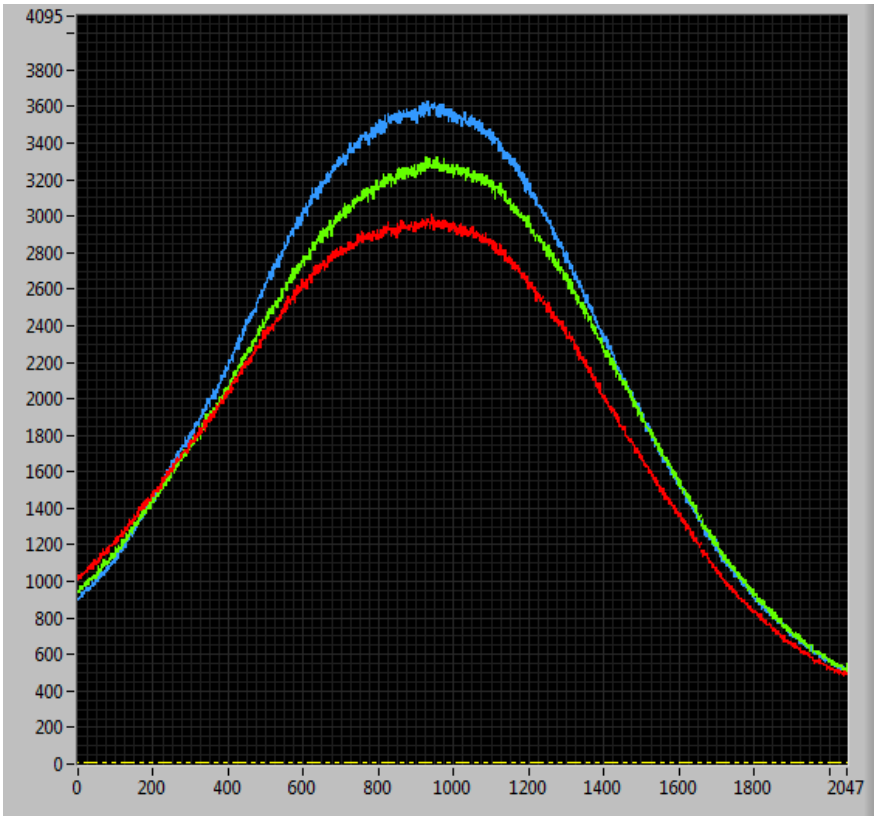
Haluttaessa voidaan kaapata vain yksittäisiä kuvia jatkuvan kaappauksen sijaan, jolloin kaappaus pysäytetään automaattisesti ensimmäisen kuvan kaappauksen ja käsittelyn valmistuttua. Muuten kaappaussekvenssi toimii täysin samoin kuin jatkuvan kaappauksen tilassakin. Tälläkin toiminnolla muistipuskurit täytetään järjestyksessä jatkaen seuraavasta vapaasta puskurista, johon edellinen kuva kaapattiin joko jatkuvaa tai yksittäisen kuvan kaappaustoimintoa käyttäen.



Kuva 9. Kuvankaappaussekvenssi kuvattuna aktiviteettikaaviolla.

Viivadatan laskenta

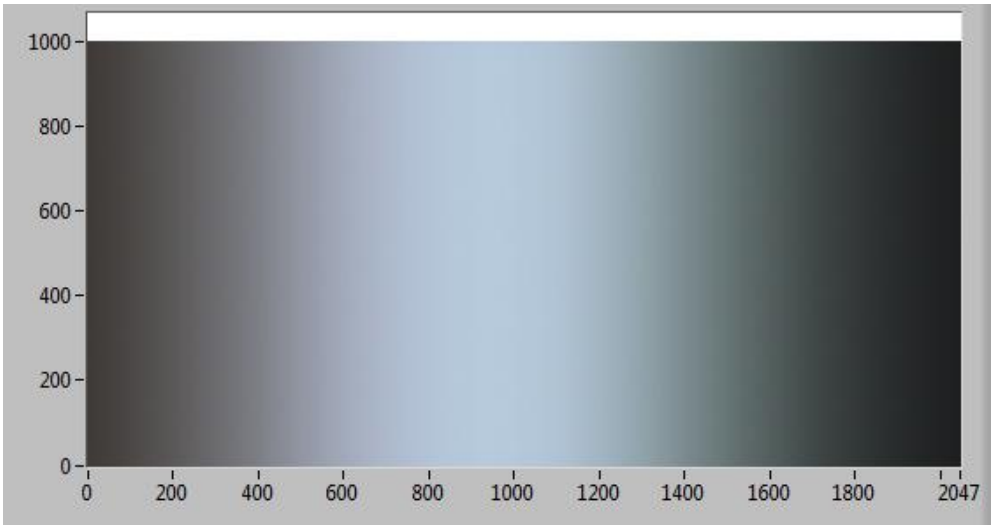
Viivakameran sensorin tuottamaa kuvadataa tutkitaan kameran tuotanto-, testaus- ja tuotekehitysvaiheissa yleensä graafisesti piirtämällä siitä kuvaaja kuten kuvassa 10. Kuvaajassa pikseleiden ja värikanavien väliset erot ovat selkeämmin havaittavissa kuin jos ne piirretään bittikarttagrafiikkana. Kuvaajan vaaka-akseli edustaa sensorin pikseleitä ja pystyakseli pikseleiden digitaalisia data-arvoja, eli pikseleiden keräämän valon intensiteettiä. Kaapatusta raakadatasta poimitaan haluttu yksittäinen vaakaviiva ja talletetaan viivadatan tietueeseen samalla bittisyvyydellä. Viivadatan tietue on kuvattu luvussa 5.2.6. Vaihtoehtoisesti kaapatun kuvan viivoista lasketaan kohinan poistamiseksi keskiarvo ja tulos talletetaan tähän tietueeseen. Todellista viivadataa ja viivojen keskiarvoa ei siten saada luettua samanaikaisesti. Tietue voidaan myös uudelleen allokoida koko kuvan kokoiseksi ja näin ollen tallettaa koko kuva alkuperäisellä bittisyvyydellä. Tällöin keskiarvon laskentaa ei voida käyttää. Edellä mainitut optiot voidaan parametrien avulla määrittää asiakasohjelmassa, myös käynnissä olevan kaappauksen aikana. Viivadata on muodostamisen jälkeen luettavissa asiakaskasohjelmassa.



Kuva 10. Kaapatusta kuvasta poimitun yhden viivan pikseleistä piirretty kuvaaja. Kameran sensorin pituus on 2048 pikseliä ja bittisyvyys 12 bittiä. Kameralla on kuvattu pistemäistä valonlähdettä, joka osoittaa sensorin keskikohtaan.

Bittikarttakuvadatan laskenta

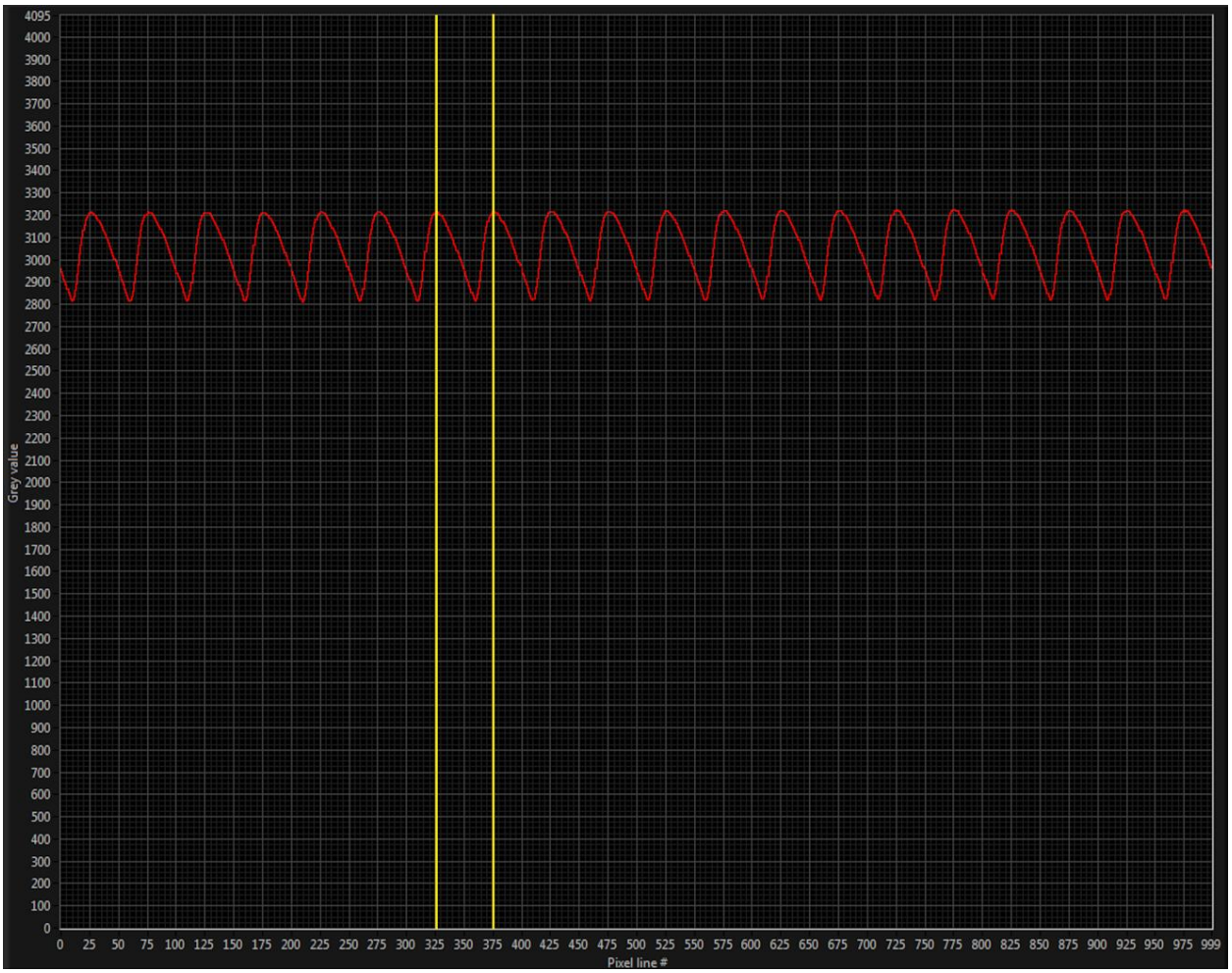
Viivakameralla valotetuista peräkkäisistä viivoista muodostetaan tavanomaisia kaksiulotteisia kuvia useimmissa käytännön sovelluksissa. Siksi kameran kuvanlaatua on tarpeen tarkastella myös esimerkiksi jonkinlaisesta testikuvasta kuvatusta ja muodostetusta bittikarttakuvasta. Siitä voidaan helpoimmin arvioida käytännön kuvanlaatua kuten terävyyttä tai värien toistoa koko kaapatulla kuva-alueella. Lisäksi kameroita esiteltäessä esimerkiksi messuilla on mielekästä näyttää kokonaista havainnollista kuvaa. Bittikarttakuvadataa on havainnollistettu kuvassa 11. Kaapattu raakadata muunnetaan 8-bittiseksi näytölle piirtämistä varten. Kuvadata on muodostamisen jälkeen luettavissa asiakasohjelmassa. Bittikarttakuvadatan tietue on kuvattu luvussa 5.2.6.



Kuva 11. Tuhannesta kaapatusta viivasta muodostettu bittikarttakuva. Sekä kamera että pistemäinen valonlähde ovat olleet kuvattaessa paikallaan, jolloin viivat toistuvat samanlaisina koko kuvan pituudelta. Kuvan ensimmäinen (alin) viiva on esitetty kuvaajamuodossa kuvassa 10.

Pikseleiden keskiarvodatan laskenta

Kameran valottamien peräkkäisten viivojen välistä vaihtelua eli hajontaa voidaan tutkia laskemalla kuvasta kunkin yksittäisen viivan pikseleiden intensiteettien keskiarvo. Tulos voidaan esimerkiksi esittää graafisesti kuvaajassa, jossa vaaka-akseli edustaa viivoja ja pystyakseli viivan pikseleistä laskettua keskiarvoa. Esimerkki kuvaajasta on kuvassa 12. Keskiarvodata muodostetaan kaapatusta raakadatasta, ja se on muodostamisen jälkeen luettavissa asiakasohjelmassa. Datan muoto on kuvattu luvussa 5.2.6.

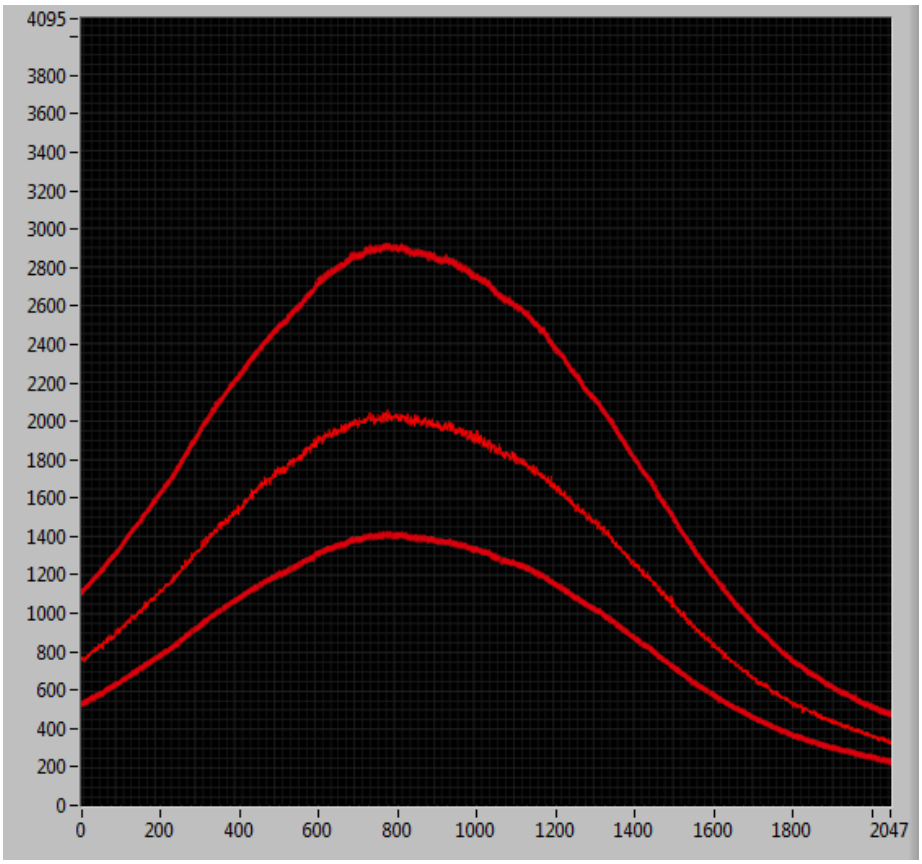


Kuva 12. Kuvan esimerkissä nähdään 50 viivan jaksoissa toistuva siniaalto. Kun tunnetaan kameran viivataajuus (esimerkin tapauksessa 5000 viivaa/s), saadaan siniaallon jaksonajaksi 0,01 s, jonka käänteislukuna saadaan siniaallon taajuudeksi 100 Hz. Tästä voidaan päätellä, että kuvan otossa on käytetty sähköverkkoon kytkettyä vaihtovirtavalaisinta. Kuvassa näkyvät keltaiset pystyviivat ovat apuviivaimia. Kuvassa on näkyvissä vain punaisen värikanavan data.

Pikselikohtainen ääriarvojen seuranta

Erilaisten satunnaisten virhetilanteiden kuten pätkivien sähkövikojen etsimistä ja toteamista helpottamaan kaapattujen viivojen pikseleiden ääriarvoja voidaan rekisteröidä. Jokaisen uuden kaapatun viivan pikseleiden arvoja verrataan tallennettuihin maksimi- ja minimiarvoihin. Uusi arvo tallennetaan pikselikohtaisesti maksimi- tai minimitietueeseen, mikäli arvo ylittää tai alittaa vanhan arvon. Rekisteröintiä jatketaan niin kauan, kunnes tietue nollataan, jonka jälkeen rekisteröinti aloitetaan alusta seuraavasta viivasta. Toiminnossa on kaksi toimintatilaa liittyen tietueen nollaukseen. Toisessa niistä nollaus suoritetaan vain käyttäjän toimesta. Toisessa nollaus suoritetaan automaattisesti jokaisen viivan jälkeen lukuun ottamatta heti nollauksen jälkeistä ensimmäistä

viivaa, jolloin nähdään muutos vain edelliseen viivaan. Muodostuneista pikseleiden ääriarvoista nähdään kunkin pikselin vaihteluväli, ja niistä voidaan asiakasohjelmassa esimerkiksi piirtää verhokäyrät viivadatan yhteyteen, kuten kuvassa 13. Datan muoto on kuvattu luvussa 5.2.6.



Kuva 13. Viivagraafiin on piirretty viivadatan (keskimmäinen käyrä) lisäksi pikseleiden hajontaa ilmaisevat maksimi- ja minimiverhokäyrät. Kuvasta on poistettu näkyvistä muut värikanavat kuin punainen.

Käsiteltävien värikanavien valinta

Komponentin suorittamat kuvankäsittelytoiminnot voidaan kohdistaa vain halutuille värikanaville. Värikanavien valinta voidaan tehdä kullekin kuvadatan muodostustoiminnolle erikseen tai kerralla kaikille yhteisesti. Niiden värikanavien data, joita ei käsitellä, tallennetaan kuvadatan tietueeseen nollaksi. Valinta voidaan tehdä kaappauksen aikana, mutta myös takautuvasti jo kaapatuille, raakakuvapuskureissa oleville kuville. Tällöin laskennat suoritetaan uudestaan ja valmis data tarjotaan asiakasohjelman käytettäväksi.

Kuvankäsittelyoperaatioiden valinta

Komponentin raakakuville suorittamat kuvankäsittelyoperaatiot voidaan yksittäin kytkeä päälle ja pois, myös kaappauksen aikana. Tavoitteena on lisätä tehokkuutta muodostamalla vain ne kuvadatat, joille on käyttöä. Muisti varataan kaikille kuvadatoille joka tapauksessa. Haluttujen operaatioiden valinta tehdään asiakasohjelmassa.

Kuvapuskureiden selaus

Kun kuvankaappaus on pysäytetty, voidaan raakakuvapuskureihin kaapattuja kuvia noutaa tarkasteltavaksi puskurin indeksillä. Noudetulle raakakuvadatalle suoritetaan kaikki määrätyt kuvankäsittelyoperaatiot kuten kaappauksenkin aikana. Operaatioiden parametreja voidaan myös muuttaa ennen puskurin noutoa, ja noudon yhteydessä laskennat suoritetaan uudelleen vastaavasti. Samalla tapaa myös kuvan yksittäisiä viivoja voidaan selata halutusta puskurista.

Kuvankaappaustaajuuden laskenta

Jatkuvan kuvankaappauksen tilassa lasketaan reaaliajassa käynnissä olevan kaappauksen nopeus eli kaapattujen ja käsiteltyjen kuvien määrä sekunnissa. Tulos ilmoitetaan yksikössä FPS (frames per second, kuvia sekunnissa). Tällä arvolla voidaan seurata järjestelmän suorituskykyä suhteessa kameran viivataajuuteen. Matalampi arvo tarkoittaa, että kuvien kaappauksen väleissä on taukoja. Nopeuteen vaikuttavia tekijöitä ovat kuvan koko pituussuunnassa eli viivojen lukumäärä, asetettu viivataajuus ja PC:n suorituskyky.

Kaappausparametrien ylläpito

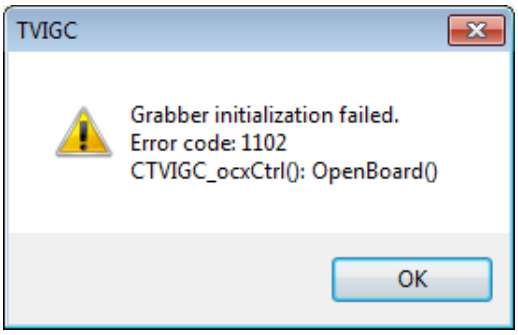
Komponentti ylläpitää reaaliajassa taulukkoa, josta on luettavissa komponentin tilaan liittyviä tietoja sekä parametrit, joilla se on alustettu. Taulukon sisältämät parametrit on esitetty taulukossa 2. Taulukko talletetaan tiladatatietueeseen, joka esiteltiin luvussa 5.2.6.

Taulukko 2. Komponentin ylläpitämät kaappausparametrit.

Indeksi	Parametri
0	Kaappaus käynnissä/pysäytettynä
1	Viivojen lukumäärä
2	Viivan pituus (pikseleiden lukumäärä)
3	Bittisyvyys
4	Viivan liipaisun toimintatila
5	Viivan liipaisusignaalin aktiivinen reuna
6	Viivan liipaisusignaalin tyyppi
7	Viivan liipaisusignaalin lähde
8	Kuvan liipaisun toimintatila
9	Kuvan liipaisusignaalin aktiivinen reuna
10	Kuvan liipaisusignaalin tyyppi
11	Kuvan liipaisusignaalin lähde
12	Kuvankaappaustaajuus (FPS)
13	TVIGC:n versio
14	Aikakatkaisun yläraja millisekunteina (timeout)
15	Raakakuvapuskureiden lukumäärä
16	Datan ulostulotapa
17	Kaappaus alustettu/alustamatta
18	Värikanavien lukumäärä
19	Rinnakkain ulostulevien pikseleiden lukumäärä
20	Valittu kuvankaappauskortti

Virheilmoitusten esitys

Komponentissa tapahtuville mahdollisille virhetilanteille määritettiin virhekoodit ja niitä vastaavat virheilmoitustekstit. Virheilmoitusten esitystapa on valittavissa asiakasohjelmassa. Virhetilanteen synnyttyä voidaan generoida virheilmoitusikkuna kuten kuvassa 14, laukaista komponentin tapahtumarajapinnan kautta tapahtuma, joka välittää virhekoodin tai olla ilmoittamatta virheestä lainkaan.



Kuva 14. Esimerkki komponentin tuottamasta virheilmoitusdialogista.

Komponentin käyttöliittymä

Komponentti tarjoaa yksinkertaisen grafiikan piirtotoiminnon ActiveX-kontrollin käyttöliittymän kautta, joka voidaan upottaa asiakasohjelmaan. Käyttöliittymään voidaan piirtää jokin muodostetuista datoista, kuten viivagraafi tai bittikarttakuva. Grafiikan piirron lisäksi on mahdollista yksittäisten pikseliarvojen poiminta kuvasta ja kuvan vieritys ja zoomaus. Käyttöliittymästä on nähtävissä mallikuva luvussa 5.5 kuvassa 18.

5.3 Tekninen suunnittelu

Komponentin suunnitteluvaiheessa määriteltiin ja suunniteltiin sen tekninen toteutus luvussa 5.2 esitetyn toiminnallisen määrittelyn mukaisesti. Ensin suunniteltiin komponentin sisäinen rakenne eli arkkitehtuuri, missä ohjelman toiminnot jaettiin moduuleihin ja moduulien väliset rajapinnat määriteltiin. Komponentissa suoritetaan toimintoja rinnakkain säikeissä, joten säikeiden välinen kommunikointi suunniteltiin. Arkkitehtuurin suunnittelun jälkeen suunniteltiin kunkin moduulin sisäinen rakenne. Lisäksi määriteltiin julkinen rajapinta, jonka kautta asiakasohjelma komponenttia käyttää. Suunnitteluvaiheen lopputuloksena syntyi tekninen määrittely.

Kuten jo prototyypin toteutusvaiheessa oli päätetty, komponentti toteutettiin käyttäen Microsoftin ActiveX-tekniikkaa ja MFC-sovelluskehystä, sekä toteutuskielenä C++-kieltä. Suunnittelun lähtökohtana oli olioperustainen toteutus. ActiveX-komponenttitekniikka asetti suunnittelun osalta jonkin verran reunaehtoja muun muassa julkisen rajapinnan määrittelyn suhteen, sillä komponentin rajapintaan voidaan määritellä metodien lisäksi myös ominaisuuksia ja tapahtumia. Lisäksi komponentille toteutettiin graafinen käyttöliittymä, jonka hyödyntäminen asiakasohjelmassa ei kuitenkaan ole vaatimus,

vaan komponentin tarjoamia palveluja voidaan käyttää asiakasohjelmassa hyvin myös ilman käyttöliittymää, mikäli sille ei ole tarvetta.

Arkkitehtuurisuunnittelussa pyrittiin selkeään modulaariseen rakenteeseen komponentin toiminnan ymmärrettävyyden ja myöhemmän helpon laajentamisen vuoksi. Pyrkimyksenä oli tehdä moduuleista eli käytännössä luokista mahdollisimman toisistaan riippumattomia, jotta mahdolliset myöhemmät lisäykset ja muutokset olisi helpompi toteuttaa. Koska jo etukäteen oli tiedossa, että komponenttiin tullaan melko varmuudella myöhemmin lisäämään uusia ominaisuuksia ja tukia kuvankaappauskorteille, tuli arkkitehtuurin mahdollistaa se, että komponentista ei tarvitse lisäysten myötä tehdä rinnakkaisia versioita. Komponentin kustakin kehitysasteesta on kerrallaan olemassa vain yksi lähdekoodi, ja siitä tehdään yksi käännös, johon kaikki toiminnot sisältyvät. Tämä helpottaa oleellisesti jatkokehitystä ja ylläpitoa. Uusissa versioissa pyritään säilyttämään taaksepäin yhteensopivuus, jotta komponentin päivitykset eivät aiheuta välitöntä muutostarvetta sitä käyttäviin asiakasohjelmiin. Lisäksi komponentista oli tarkoitus tehdä mahdollisimman yleiskäyttöinen, jotta sitä voitaisiin hyödyntää useissa eri sovelluksissa.

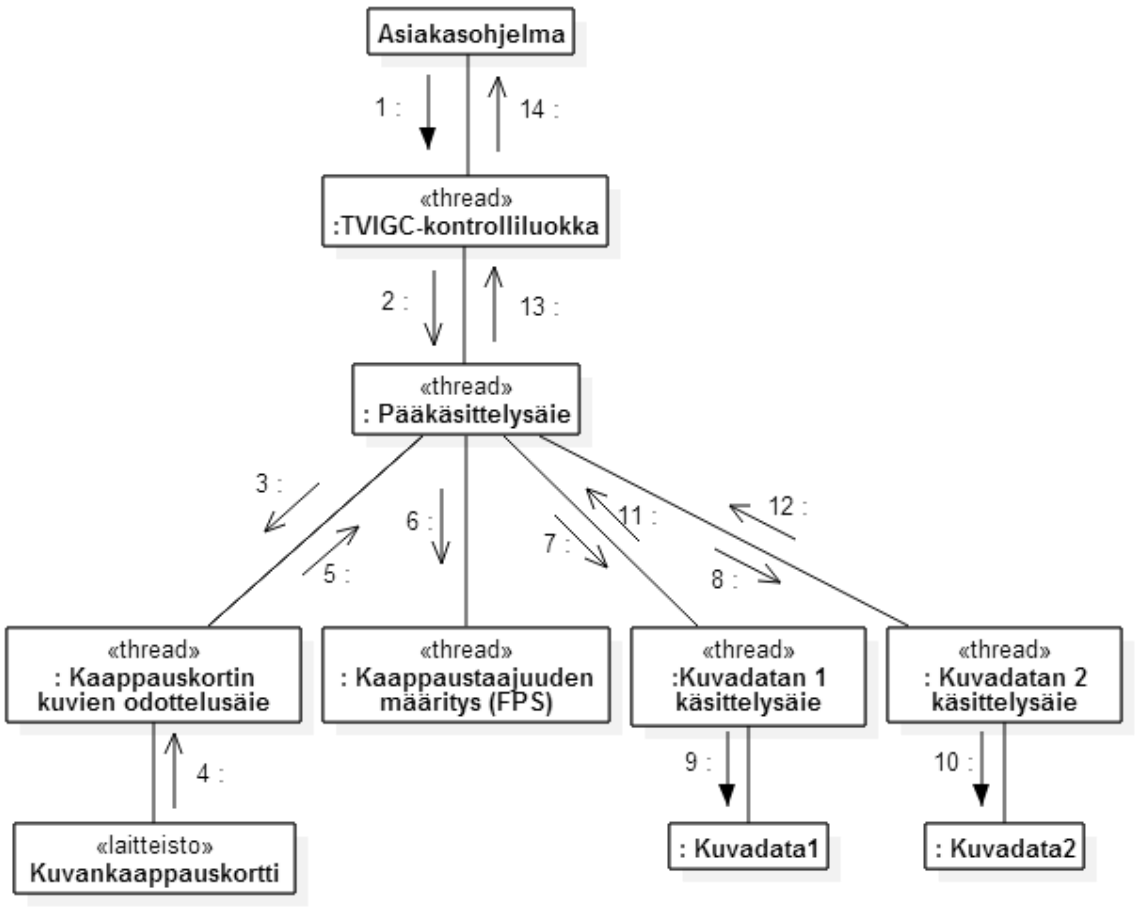
5.3.1 Komponentin arkkitehtuuri

Komponentti kuuluu aina osaksi kokonaista järjestelmää, johon se liittyy julkisen asiakasohjelmarajapintansa kautta sekä kuvankaappauskortin ohjelmistoliitynnän kautta, joka koostuu yleensä kaappauskorttivalmistajan toimittamista ohjelmistomoduuleista. Luvussa 5.2.1 esitetty kuva 4 havainnollistaa komponentin liittymistä ympäristöönsä.

Komponentin osittamista suunniteltaessa lähdettiin siitä, että komponentilla tulee olemaan kaksi päätehtävää: kuvadatan käsittely ja säilytys sekä kuvankaappauskortin ohjaus. Nämä kumpikin oli helppo nähdä komponentin avainabstraktioina. Komponentin kuvankaappauskorttirajapintaan luotiin abstrakti luokka, joka määrittelee palvelut, jotka kuvankaappauskorttikohtaisten aliluokkien on toteutettava. Kuvadatat ovat kuvankaappauskortin tuottamasta raakadatasta laskettuja tai haluttuun muotoon muodostettuja datoja. Muodostetut datat voivat olla sisällöiltään hyvinkin erilaisia keskenään, mutta koska niiden käsittely- ja hallintarajapinta voidaan nähdä jo määrittelyn perusteella varsin yhteneväisenä, datoille luotiin yksi yhteinen yliluokka, josta eri kuvadatoille periytetään aliluokat.

Kuvankaappauskorttien muodostama raakadata on kortista ja kaappausasetuksista riippuen korttivalmistajan tai liityntästandardin määrittelemässä muodossa, joka voi vaihdella paljonkin. Jotta komponentin raakadatasta tekemien kuvankäsittelyoperaatioiden toteutus yksinkertaistuu, muunnetaan kaapattu raakadata ensin määrättyyn vakiomuotoon, jota kaikki kuvankäsittelyoperaatiot käyttävät. Tämä lisää hieman muistinkulutusta ja alentaa suoritustehokkuutta, mutta helpottaa oleellisesti ylläpitoa, ja etenkin uusien ominaisuuksien lisäämistä.

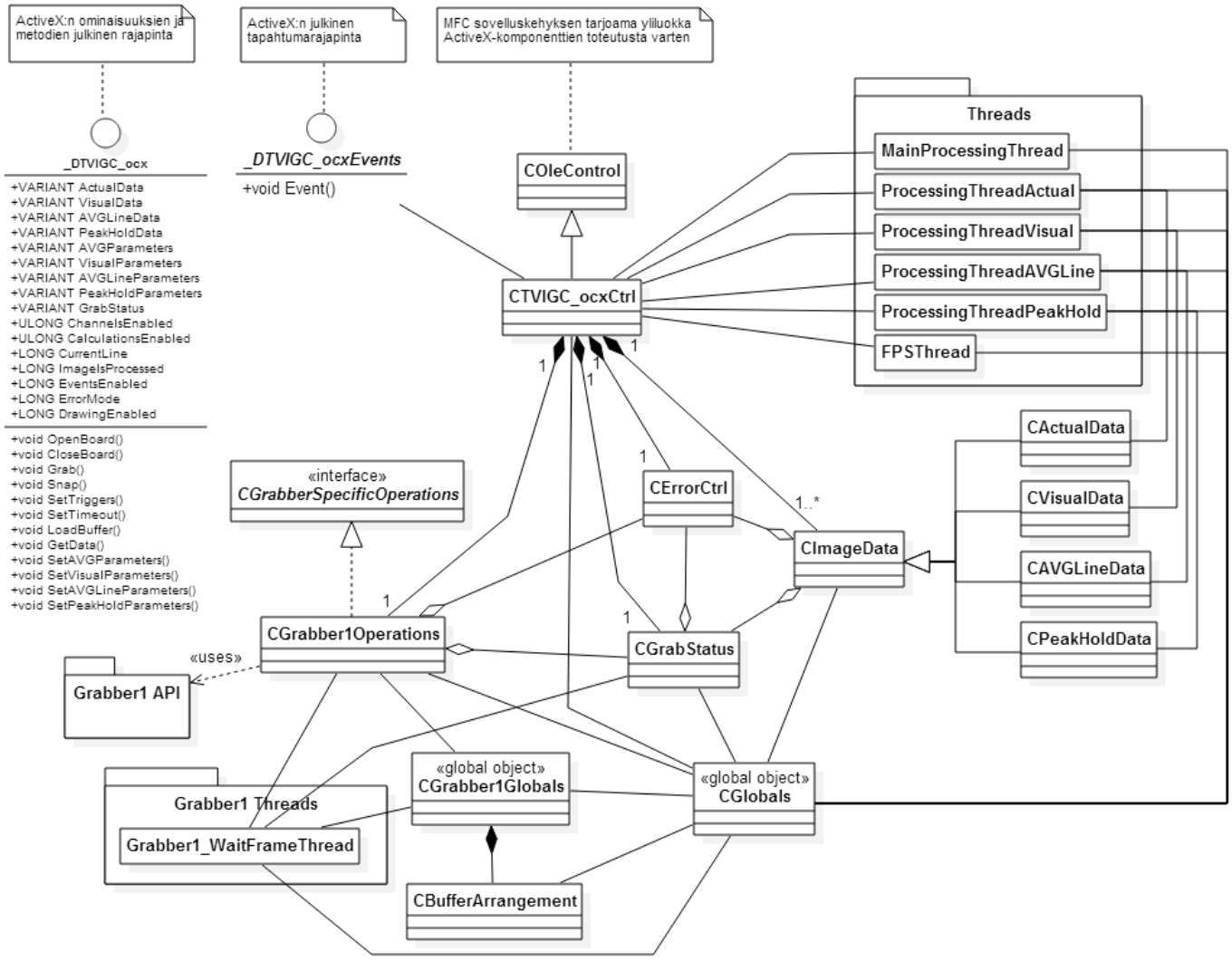
Koska kuvadatat muodostetaan kaapatusta raakakuvasta toisistaan riippumattomasti, voidaan kunkin kuvadatan laskenta suorittaa omassa säikeessään. Tätä varten luotiin kutakin kuvadataa varten globaali säiefunktio. Säikeet voidaan käynnistää yhtä aikaa, jolloin hyödynnetään moniydinprosessoreiden tarjoamaa rinnakkaislaskentaa ja saadaan siten lisättyä tehokkuutta. Myös kuvankaappaukseen uusien, valmistuvien kuvien odottelua varten on tarpeen luoda erillinen säie (MainProcessingThread), jotta ohjelman pääsäikeen suoritus ei keskeydy. Kuvien odottelun pääsäikeen toteutus kirjoitettiin globaaliin funktioon, ja se käynnistetään komponentin kontrolliluokasta. Uuden kuvan valmistuttua säie vapauttaa odotustilassa olevat kuvankäsittelysäikeet. Lisäksi kuvien odottelusäikeestä oli luotava vielä kuvankaappauskorttikohtainen toteutus, jonka suorituksen käynnistää kuvankaappauskorttikohtainen olio. Tämä kuvankaappauskorttikohtainen säie informoi kuvien odottelun pääsäiettä valmiista kuvista. Liitteessä 1 on esitetty tapahtumasekvenssikaavio käynnissä olevan kuvankaappauksen toistorakenteesta. Näiden lisäksi luodaan vielä yksi säie, joka laskee käynnissä olevan kuvankaappauksen kuvien kaappaustaajuuden, joka ilmaisee, kuinka monta kuvaa sekunnissa kaapataan (FPS, Frames Per Second). Säikeiden välistä kommunikointia havainnollistaa kuvassa 15 oleva yhteistyökaavio. Säikeiden yhteisiä tietoja koottiin omiin luokkiin, joista luotujen globaaleiden olioiden viittaukset annettiin tietoja tarvitseville säiefunktioille tai luokille.



Kuva 15. Prosessissa suorituksessa olevien säikeiden välistä kommunikointia havainnollistava yhteistyökaavio. Asiakasohjelma ja TVIGC-komponentti suoritetaan prosessin pääsäikeessä. Muut säikeet ovat komponentin luomia toissijaisia säikeitä. Aktiivisten olioiden välinen kommunikointi on aina asynkronista (tikkunuolenpäät). Tavallisten olioiden kanssa kommunikointi voi olla myös synkronista (täytetyt nuolenpäät). Asiakasohjelmalta komponentille tulevat metodikutsut ovat synkronisia, komponentti taas kommunikoi asiakasohjelman kanssa viestien avulla. Kuvankaappauslaitteistolta tulevat viestit ovat asynkronisia. Numerointi kuvaa viestien järjestystä, joka voi vaihdella erityisesti rinnakkain suoritettavien kuvadatan käsittelysäikeiden tapauksessa. Käsittelysäikeet viestivät pääkäsittelysäikeelle siinä järjestyksessä, kun niiden suoritus valmistuu.

Komponentin kaappausparametritaulukon eli tiladatan ylläpitoa sekä komponentissa syntyvien virhetilanteiden käsittelyä varten luotiin omat luokat. Kummastakin luokasta luotiin komponentissa yhdet oliot, joiden viittauksia jaettiin kaikille niille komponentin sisäisille moduuleille, jotka kyseisiä toimintoja tarvitsevat.

Edellä esitellyistä moduuleista laadittiin kuvan 16 mukainen luokkakaavio. Luokkien tarkempi esittely on seuraavassa luvussa.



Kuva 16. TVIGC-komponentin luokkakaavio.

5.3.2 Moduulisuunnittelu

Tässä luvussa on esitelty lyhyesti tärkeimmät komponenttiin toteutetut luokkamoduulit.

CTVIGC_ocxCtrl on ActiveX-kontrolliluokka, johon kirjoitetaan kontrollin toteutuskoodi. Se periytetään MFC:n luokasta COleControl, joka taas tarjoaa joukon funktioita ActiveX-kontrollin toimintaan liittyen. Luokka sisältää komponentin julkisten rajapintojen ominaisuuksien jäsenmuuttujat, metodien toteutusfunktiot ja tapahtumien laukaisemiseen tarvittavat funktiot. [8, s. 436.] Luokassa luodaan oliot kaappauskorttikohtaiseen luokkaan, kuvadataluokkiin, kaappausparametriluokkaan ja virheiden käsittelyluokkaan. Luokka sisältää myös komponentin käyttöliittymän piirtokoodin, joka tosin voitaisiin erottaa omaan moduuliinsa.

CGrabberSpecificOperations on abstrakti luokka kaappauskortti- tai laitteistokohtaisten aliluokkien toteutusta varten. Luokka ei sisällä toteutuksia, sillä kaappauskorttien ohjelmistorajapinnat ovat poikkeuksetta erilaisia. Kuitenkin useiden eri kaappauskorttivalmistajien ohjelmistorajapintoja tutkimalla havaittiin niiden olevan toimintaperiaatteeltaan varsin samankaltaisia, jolloin rakennetta päätettiin soveltaa myös abstraktin luokan määrittelyssä. Luokka sisältää seuraavat puhtaat virtuaaliset metodit: InitBoard, CloseBoard, GrabImages, SnapImage, GetBufferPointer, SetTimeout ja SetTriggersToBrd.

Kaappauskorttikohtaisten luokkien (esimerkkinä **CGrabber1Operations** luokkakaaviossa) tehtävänä on toimia rajapintana komponentin ja kortin välillä. Luokat käyttävät korttivalmistajien toimittamia ohjelmointirajapintoja (API). Luokkien avulla ohjataan kaappauskortteja siten, että tuloksena on muistiin kaapattua raakakuvadataa. Korttikohtaisista aliluokista voidaan tarvittaessa vielä periyttää aliluokkia, esimerkiksi saman korttivalmistajan eri malleja varten. Tällöin yliluokkaan voidaan todennäköisesti joillekin metodeille kirjoittaa oletustoteutukset. Kerrallaan vain yksi kaappauskorttiolio voi olla luotuna ajon aikana.

Kullekin kaappauskorttikohtaiselle luokalle kirjoitettiin globaali säiefunktio (**Grabber1_WaitFrameThread**), jonka suorituksen omassa säikeessään käynnistävät luokan GrabImages- ja SnapImage-metodit. Funktio koostuu uusia kaapattuja kuvia odottavasta silmukasta. Lisäksi niitä kaappauskorttikohtaisia tietoja, kuten kortin kahva, muistipuskureiden osoittimet ja vakiomuotoon järjestetty kuvapuskuridata, joita tarvitaan sekä korttikohtaisessa luokassa että korttikohtaisessa säiefunktiossa, varten luotiin säieturvallinen luokka (**CGrabber1Globals**), jonka globaalin olion avulla tiedot on luettavissa ja kirjoitettavissa.

Kaappauskortin kaappaamasta raakakuvadatasta laskettavat ja muodostettavat valmiit kuvadatat säilötään omissa olioissaan, jotka luodaan **CImageData**-luokasta. Luokka sisältää myös datan käsittelymetodit ja käsittelyä ohjaavat parametrit, joita asiakasohjelmassa voidaan asettaa. Erityyppisiä datoja varten CImageData-luokasta periytettiin aliluokat, kuten muun muassa **CActualData** tai **CVisualData**, joihin toteutettiin datatyyppikohtainen kuvien käsittely. Kuvadatojen käsittelyn samankaltaisuudesta johtuen yliluokkaan kirjoitettiin oletustoteutukset yleisimmille metodeille, jotka voidaan tarvittaessa korvata aliluokissa. Käsittely käynnistetään kutakin kuvadataa vastaavasta säiefunktiosta (esim. **ProcessingThreadActual**) kutsumalla CImageData-luokan metodia

SetImageData, jolle annetaan parametrina osoitin vakiomuotoon järjestettyyn raakakuvapuskuriin. Vakiomuotoon järjestys tehdään **CBufferArrangement**-luokassa, johon luotiin metodeita erityyppisiä kaappauskorttien tuottamia raakakuvadatoja varten.

Kaikkien säikeiden ja luokkien yhteisessä käytössä olevat globaalit muuttujat kapseloitiin säieturvalliseen luokkaan **CGlobals**. Luokka sisältää muutamien lippumuuttujien lisäksi komponentissa luotavien säikeiden osoittimet ja säikeiden synkronointioliot. Muuttujien käsittely tapahtuu kriittisten alueiden synkronointimenetelmällä suojatuissa get- ja set-metodeissa. Luokasta luotiin yksi globaali olio.

Komponentin kaappausparametrit, jotka sisältyvät määrittelyssä esitettyyn parametritaulukkoon (luku 5.2.7), kapseloitiin **CGrabStatus**-luokkaan samalla tavalla kuin CGlobals-luokan globaalit muuttujat.

CErrorCtrl-luokan tehtävä on huolehtia virhetilanteiden ilmoittamisesta asiakasohjelmalle ja käyttäjälle valitulla tavalla. Mahdollisista virhetilanteista luotiin tietuetaulukko, johon talletettiin virhekoodit ja niitä vastaavat tekstimuotoiset virheilmoitukset. Taulukosta on ote koodiesimerkissä 1.

```
TVIGC_ERRORS CErrorCtrl::ErrorTable[] =
{
  {1000, _T("OK")},                                  //TVIGC_OK
  {1001, _T("Unknown error occurred.")},             //TVIGC_ERROR_UNKNOWN

  //Grabber initialization & setting errors
  {1050, _T("Grabber error")},                       //TVIGC_ERROR_GRABBER
  {1102, _T("Grabber initialization failed.")},      //TVIGC_ERROR_BOARDINIT

  //Startup & resource allocation and accessing errors
  {2000, _T("Memory allocation failed.")},           //TVIGC_ERROR_MEMALLOC
  {2001, _T("Memory allocation for Actual Data failed.")},
                                          //TVIGC_ERROR_MEMALLOCACTUALDATA
  //Grabbing errors
  {4000, _T("Grabbing is already stopped.")},        //TVIGC_ERROR_GRABSTOPPED
  {4020, _T("Grab start timed out.")},               //TVIGC_ERROR_GRABSTARTTIMEOUT
};
```

Koodiesimerkki 1. Ote virhekooditaulukosta. Virheet on ryhmitelty niiden tyypin mukaan. Rivien perässä on kommenteissa riveille annetut muistikasnimet, joilla taulukon rivinumerot on vakioitu C++-kielen luetellun tyypin enum avulla taulukon riveihin viittaamisen helpottamiseksi.

5.3.3 Komponentin asiakasohjelmarajapinta

TVIGC ActiveX -kontrollin tärkein rajapinta on sen julkinen rajapinta, jonka kautta se tarjoaa palveluja asiakasohjelmalle. Rajapinnan kautta voidaan asiakasohjelmassa lukea ja asettaa komponentin ominaisuuksia, kutsua metodeita sekä sanomarajapinnan kautta vastaanottaa tapahtumia. Rajapinnasta voi olla toteutettuna samassakin komponentissa useita versioita, joita syntyy, kun rajapintaan tehdään muutoksia uusien komponenttiversioiden myötä. Tämä helpottaa taaksepäin yhteensopivuuden säilyttämistä. Rajapintaan voidaan katsoa kuuluvaksi myös komponentin käyttöliittymä, jota asiakasohjelma voi tarvittaessa hyödyntää. Seuraavassa on esitelty rajapinnan tarjoamat toiminnot. Ominaisuuksien ja metodien tarkemmat kuvaukset ovat liitteessä 2.

Ominaisuudet

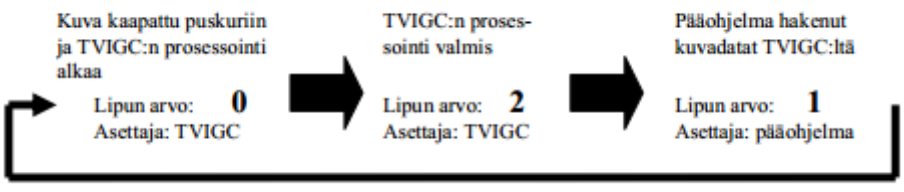
ActualData on VARIANT-tietotyyppiin käärittyunsigned long -tyyppinen taulukko, joka sisältää komponentin muodostaman viivadatan, joka on määritelty luvussa 5.2.7. VARIANT on standardi tietorakenne, johon on kapseloitu varsinaisen tiedon lisäksi kenttä, joka ilmaisee tiedon tyypin. VARIANT-tietotyypin avulla voidaan välittää monimutkaisempia tietorakenteita eri ohjelmointikielten välillä. [8, s. 357-360.] Asiakasohjelmalla on tähän ominaisuuteen vain lukuoikeus. Vastaavia ominaisuuksia ovat **VisualData**, joka sisältää komponentin muodostaman bittikarttakuvadatan; **AVGLineData**, joka sisältää pikseleiden keskiarvodatan; **PeakHoldData**, joka sisältää pikselikohtaisten ääriarvojen seurantadatan; **GrabStatus**, joka sisältää kaappausparametritaulukon.

Edellä esiteltyjen kuvadatojen laskentaa ja muodostusta ohjataan aseteltavien parametrien avulla. Kullekin kuvadatalle on oma ominaisuutensa, joka koostuu ohjausparametrit sisältävästä VARIANT/long-tyyppisestä taulukosta. Nämä ominaisuudet ovat sekä luettavissa että kirjoitettavissa asiakasohjelmassa. **AVGParameters**-ominaisuudella määritetään viivadatan (ActualData) paikalle talletettavan viivojen keskiarvon laskennan toiminta. Bittikarttakuva muodostuu kolmesta osaväristä RGB-värimallin mukaisesti, joten mikäli kamerassa on värikanavia vähemmän tai enemmän kuin kolme, voidaan **VisualParameters**-ominaisuudella valita, mitkä kolme värikanavaa bittikarttakuvassa (VisualData) näytetään. **AVGLineParameters**-ominaisuudella ohjataan pikseleiden keskiarvodatan (AVGLineData) laskentaa, ja **PeakHoldParameters**-ominaisuudella pikselikohtaisten ääriarvojen seurantadatan (PeakHoldData) laskentaa.

ChannelsEnabled on 32-bittinen unsigned long -tyyppinen muuttuja, jonka kukin bitti toimii yhtä värikanavaa edustavana lippuna. Lippu määrittelee, suoritetaanko kuvadatojen laskentaoperaatiot ko. värikanavan osalta. Toiminto on määritelty tarkemmin luvussa 5.2.7, kohdassa *Käsiteltävien värikanavien valinta*. Ominaisuus on sekä luettavissa että kirjoitettavissa asiakasohjelmassa. **CalculationsEnabled** on muuten vastaava bittikenttämuuttuja, paitsi että sillä valitaan halutut kuvadatojen laskentaoperaatiot.

CurrentLine on 32-bittinen long-tyyppinen muuttuja, jolla määritetään viiva, joka kaapatusta raakakuvadatasta poimitaan viivadata-tietueeseen (ActualData). Viivojen indeksointi alkaa nollasta ja suurin muuttujaan tallennettu arvo voi olla kaapatun kuvan viivojen lukumäärä vähennettynä yhdellä. Muuttujalle voidaan antaa myös arvo -1, jolloin koko kuva (kaikki viivat) talletetaan viivadata-tietueeseen, kuten määritelty luvun 5.2.7 kohdassa *Viivadatan laskenta*. Muuttujan avulla toteutetaan myös viivojen selaus -toiminto, kuten määritelty em. luvun kohdassa *Kuvapuskureiden selaus*. Ominaisuus on sekä luettavissa että kirjoitettavissa asiakasohjelmassa.

ImageIsProcessed on long-tyyppinen lippu-muuttuja, jota käytetään kuvankaappauksen ja -käsittelyn synkronointiin komponentin ja asiakasohjelman välillä. Lipun avulla komponentti ilmoittaa asiakasohjelmalle valmistuneista kaapatuista ja käsitellyistä kuvista. Samoin asiakasohjelma ilmoittaa komponentille oman käsittelynsä valmistumisesta. Lippu voi saada arvoja kuvan 17 mukaisessa syklissä. Asiakasohjelma pollaa lipun arvoa. Komponentti asettaa lipun arvoksi 0 jokaisen uuden kuvan kaappausta aloitettaessa. Kun kuvan kaappaus ja komponentin suorittama prosessointi valmistuu, komponentti asettaa lipun arvoksi 2. Asiakasohjelman prosessoitua kuvan, se asettaa lipun arvoksi 1. Kaappaussekvenssi on esitetty tarkemmin luvun 5.2.7 kohdassa *Jatkuvan kuvankaappauksen käynnistys ja pysäytys* sekä liitteessä 1.



Kuva 17. ImageIsProcessed-ominaisuuden asettelusykli.

EventsEnabled on long-tyyppinen muuttuja, jonka avulla otetaan käyttöön (arvo 1) tai poistetaan käytöstä (arvo 0) valmiista kuvasta ilmoittaminen asiakasohjelmalle tapahtumien avulla. Jos toiminto on käytössä, komponentti laukaisee tapahtuman, jonka asiakasohjelma vastaanottaa. Täten asiakasohjelman ei tarvitse pollata ImageIsProcessed-lippua. Lippu kuitenkin asetetaan normaalisti.

ErrorMode on long-tyyppinen muuttuja, jolla määritetään, kuinka komponentin generoimat virheilmoitukset esitetään käyttäjälle. Komponentti voi mm. generoida dialogin tai laukaista tapahtuman virhekoodin kanssa.

DrawingEnabled on long-tyyppinen muuttuja, jolla määritetään, mikä komponentin muodostamista kuvadatoista sen käyttöliittymään piirretään.

Metodit

OpenBoard(…)-metodilla alustetaan komponentin kaappausrajapinta eli useimmiten kuvankaappauskortti ja varataan muistia kuvapuskureita varten. Parametreina annetaan tiedot kuvan koosta, kaappauskortin tyypistä jne. Parametreille voidaan antaa arvo 0, jolloin komponentti käyttää oletusarvona kaappauskortin konfigurointitiedostosta saatavaa arvoa. **CloseBoard()**-metodilla vapautetaan varatut resurssit, kuten muisti ja kuvankaappauskortti.

Grab(long Start)-metodilla aloitetaan ja pysäytetään kuvankaappaus muistipuskureihin. Start-parametrin arvolla 1 käynnistetään kaappaus ja arvolla 0 pysäytetään kaappaus odottaen, että kaikki kuvapuskurit on käsitelty. Arvolla 2 pysäytetään kaappaus välittömästi. GrabStatus-ominaisuuden *Kaappaus käynnissä/pysäytettynä* -arvo vaihtuu arvoksi 1 heti, kun kaappaus alkaa, ja arvoksi 0 sitten, kun viimeinenkin kaapattu kuva on käsitelty. ImageIsProcessed-ominaisuus kertoo, milloin kuva on valmis haettavaksi, tai vaihtoehtoisesti tieto saadaan tapahtumien avulla. **Snap()**-metodia käytetään kuten Grab-metodia, mutta jokaisella Snap-metodin kutsulla kaapataan vain yksi kuva, jonka jälkeen kaappaus pysäytetään. Valmiista kuvasta ilmoitetaan kuten Grab-metodia käytettäessä.

SetTriggers(…)-metodilla määritetään kuvankaappauksessa käytettävät liipaisuasetukset sekä viivan että kuvan liipaisupulssien osalta. Viivan liipaisun (horizontal encoder) avulla yksittäisiä viivoja kaapataan kortille signaalilähteestä tulevien pulssien

tahdissa, kun taas kuvan liipaisua (vertical trigger) käytettäessä pulssi aloittaa koko kuvan kaappauksen. Molempia voidaan käyttää myös samanaikaisesti. Asetuksia voi muuttaa lennossa kesken kuvankaappauksen.

SetTimeout(ULONG Timeout)-metodilla asetetaan uusi kaappauksen aikakatkaisun yläraja-arvo millisekunteina. Arvo on luettavissa GrabStatus-ominaisuuden sisältämästä taulukosta. Oletuksena raja-arvo otetaan konfigurointitiedostosta. Raja-arvo voidaan asettaa rajattomaksi (infinite) antamalla parametriksi 0.

LoadBuffer(LONG BufferNumber)-metodilla ladataan kaapattujen, muistissa olevien raakakuvapuskureiden sisältöjä tarkasteltavaksi silloin, kun kaappaus on pysäytettynä. BufferNumber-parametrilla kerrotaan halutun kuvan indeksi. Vanhimman kaapatun kuvan indeksi on 0 ja tuoreimman indeksi saadaan kaavalla puskureiden_lkm - 1. Kuvalle suoritetaan ladattaessa samat laskennat ja muodostusoperaatiot asetettujen asetusten mukaisesti kuin kaapattaessakin. Ladattu kuva siis luetaan asiakasohjelmassa ActualData-, VisualData- jne. ominaisuuksien kautta. Kun lataus on valmis, komponentti asettaa ImageIsProcessed-lipun arvoksi 2 (sekä lähettää tapahtuman, jos toiminto käytössä), joka asiakasohjelman tulee kuitata arvolla 1 sitten, kun asiakasohjelma on valmis kuvan haun kanssa.

Vaikka ominaisuudet voidaankin lukea suoraan rajapinnan kautta, saattaa joissain tapauksissa olla käytännöllisempää lukea ominaisuuksien arvot osoitinmuuttujan avulla, etenkin VARIANT-tyyppisten taulukoiden tapauksessa. **GetData(LONG DataIndex, ULONG DataPointer)**-metodi kopioi DataIndex-parametrilla määrätyn datan sellaisenaan alkio alkiolta DataPointer-parametrilla saadun osoittimen osoittamaan muistiin, joka on varattava asiakasohjelmassa. Parametrit on esitetty tarkemmin liitteessä 2. Koodiesimerkissä 2 näytetään, kuinka asiakasohjelmassa haetaan ActualData-ominaisuuden sisältämä data.

```
long lViivat = 0;    //viivojen lkm
long lPikselit = 0; //pikseleiden lkm (sensorin pituus)
long lVarit = 0;     //värikanavien lkm

//varataan muisti GrabStatus-taulukkoa varten
unsigned long ulGrabStatus[21];

//haetaan GrabStatus-taulukko komponentilta
m_tvigc.GetData(0, (ULONG)ulGrabStatus);

//poimitaan taulukosta halutut alkiot
lViivat = ulGrabStatus[1];   //viivojen lkm
lPikselit = ulGrabStatus[2]; //pikseleiden lkm
lVarit = ulGrabStatus[18];   //värikanavien lkm

//varataan muisti ActualData-kuvadataa varten
unsigned long* ulViivaData = NULL;
unsigned long ulKuvanKoko = lViivat*lPikselit*lVarit;
ulViivaData = new unsigned long[ulKuvanKoko];

//haetaan ActualData komponentilta
m_tvigc.GetData(1, (unsigned long)ulViivaData);
```

Koodiesimerkki 2. ActualData-ominaisuuden sisältämän kuvadatan nouto C++-kielellä toteutetussa asiakasohjelmassa. Esimerkin tapauksessa CurrentLine-ominaisuuteen on asetettu arvo "-1", joten ActualData sisältää kaikki kaapatut viivat.

Vaikka ominaisuuksia voidaan myös asettaa suoraan rajapinnan kautta, toteutettiin komponenttiin mahdollisuus asettaa VARIANT-tyyppiset ominaisuudet metodien parametrien avulla. Nämä metodit ovat: **SetAVGParameters(…)**, **SetVisualParameters(…)**, **SetAVGLineParameters(…)** ja **SetPeakHoldParameters(…)**. Metodien parametrit vastaavat kunkin ominaisuuden kautta asetettavia parametreja.

Tapahtumat

Komponentti voi ilmoittaa uuden kuvan valmistumisesta tai virhetilanteesta laukaisemalla tapahtuman **Event(LONG EventCode)**, jonka asiakasohjelma vastaanottaa ja käsittelee. EventCode-parametrilla toimitetaan koodi, joka kertoo tapahtuman merkityksen. Koodilla 0 ilmoitetaan valmiista kuvasta. Muut koodit ilmaisevat virhetilannetta. Tapahtumien laukaisu voidaan kytkeä päälle tai pois EventsEnabled- ja ErrorMode-ominaisuuksien avulla. Virhetilanteen jälkeen on suositeltavaa asiakasohjelmassa lukea GrabStatus-ominaisuudesta, missä tilassa komponentti on, erityisesti *Kaappaus käynnissä/pysäytettynä*- ja *Kaappaus alustettu/alustamatta* -kentät. Tällöin asiakasohjelman pystyy asettamaan vastaavaan tilaan.

5.4 Komponentin toteutus

Komponentin toteutus eli ohjelmointi suoritettiin teknisen suunnittelun pohjalta. Kehitysympäristönä käytettiin Microsoftin Visual Studio 2010:ää. Visual Studiolla voidaan luoda valmis runko ActiveX-kontrolleille, josta kehitystyö lähti helposti käyntiin. Toimintoja toteutettiin vähitellen luoden versioita, joita voitiin tarjota samanaikaisesti varsinaista testausohjelmistoa kehittäneen insinööritoimiston käyttöön. Järjestelmien integrointia ja virheiden jäljitystä suoritettiin myös ketterien menetelmien pariohjelmointia käyttäen.

Erityisesti rinnakkaisuuden toteuttaminen ja säikeiden synkronoinnin toimimaan saanti osoittautuivat melko haasteellisiksi. Kaappaussekvenssin määritelmää ja suunnitelmaa jouduttiin tarkentamaan useaan otteeseen toteutuksen aikana. Virhetilanteita saattoi esiintyä hyvin harvakseltaan ja niiden toistaminen keinotekoisesti oli vaikeaa tai mahdotonta. Sekvenssissä havaittujen virheiden jäljitys (debugging) ja esiin saaminen kääntäjän työkaluilla (debugger) osoittautui niin ikään hyvin vaikeaksi ilmeisesti virheenjäljittimen hidastaessa ohjelman suoritusta siten, että rinnakkain ajossa olevien säikeiden suoritus pysyi helpommin ja pidempään oikeassa tahdissa. Jotta ohjelmaa voitiin ajaa reaaliaikaisesti sen luontaisella nopeudella ja kerätä samalla tietoja suorituksen etenemisestä, komponenttiin rakennettiin yksinkertainen virheiden jäljitysmekanismi. Haluttuihin kohtiin koodissa lisättiin rivi, joka kirjoittaa aina samaan tiedostoon merkinnän suorituskohdasta ja tarvittaessa myös tietoa haluttujen muuttujien senhetkisistä arvoista. Toisinaan ohjelmaa saattoi joutua ajamaan pitkäänkin ennen vian esiintymistä. Tässä tapauksessa mekanismi nopeutti vianhakua, sillä ohjelmaa voitiin ajaa itsekseen tarvittavan ajan, ja lopuksi analysoida raporttia. Myös mekanismi luonnollisesti hidastaa suoritusta ja voi vaikuttaa säikeiden suoritusjärjestykseen ja komponentin toimintaan. Valmiin, virheettömän, oikein synkronoidun sekvenssin ja sen toteutuksen tulee luonnollisesti toimia siten, että tauot ja viiveet tai satunnaiset järjestelmän hidastelutkaan eivät vaikuta suoritukseen, koska kuvien kaappaukselle ja niiden käsittelylle (sekä komponentissa että asiakasohjelmassa) ei voida ennustaa kestoaikaa.

Komponentissa luotavat toissijaiset säikeet käynnistetään MFC:n AfxBeginThread()-funktiolla, joka luo säiettä edustavan CWinThread-olion. Olion avulla voidaan selvittää esimerkiksi säikeen suorituksen tilaa. Funktiolle annetaan parametreina muun muassa säiefunktion osoitin, funktiolle välitettävät parametrit ja säikeen prioriteetti. [8, s. 197-200.] Koodiesimerkki 3 havainnollistaa säikeen luontia.

```
void CTVIGC_ocxCtrl::Grab(LONG Start)
{
  //varmistetaan, että kaappaus ei ole jo käynnissä, ja että
  //kaappausta ollaan käynnistämässä (Start parametri)
  if(m_pStatusObj->GetGrabStatusValue(GRABBING) != 1 && Start == 1)
  {
    //asetetaan lippuja kaappauksen ja säikeiden käynnissä olemisen
    //ilmaisemiseksi
    m_pStatusObj->SetGrabStatusValue(GRABBING, TRUE);
    g_GlobObj.SetRunning(TRUE);

    //luodaan ja käynnistetään säikeet

    //Varmistetaan, että säiettä ei ole jo luotu eli osoitin säikeen
    //CWinThread-olioon on 0. Osoittimia säilytetään globaalissa
    //luokassa CGlobals.
    if(!g_GlobObj.GetThreadPtr(PROCESSINGTHREADACTUAL))
    {
      //Käynnistetään viivadatan käsittelysäie. Parametreina annetaan
      //säiefunktion osoitin, säiefunktiolle välitettävänä parametrina
      //viivadataolion osoitin sekä säikeen prioriteetti.
      CWinThread* pActual = AfxBeginThread(ProcessingThreadActual
                                           ,(LPVOID)m_pActualObj
                                           ,THREAD_PRIORITY_HIGHEST));
        //talletetaan luodun CWinThread-olion osoitin
      g_GlobObj.SetThreadPtr(PROCESSINGTHREADACTUAL, pActual);
    }
  }
}
```

Koodiesimerkki 3. Ote komponentin asiakasrajapinnan metodista Grab(), jossa luodaan kaappausta käynnistettäessä säie viivadatan (ActualData) käsittelyä varten.

Komponentin säiefunktioiden toteutus koostuu ikuisesta silmukasta, jossa odotetaan muilta säikeiltä tai kuvankaappauskortilta tulevia tapahtumia suorituksen jatkamiseksi sekä suoritetaan säikeille suunniteltuja tehtäviä, kuten kuvankäsittelyä. Säikeiden silmukan yksi kierros vastaa yhtä kaapattua kuvaa. Säikeiden suoritus on toisiinsa nähden asynkronista, joten jotta yksikään säie ei lähde etenemään eri tahdissa toisiin nähden tai jotta säikeet eivät lue ja kirjoita yhteisiä tietoja yhtä aikaa, niiden suoritusta jouduttiin kontrolloimaan käyttäen sopivia synkronointimenetelmiä. Windows ja MFC tarjoavat monipuoliset toiminnot synkronoinnin toteuttamiseen. Säikeiden suoritusten synkronointiin käytettiin tapahtumia (event). Tapahtumaolio voi olla joko signaloivassa tai signaloimattomassa tilassa. Tapahtumia kuunnellaan kutsumalla Windows API:n funktiota WaitForSingleObject(). Funktio ottaa säikeen pois ajosta siksi aikaa, kunnes kuunneltava tapahtuma muuttuu signaloivaksi, jonka jälkeen säie voi jatkaa suoritustaan. Kun säie on pois ajosta, se ei kuluta prosessoriaikaa lainkaan. [13.] Koodiesimerkissä 4 on esitetty viivadatan käsittelysäiefunktion toteutus.

```
UINT ProcessingThreadActual(LPVOID lpdwParam)
{
  //vastaanotetaan osoitin viivadataolioon
  CActualData* pActualObj = (CActualData*)lpdwParam;
  //käynnistetään silmukka, jota suoritetaan, kunnes säikeiden suoritus
  //halutaan lopettaa
  while(g_GlobObj.GetRunning())
  {
    //haetaan kuunneltavan tapahtuman osoitin
    CEvent* pStartA;
    pStartA = g_GlobObj.GetGlobalEvent(EVENTSTARTPROCESSINGTHREADACTUAL);

   //kuunnellaan tapahtumaa niin kauan kunnes se asetetaan signaloivaksi,
   //mikä tehdään kuvien pääkäsittelysäikeessä (MainProcessingThread)
   //uuden kaapatun kuvan tullessa käsiteltäväksi tai kun koko ohjelma
   //halutaan sulkea, jolloin säikeet on ensin lopetettava.
    WaitForSingleObject(pStartA->m_hObject, INFINITE);
    //tarkistetaan lipun arvo, onko säikeet merkattu lopetettaviksi
    if(!g_GlobObj.GetRunning())
      break;
    //kutsutaan viivadataluokan metodia, jossa varsinainen kuvankäsittely
    //tapahtuu
    pActualObj->SetImageData(g_GlobObj.GetImageBufferPtr());
   //asetetaan tapahtuma, jolla halutaan viestittää pääkäsittelysäikeelle
   //viivadatan käsittelyn valmistumisesta, signaloivaan tilaan
    (g_GlobObj.GetGlobalEvent(EVENTPROCESSACTUALREADY))->SetEvent();
  }

  //Säikeen lopetus. Asetetaan CWinThread-olion osoitin nollaksi.
  g_GlobObj.SetThreadPtr(PROCESSINGTHREADACTUAL, NULL);

/*DEBUG - esimerkki komponentissa käytetystä virheidenjäljitysmekanismis-
ta, jossa koodiin on lisätty debug-dataa tiedostoon kirjoittavan metodin
kutsu*/
g_GlobObj.WriteGCDebugDataLog(0, _T("GC: Close:ProcessingThreadActual"));

  //päätetään säikeen suoritus
  AfxEndThread(0, false);
  //palautetaan arvo 0, joka merkitsee funktion onnistunutta suoritusta
  return 0;
}
```

Koodiesimerkki 4. Viivadatan (ActualData) käsittelysäikeen säiefunktio.

Yhteisten tietojen suojaamiseksi käytettiin synkronointimenetelmänä *kriittisiä alueita* (critical sections). Ennen suojattavaa tietoa käsittelevää koodia kutsutaan yhteisen synkronointiolion saantifunktiota *EnterCriticalSection()*, jossa odotetaan synkronointiolion vapautumista muilta säikeiltä. Tämän jälkeen funktiosta palataan ja suoritusta jatketaan kriittisellä alueella. Alueelta poistuttaessa synkronointiolio vapautetaan *LeaveCriticalSection()*-funktiolla. Komponentin tietovarastoluokista kuten CGrabStatus ja CGlobals tehtiin säieturvallisia käyttämällä kriittisiä alueita luokan kaikissa tietoja käsittelevissä metodeissa. Koodiesimerkki 5 havainnollistaa kriittisten alueiden käyttöä.

```
int CGlobals::SetRunning(bool bNewValue)
{
  ::EnterCriticalSection(&m_cs);

  m_bRunning = bNewValue;

  ::LeaveCriticalSection(&m_cs);

  return 0;
}
```

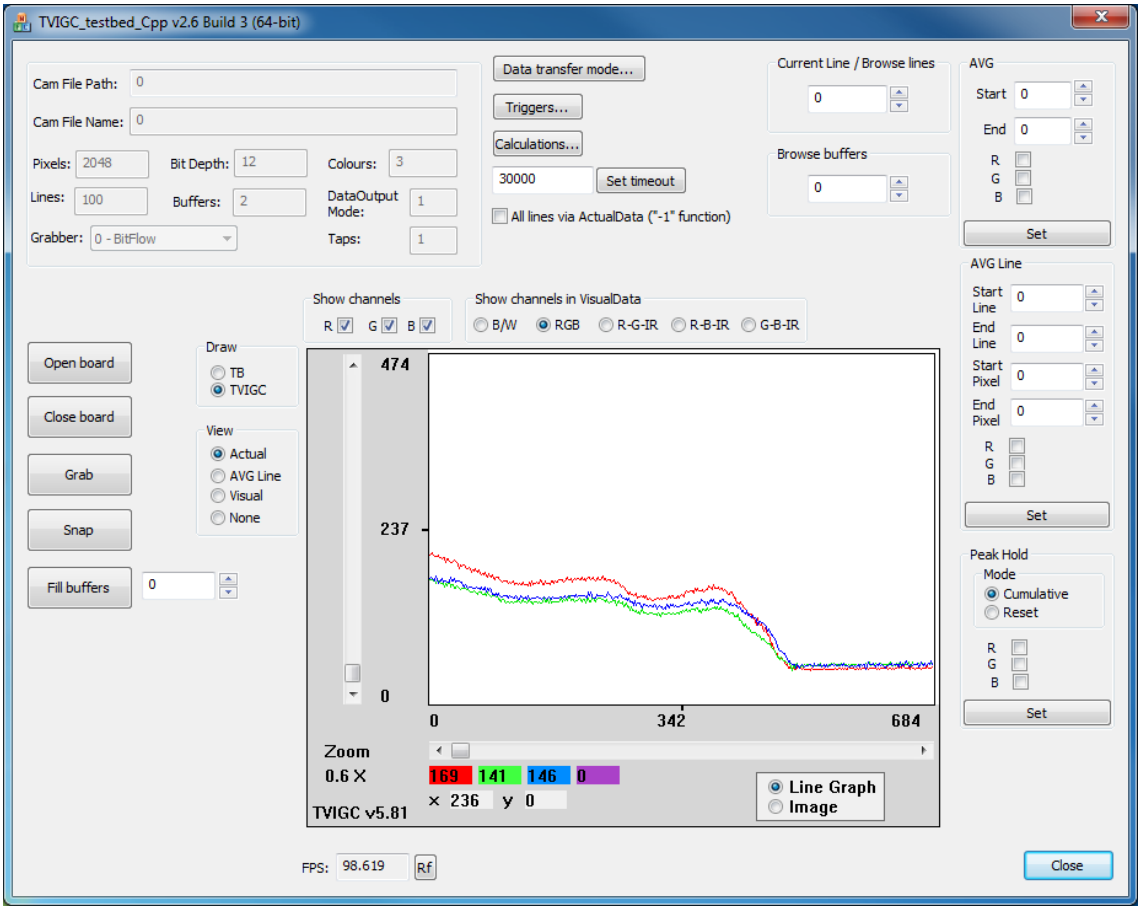
Koodiesimerkki 5. Säikeiden suoritusta ohjaavan lippumuuttujan asetus globaalin luokan CGlobals metodilla SetRunning() on suojattu kriittisellä alueella.

Komponenttiin lisätään tuki uudelle kuvankaappauskortille toteuttamalla uusi kaappauskorttikohtainen luokka (CGrabber1Operations) ja -säiefunktio (Grabber1_WaitFrameThread) sekä tarvittaessa kaappauskorttikohtainen globaali luokka (CGrabber1Globals), ja mahdollisesti kaappauskorttimallikohtaisia aliluokkia. Korttivalmistajan toimittaman ohjelmointirajapinnan DLL-komponentit linkitetään projektiin implisiittisesti käyttäen tuontikirjastoja (LIB-tiedostot). DLL:t ladataan dynaamisesti TVIGC-komponenttia käynnistettäessä. Jotta kaikkien komponentin tukemien kaappauskorttien ja niiden API-DLL-komponenttien ei tarvitsisi olla asennettuna samanaikaisesti komponenttia käyttävässä koneessa, käytetään viivästettyä DLL:ien latausteknikkaa, jossa DLL ladataan vasta sitten, kun sen jotakin funktiota ensimmäistä kertaa kutsutaan. Viivästetyksi halutut DLL:t määritellään kehitysympäristössä projektin ominaisuuksissa. [14.] Viivästetty lataus on käytännössä edellytys, jotta komponentista ei tarvitse toteuttaa rinnakkaisia versioita.

Komponentista käännetään sekä 32- että 64-bittiset versiot. Ne käännetään samasta lähdekoodista ja ovat bittisyyttä lukuun ottamatta muuten täysin identtisiä. 64-bittinen versio on ajettavissa vain 64-bittisessä Windows-järjestelmässä. Myös sekä asiakasohjelman että kuvankaappauskortin API:n on oltava 64-bittinen. 64-bittinen versio mahdollistaa laajemman muistiavaruuden käytön, joten tietokoneeseen voidaan asentaa enemmän keskusmuistia. Tämän myötä kaapattavat kuvat voivat olla suurempia ja kuvapuskureita voidaan määritellä enemmän. [15.]

5.5 Komponentin testaus

Komponentin ohjelmakoodin testausta varten rakennettiin testipetit sekä C++-kielellä että LabView:llä erityisesti moduulien ja yksittäisten toimintojen testaukseen sekä komponentin ja asiakasohjelman integroinnin testaukseen. Samoja testipetejä käytettiin apuna myös havaittujen virheiden jäljityksessä. Kuvassa 18 on C++-testipeti, johon on myös upotettu komponentin graafinen käyttöliittymä. Testipetillä voidaan kutsua ja asettaa kaikkia komponentin metodeja ja ominaisuuksia. Vasteita voidaan tarkastella visuaalisesti kuvassa näkyvinä muutoksina tai testidataa kirjoittaa tiedostoon. Komponentin yksittäisiä funktioita testattiin myös ajamalla komponenttia testipeteineen kääntäjän virheenjäljittimellä (debugger), jolla oli helppo edetä koodissa askel kerrallaan ja nähdä muuttujien ja parametrien saamia arvoja.



Kuva 18. TVIGC-komponenttia varten rakennettu testipeti. Keskelle on upotettu (tummanharmaa pohja) komponentin sisältämä käyttöliittymä, joka tarjoaa yksinkertaisen grafiikan piirtotoiminnon.

Koska komponentin kehitys ja erityisesti toimintojen toteuttaminen tapahtui asteittain, suoritettiin merkittävimpien väliversioiden kohdalla kattavampi järjestelmätestaus regressiotestauksena varsinaisella kameroiden testausohjelmistolla, jota varten komponentti ensisijaisesti kehitettiin. Tämä testaus painottui komponentin toimintoihin, mutta myös testausohjelmistoon liittyviä testitapauksia oli mukana. Testauksesta laadittiin määritelmien pohjalta testaussuunnitelma, jossa kuvattiin testimenetelmien lisäksi testitapaukset. Testaus toteutettiin etenemällä testisuunnitelmassa systemaattisesti kirjaten kuhunkin testitapaukseen ylös havaintoja sekä merkinnän testitapauksen hyväksymisestä tai hylkäämisestä. Täytetystä testisuunnitelmasta syntyi testiraportti, josta on ote taulukossa 3.

Taulukko 3. Ote komponentin ja kameratestausohjelmiston järjestelmätestausraportista.

Id	Kohde	Syöte	Odotettu tulos	Huom./*Saatu tulos*	OK?
110	**OpenBoard-parametrit**				
111	Syötetään kelvolliset parametrit	Aseta parametrit OpenBoard	Kaappaus käynnistyy ko. parametreilla.		OK
112	Syötetään epäkelpoja parametreja	Virheellinen pikselimäärä / bittisyvyys Liian suuri viiva-/puskuriluku jne. Ohjelma käynnissä, kaappaus pys. Painetaan OpenBoard. Yritetään useamman kerran peräkkäin.	Ohjelma antaa virheilmoituksen ja palaa “Closed”-tilaan.		OK
210	**Jatkuva kaappaus (Grab)**				
211	Kaappauksen käynnistys	Kaappaus pysäytettynä. Painetaan ”Grab”-nappia.	”Grab”-nappi muuttuu vihreäksi ja kaappaus käynnistyy.	Kokeile myös useamman kerran peräkkäin.	OK
212	Kaappauksen pysäytys	Kaappaus käynnissä. Painetaan ”Grab”-nappia.	”Grab”-nappi muuttuu punaiseksi ja kaappaus pysähtyy.		OK
250	**Toiminnot**				
251	Channels Enabled	Grab ON Muutetaan ko. parametreja	Poiskytketyt kanavat ovat pimeinä sekä Actual-, Visual-että AVGLine-kuvissa		OK
252	Calculations Enabled	Grab ON Muutetaan ko. parametreja	Poiskytkettyjä laskentoja ei suoriteta, eli kuvat eivät päivity.	*Testausohjelmassa ei mahdollista asettaa. C++-testipeti: OK*	(OK)
300	**Kuvat ja kaappauksen stop-tila**				
301	Grab stopin jälkeen viimeinen ruudulle päivittyvä kuva on oikeasti viimeiseksi kaapattu kuva	Grab ON Grab Stop Kokeile sekä lyhyillä että pitkillä kuvilla ja hitailla ja **nopeilla** viivataajuuksilla	Ok.		OK
310	ActualData, 8-bit	Kaappaus ja kamera 8-bit. Kaapataan kuva, jossa on suurta vaihtelua valotuksessa esim. testikuvio.	Viiva on 8-bittinen, ehjä, oikean pituinen ja pikselit ”oikeilla paikoillaan”, ym.	*Tutkittu kameran muodostamalla tunnetulla testikuviolla.*	OK

Käytännön testausta suoritettiin myös todellisissa tilanteissa antamalla järjestelmä yrityksen eri työntekijöiden käyttöön erilaisiin laitteistokokoonpanoihin. Testaajat suorittivat testausta mustalaatikkotestauksena käyttämällä järjestelmää ja sen eri toimintoja monipuolisesti kirjaten ylös havaintoja muun muassa virhetilanteista, virheilmoituksista, luotettavuudesta, käytettävyyteen liittyvistä asioista ja myös toimintojen puutteellisuuksista.

Testausta helpotti monelta osin merkittävästi se, että komponentista ei toteutettu rinnakkaisia versioita eli variaatioita esimerkiksi kutakin eri kuvankaappauskorttia varten. Kuvankaappauskortteihin liittyvät toiminnot jouduttiin kuitenkin testaamaan uudelleen jokaista korttia kohden. Komponentin 32- ja 64-bittisten käännösten oletettiin olevan niin identtisiä, että 32-bittiselle versiolle tehtiin vain kevyt järjestelmätestaus.

Komponentin reaaliaikajärjestelmäluonteesta johtuen järjestelmän toimintaa testattiin myös muiden ohjelmien voimakkaasti prosessoria ja muistia kuormittamassa laitteistossa, millä oletettiin olevan vaikutuksia erityisesti säikeiden ajoituksiin. Lisäksi seurattiin järjestelmän toimintaa pitkän aikavälin testissä, jossa järjestelmää ajettiin keskeytyksettä useita päiviä. Komponentin suorituskykyä arvioitiin vertaamalla sen laskemaa kuvankaappaustaajuuden arvoa (FPS) kaappauskortille tai kameralle asetettuun viivataajuuteen. Poikkeamia havaittiin riippuen viivataajuudesta, kuvan koosta, käytössä olevien ominaisuuksien määrästä, käytetystä asiakasohjelmasta ja laitteistosta.

5.6 Jatkokehitys

Komponentin jatkokehitys tulee olemaan ennen kaikkea päivitysten tekemistä uusien kameramallien myötä, jolloin muutoksia saatetaan tarvita uusia kaappausparametreja varten sekä uudessa järjestyksessä olevan kaapatun raakakuvadatan muuntamista varten. Komponentin kaappausrajapintaan voidaan lisätä tukia uusille kuvia tuottaville laitteille tai tekniikoille, useimmiten kuvankaappauskorteille. Näiden lisäksi voidaan lisätä uusia toimintoja, kuten kuvadatalle tehtäviä käsittelyjä ja laskentoja. Esimerkiksi suunnitteilla on toiminto, jonka avulla komponentin muodostamia kuvadatoja voi tallentaa tiedostoon. Käyttöliittymän piirtokoodi on tarkoitus siirtää omaan moduuliinsa, jolloin sitä voisi hyödyntää myös muissa projekteissa.

Komponentin tehokkuuden parantamiseksi voidaan tutkia uusien tekniikoiden hyödyntämistä, kuten GPGPU-laskentaa (General-purpose computing on graphics processing units), jossa näytönohjaimien laskentatehoa voidaan hyödyntää yleiskäyttöisesti muunkin datan kuin näytölle piirrettävän grafiikan laskentaan. GPU-laskenta sopii erityisesti kohteisiin, joissa voidaan käyttää rinnakkaislaskentaa. Suosituimpia ohjelmointikieliä GPU-laskennan toteuttamiseen ovat opencl ja Nvidian CUDA. [16.]

6 Yhteenveto

Insinöörityön tavoitteena oli suunnitella ja toteuttaa Windows-ympäristössä käytettävä kuvankaappausrajapinta digitaalisia viivakameroita valmistavalle JAI Oy:lle kehitettävänä ollutta uutta kameroiden testausjärjestelmää varten. Rajapinta toteutettiin ActiveX-ohjelmistokomponenttina, joka voitiin integroida testausjärjestelmään tai muuhun kuvankaappauspalveluita tarvitsevaan sovellukseen. Tavoitteina komponentin toteutukselle olivat testausjärjestelmää varten valitun PCIExpress-väyläisen kuvankaappauskortin ohjaus ja kameralta kortin välityksellä saatavan kuvadatan esikäsittely sekä datan tarjoaminen testausohjelmalle. Komponentin tuli suoriutua kuvankaappauksesta ja -käsittelystä hallitusti silloinkin, kun käsittely kestää kaappausta kauemmin. Lisäksi tavoitteena oli suunnitella komponentti modulaariseksi siten, että uusien kuvankaappauskorttimallien tai vastaavien laitteistojen sekä kuvankäsittelyoperaatioiden lisääminen olisi jatkossa helppoa. Komponentilta odotettiin myös hyvää suorituskykyä ja helppokäyttöisyyttä.

Komponentin toteutuksen haasteellisimmaksi vaiheeksi osoittautui kuvankaappauksen tahdistuksen toteutus säikeistystä käyttäen. Rinnakkain suoritettavien säikeiden synkronoinnin toimimaan saaminen edellytti määrittelyjen korjaamista ja tarkentamista moneen otteeseen. Lopulta siinä kuitenkin onnistuttiin ainakin riittävällä tasolla, mistä osoituksena oli noin viikon yhtämittaisen testijakson ajan moitteettomasti toiminut järjestelmä. Rinnakkaisohjelmointiin perehtyminen oli hyödyllistä myös tulevia projekteja ajatellen, sillä nykyinen trendi prosessoreiden laskentatehon lisäämiseksi prosessoriytimien määrää kasvattamalla tullee lisäämään myös rinnakkaisohjelmoinnin merkitystä.

Komponentista saatiin arkkitehtuuriltaan suhteellisen toimiva ja selkeä, mikä helpottaa rakenteen hahmottamista, ylläpitoa ja uusien toimintojen lisäämistä. Toisaalta arkkitehtuuri sisältää paljon sekä olio- että sanomarajapintoja, mikä on luultavasti ainakin

osasyynä siihen, että suorituskyvyssä ei kaikissa tilanteissa päästy aivan reaaliaikaiselle tasolle. Tätä voisi pyrkiä jatkokehityksen yhteydessä optimoimaan. Näissäkin tilanteissa hallittu toiminta varmistettiin käyttämällä ohjelmallista liipaisua uuden kuvan kaappauksen aloittamiseen.

Yritykselle suunnitellun testausjärjestelmän lisäksi komponentti on käytössä jo muissakin kuvankaappausta tarvitsevissa projekteissa. Kokemuksen perusteella voidaan todeta, että asiakasohjelmarajapinta saatiin suunniteltua kohtalaisen yksinkertaiseksi ja geneeriseksi niin, että se abstrahoi käytetyn kuvankaappauslaitteiston varsin hyvin. Komponenttiin on lisätty tuki jo useammalle kuvankaappauskortille, mutta asiakasohjelmalle tämä näkyy parhaimmillaan vain yhden parametrin muuttamisena. Komponentin integroiminen asiakasohjelmaan vaatii käyttöohjeen lukemista, mutta on yleisesti ottaen varsin helppo ja nopea toimenpide.

Testausjärjestelmä TVIGC-kuvankaappauskomponentin kanssa on ollut JAI Oy:ssä käytössä jo jonkin aikaa, ja siitä on muodostunut yksi keskeisimmistä ohjelmistotyökaluista yrityksessä.

Lähteet

1 Digikameroiden kennotyyppien eroja. 2003. Verkkodokumentti. DigiFAQ. <http://digifaq.info/digi_omat/kennot.html>. 10.9.2003. Luettu 24.4.2015.

2 Digikameran dynamiikka. 2003. Verkkodokumentti. DigiFAQ. <http://www.digifaq.info/digi_omat/dynamiikka/>. 28.11.2003. Luettu 24.4.2015.

3 Digital camera. 2015. Verkkodokumentti. Wikipedia. <http://en.wikipedia.org/wiki/Digital_camera#Sensor_resolution>. 18.4.2015. Luettu 24.4.2015.

4 Three-CCD camera. 2015. Verkkodokumentti. Wikipedia. <http://en.wikipedia.org/wiki/Three-CCD_camera>. 24.3.2015. Luettu 24.4.2015.

5 ADR Technology. 2015. Verkkodokumentti. Euresys. <http://www.euresys.com/Products/grablink/ADRTechnology.asp>. Luettu 24.4.2015.

6 Machine vision. 2015. Verkkodokumentti. Wikipedia. <http://en.wikipedia.org/wiki/Machine_vision>. 20.3.2015. Luettu 24.4.2015.

7 Software Development Kit. 2015. Verkkodokumentti. BitFlow. <http://www.bitflow.com/products/details/software-development-kit>. Luettu 24.4.2015.

8 Microsoft (toim.) & Rautiainen, Juha (käänt.). 2000. MCSD Training Kit Visual C++ 6.0. Helsinki: Edita.

9 Windows API. 2015. Verkkodokumentti. Wikipedia. <http://en.wikipedia.org/wiki/Windows_API>. 11.4.2015. Luettu 24.4.2015.

10 ActiveX. 2013. Verkkodokumentti. Wikipedia. <http://fi.wikipedia.org/wiki/ActiveX>. 9.3.2013. Luettu 24.4.2015.

11 LabView. 2015. Verkkodokumentti. Wikipedia. <http://en.wikipedia.org/wiki/LabVIEW>. 22.4.2015. Luettu 24.4.2015.

12 Haikala, Ilkka, Märijärvi, Jukka. 2006. Ohjelmistotuotanto. Helsinki: Talentum.

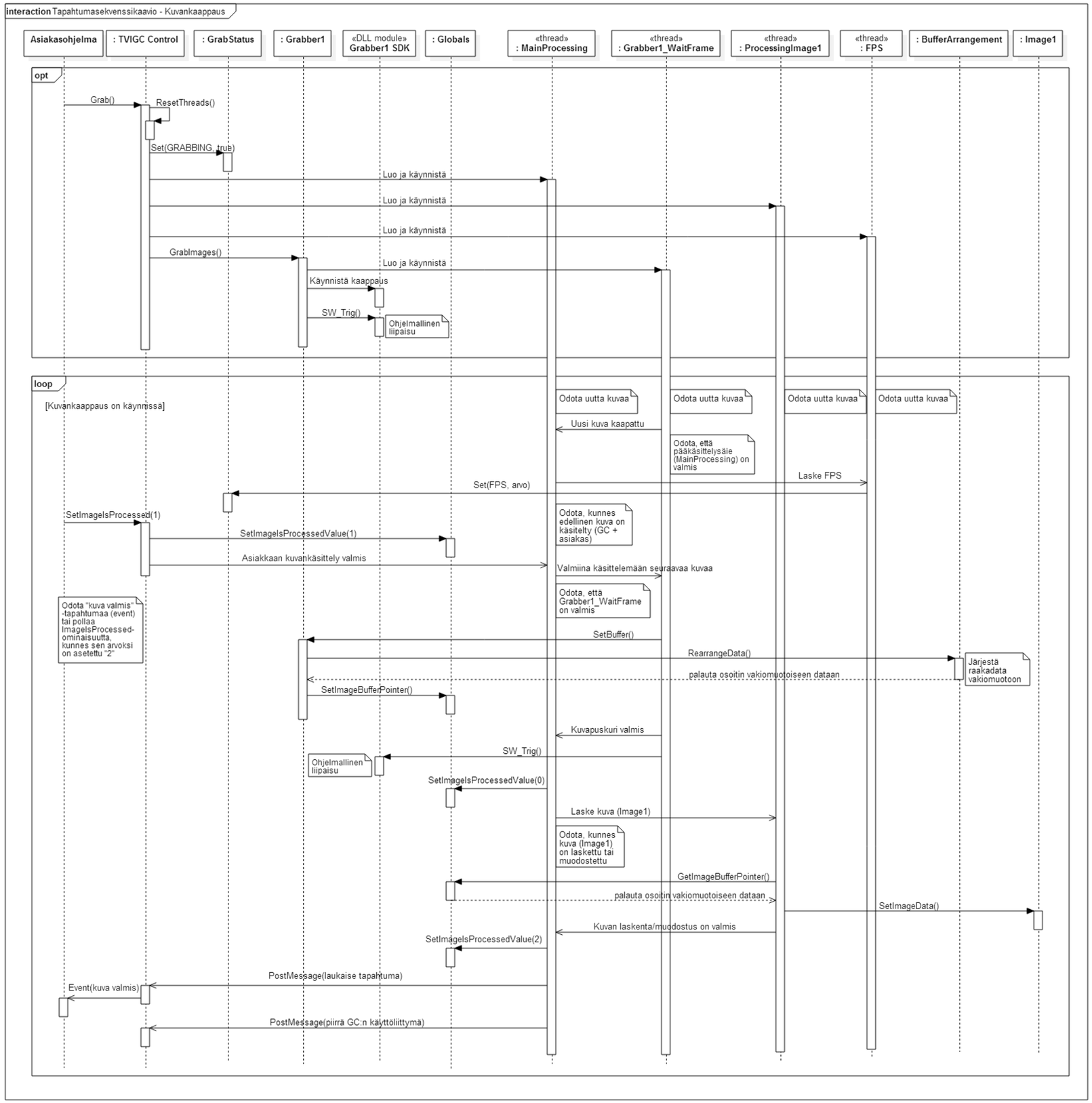
13 WaitForSingleObject function. 2015. Verkkodokumentti. Microsoft. <https://msdn.microsoft.com/en-us/library/windows/desktop/ms687032%28v=vs.85%29.aspx>. Luettu 24.4.2015.

14 Linker Support for Delay-Loaded DLLs. 2015. Verkkodokumentti. Microsoft. <https://msdn.microsoft.com/en-us/library/151kt790.aspx>. Luettu 24.4.2015.

15 64-bit computing. 2015. Verkkodokumentti. Wikipedia. <http://en.wikipedia.org/wiki/64-bit_computing>. 17.4.2015. Luettu 24.4.2015.

16 Saarelainen, Ari. 2014. Valjasta näytönohjain teholaskentaan. Tivi syyskuu 2014, s. 50.

TVIGC-komponentin kuvankaappaussekvenssi



TVIGC-komponentin ohjelmointirajapinta

Ominaisuudet

Nimi	ActualData
Tietotyyppi	VARIANT / unsigned long
Sisältö	Viivadata

Nimi	AVGLineData
Tietotyyppi	VARIANT / unsigned long
Sisältö	Pikseleiden keskiarvodata

Nimi	AVGLineParameters		
Tietotyyppi	VARIANT / long (taulukko)		
Sisältö	AVGLineDatan muodostuksen ohjausparametrit		
Parametrit	**Indeksi**	**Kuvaus**	**Arvot**
	0	Ensimmäinen laskentaan mukaan otettava viiva	0...(viivojen_lkm_yht. - 1)
	1	Viimeinen laskentaan mukaan otettava viiva	0...(viivojen_lkm_yht. - 1), oltava ≥ [Indeksi 0:n arvo]
	2	Ensimmäinen laskentaan mukaan otettava pikseli	0...(sensorin_pituus - 1)
	3	Viimeinen laskentaan mukaan otettava pikseli	0...(sensorin_pituus - 1), oltava ≥ [Indeksi 2:n arvo]
	4	Käytössä R-kanavalla	0 = false, 1 = true
	5	Käytössä G-kanavalla	0 = false, 1 = true
	6	Käytössä B-kanavalla	0 = false, 1 = true
	7	Käytössä IR-kanavalla	0 = false, 1 = true
	8...35	Käytössä X-kanavalla	0 = false, 1 = true

Nimi	AVGParameters		
Tietotyyppi	VARIANT / long (taulukko)		
Sisältö	ActualDatan muodostuksen ohjausparametrit		
Parametrit	**Indeksi**	**Kuvaus**	**Arvot**
	0	Ensimmäinen laskentaan mukaan otettava viiva	0...(viivojen_lkm_yht. - 1)
	1	Viimeinen laskentaan mukaan otettava viiva	0...(viivojen_lkm_yht. - 1), oltava ≥ [Indeksi 0:n arvo]
	2	Käytössä R-kanavalla	0 = false, 1 = true
	3	Käytössä G-kanavalla	0 = false, 1 = true
	4	Käytössä B-kanavalla	0 = false, 1 = true
	5	Käytössä IR-kanavalla	0 = false, 1 = true
	6...33	Käytössä X-kanavalla	0 = false, 1 = true

Nimi	CalculationsEnabled		
Tietotyyppi	unsigned long		
Sisältö	Liput muodostettavien tai laskettavien kuvadatojen valintaa varten		
Parametrit	**Bitti**	**Bitin tilat**	**Laskettava data**
	0 (LSB)	0 = laskenta pois käytöstä 1 = laskenta käytössä	ActualData
	1	"	VisualData
	2	"	AVGLineData
	3	"	PeakHoldData
	4...31	"	Varattu tulev.

Nimi	ChannelsEnabled		
Tietotyyppi	unsigned long		
Sisältö	Liput värikanavien valintaa varten		
Parametrit	**Bitti**	**Bitin tilat**	**Värikanava**
	0 (LSB)	0 = laskenta pois käytöstä 1 = laskenta käytössä	R
	1	"	G
	2	"	B
	3	"	IR
	4...31	"	Varattu tulev.

Nimi	CurrentLine	
Tietotyyppi	long	
Sisältö	Valitun viivan numero	
Parametrit	**Arvo**	**Kuvaus**
	-1	Kaikki viivat kopioidaan ActualData-ominaisuuteen
	0...*	Käsiteltävän viivan nro kuvan alusta lähtien. Indeksointi alkaa nollasta.

Nimi	DrawingEnabled	
Tietotyyppi	long	
Sisältö	TVIGC:n käyttöliittymään piirrettävän datan valinta	
Parametrit	**Arvo**	**Piirrettävä data**
	0	Vain TVIGC:n versionumero
	1	ActualData ja myös PeakHoldData, mikäli kytketty päälle
	2	VisualData
	3	AVGLineData

Nimi	ErrorMode	
Tietotyyppi	long	
Sisältö	Virheilmoitusten tiedotustavan valinta	
Parametrit	**Arvo**	**Kuvaus**
	0	Asiakasohjelmalle lähetetään tapahtumien kautta virhekoodi ja virheilmoitus näytetään TVIGC:n avaamassa modaalittomassa loki-dialogissa.
	1	Pelkästään virhekoodi lähetetään asiakasohjelmalle.
	2	Minkäänlaista virhetiedotusta ei anneta.
	3	TVIGC generoi modaalisen virheilmoitus-dialogin, lähettää virhekoodin asiakasohjelmalle ja näyttää virheilmoituksen loki-dialogissa.
	4	Virhekoodi lähetetään asiakasohjelmalle sekä generoidaan modaalinen virheilmoitus-dialogi.

Nimi	EventsEnabled	
Tietotyyppi	long	
Sisältö	Valmiin kuvan tiedotustavan valinta	
Parametrit	**Arvo**	**Kuvaus**
	0	TVIGC ei laukaise tapahtumaa kuvan valmistuttua
	1	TVIGC laukaisee tapahtuman kuvan valmistuttua

Nimi	GrabStatus	
Tietotyyppi	VARIANT / long (taulukko)	
Sisältö	Kaappausparametritaulukko	
Parametrit	**Indeksi**	**Kuvaus**
	0	Kaappaus käynnissä/pysäytettynä
	1	Viivojen lukumäärä
	2	Viivan pituus (pikseleiden lukumäärä)
	3	Bittisyvyys
	4	Viivan liipaisun toimintatila
	5	Viivan liipaisusignaalin aktiivinen reuna
	6	Viivan liipaisusignaalin tyyppi
	7	Viivan liipaisusignaalin lähde
	8	Kuvan liipaisun toimintatila
	9	Kuvan liipaisusignaalin aktiivinen reuna
	10	Kuvan liipaisusignaalin tyyppi
	11	Kuvan liipaisusignaalin lähde
	12	Kuvankaappaustaajuus (FPS)
	13	TVIGC:n versio
	14	Aikakatkaisun yläraja millisekunteina (timeout)
	15	Raakakuvapuskureiden lukumäärä
	16	Datan ulostulotapa
	17	Kaappaus alustettu/alustamatta
	18	Värikanavien lukumäärä
	19	Rinnakkain ulostulevien pikseleiden lukumäärä
	20	Valittu kuvankaappauskortti

Nimi	ImageIsProcessed	
Tietotyyppi	long	
Sisältö	Valmiista kuvasta ja käsittelystä ilmaiseva lippu	
Parametrit	**Arvo**	**Kuvaus**
	0	Alkuarvo, kun kaappaus alkaa tai aina kun TVIGC:n käsittely alkaa
	1	Asiakasohjelma asettaa kuvan käsittelyn valmistuttua
	2	TVIGC asettaa kuvan valmistuttua

Nimi	PeakHoldData
Tietotyyppi	VARIANT / unsigned long
Sisältö	Pikselikohtaisten ääriarvojen seurantadata

Nimi	PeakHoldParameters		
Tietotyyppi	VARIANT / long (taulukko)		
Sisältö	PeakHoldDatan muodostuksen ohjausparametrit		
Parametrit	**Indeksi**	**Kuvaus**	**Arvot**
	0	Toimintatila	0 = Kumulatiivinen; data nollataan toimintatilaa vaihdettaessa tai yksittäisen kanavan osalta kanava pois kytkettäessä 1 = Nollaus automaattisesti joka kuvan välissä
	1	Käytössä R-kanavalla	0 = false, 1 = true
	2	Käytössä G-kanavalla	0 = false, 1 = true
	3	Käytössä B-kanavalla	0 = false, 1 = true
	4	Käytössä IR-kanavalla	0 = false, 1 = true
	5...32	Käytössä X-kanavalla	0 = false, 1 = true

Nimi	VisualData
Tietotyyppi	VARIANT / unsigned long
Sisältö	Bittikarttakuvadata

Nimi	VisualParameters		
Tietotyyppi	VARIANT / long (taulukko)		
Sisältö	ActualDatan muodostuksen ohjausparametrit		
Parametrit	**Indeksi**	**Kuvaus**	**Arvot**
	0	R-kanava näytetään	0 = false, 1 = true
	1	G-kanava näytetään	0 = false, 1 = true
	2	B-kanava näytetään	0 = false, 1 = true
	3	IR-kanava näytetään	0 = false, 1 = true
	4...31	X-kanava näytetään	0 = false, 1 = true

Metodit

Nimi	CloseBoard
Kuvaus	Vapauttaa resurssit
Paluuarvo	void
Parametrit	-

Nimi	GetData			
Kuvaus	VARIANT-tyyppisten ominaisuuksien haku osoittimen avulla			
Paluuarvo	void			
Parametrit	LONG DataIndex			
	Arvo	**Ominaisuus**	**Alkion tietotyyppi**	**Alkioiden lukumäärä**
	0	GrabStatus	ULONG (32-bit)	21
	1	ActualData	"	CurrentLine ≥ 0: [pikselit] * [värit] CurrentLine = -1: [pikselit] * [viivat] * [värit]
	2	AVGParameters	"	2 + [värit]
	3	VisualData	"	[pikselit] * [viivat]
	4	AVGLineData	"	[viivat] * [värit]
	5	AVGLineParameters	"	4 + [värit]
	6	PeakHoldData	"	[pikselit] * [värit] * 2
	7	PeakHoldParameters	"	1 + [värit]
	8	VisualParameters	"	[värit]
	ULONG DataPointer		Osoitin, johon DataIndex-parametrilla valittu data halutaan kopioitavan.	

Nimi	Grab	
Kuvaus	Aloittaa/pysäyttää kuvankaappauksen	
Paluuarvo	void	
Parametrit	LONG Start	0 = Pysäyttää kaappauksen. Odottaa, kunnes kaikki kaapatut kuvat on kuitattu luetuiksi. 1 = Käynnistää kaappauksen. 2 = Pysäyttää kaappauksen, eikä jää odottamaan puskurien tyhjentämistä

Nimi	LoadBuffer	
Kuvaus	Kuvapuskureiden selaus	
Paluuarvo	void	
Parametrit	LONG BufferNumber	Puskuri-indeksi. Viimeiseksi kaapatun puskurin indeksi on [puskurien_lkm]-1. Vanhin puskuri on indeksiltään 0, paitsi jos kuvia on kaapattu vähemmän kuin [puskurien_lkm].

Nimi	OpenBoard	
Kuvaus	Alustaa kaappauksen	
Paluuarvo	void	
Parametrit	BSTR CfgFilePath	Konfigurointitiedoston absoluuttinen hakemistopolku.
	BSTR CfgFileName	Konfigurointitiedoston nimi.
	LONG LineWidth	Kameran sensorin pikseleiden lukumäärä.
	LONG NumOfLines	Kaapattavan kuvan viivojen lukumäärä.
	LONG BitDepth	Kaapattavan kuvan bittisyvyys.
	LONG DataOutputMode	Datan ulostulotapa 1 = Rinnakkain (Parallel) 2 = Sarjassa (Multiplexed)
	LONG NumOfBuffers	Kuvapuskureiden lukumäärä, väh. 2.
	LONG NumOfColourChannels	Värikanavien lukumäärä: 1...32
	LONG NumOfTaps	Rinnakkain ulostulevien pikseleiden lukumäärä: 1...*
	LONG Grabber	Käytettävä kaappausrajapinta: 0 = BitFlow (CameraLink) 1 = TVI Priimus (Ethernet) 2 = Active Silicon (CoaXPress)

Nimi	SetAVGLineParameters	
Kuvaus	AVGLineParameters-ominaisuuden asettaminen metodin avulla	
Paluuarvo	void	
Parametrit	UNSIGNED LONG StartLine	0…(viivojen_ lkm_yht. - 1)
	UNSIGNED LONG EndLine	0…(viivojen_lkm_yht. - 1), oltava ≥ StartLine
	UNSIGNED LONG StartPixel	0…(sensorin_pituus - 1)
	UNSIGNED LONG EndPixel	0…(sensorin_pituus - 1), oltava ≥ StartLine
	UNSIGNED LONG EnableR	0 = false 1 = true
	UNSIGNED LONG EnableG	0 = false 1 = true
	UNSIGNED LONG EnableB	0 = false 1 = true
	UNSIGNED LONG EnableIR	0 = false 1 = true

Nimi	SetAVGParameters	
Kuvaus	AVGParameters-ominaisuuden asettaminen metodin avulla	
Paluuarvo	void	
Parametrit	UNSIGNED LONG StartLine	0…(viivojen_lkm_yht. - 1)
	UNSIGNED LONG EndLine	0…(viivojen_lkm_yht. - 1), oltava ≥ StartLine
	UNSIGNED LONG EnableR	0 = false 1 = true
	UNSIGNED LONG EnableG	0 = false 1 = true
	UNSIGNED LONG EnableB	0 = false 1 = true
	UNSIGNED LONG EnableIR	0 = false 1 = true

Nimi	SetPeakHoldParameters	
Kuvaus	PeakHoldParameters-ominaisuuden asettaminen metodin avulla	
Paluuarvo	void	
Parametrit	UNSIGNED LONG Mode	0 = Kumulatiivinen 1 = Nollaus kuvien välissä
	UNSIGNED LONG EnableR	0 = false 1 = true
	UNSIGNED LONG EnableG	0 = false 1 = true
	UNSIGNED LONG EnableB	0 = false 1 = true
	UNSIGNED LONG EnableIR	0 = false 1 = true

Nimi	SetTimeout	
Kuvaus	Aikakatkaisun ylärajan asettaminen	
Paluuarvo	void	
Parametrit	ULONG Timeout	Raja-arvo millisekunteina 0 = Rajaton

Nimi	SetTriggers	
Kuvaus	Liipaisutilan asettaminen	
Paluuarvo	void	
Parametrit	LONG HEncMode	Viivan liipaisun toimintatila (Horizontal encoder, line level) 0 = Jatkuva (Free run) 1 = Liipaisu (One shot, wait for encoder)
	LONG HEncPolar	Viivan liipaisusignaalin aktiivinen reuna 0 = Nouseva reuna 1 = Laskeva reuna
	LONG HEncInput	Viivan liipaisusignaalin tyyppi 0 = Differential 1 = TTL 2 = Opto-coupled
	LONG HEncExt	Viivan liipaisusignaalin lähde 0 = Sisäinen 1 = Ulkoinen
	LONG VTrigMode	Kuvan liipaisun toimintatila (Vertical trigger, frame level): 0 = Jatkuva (Free run) 1 = Liipaisu, jokainen kuva (Edge mode, one frame per trigger) 2 = Liipaisu, jatkuvan kaapp. aloitus (Level mode, continuous) 3 = ClockWork (varmistaa, että käytössä on ohjelmalliseen liipaisuun sopiva tila, jota TVIGC:n kaappauskoneisto pääasiassa tarvitsee)
	LONG VTrigPolar	Kuvan liipaisusignaalin aktiivinen reuna 0 = Nouseva reuna 1 = Laskeva reuna
	LONG VTrigInput	Kuvan liipaisusignaalin tyyppi 0 == Differential 1 == TTL 2 == Opto-coupled
	LONG VTrigExt	Kuvan liipaisusignaalin lähde 0 = Sisäinen 1 = Ulkoinen

Nimi	SetVisualParameters	
Kuvaus	VisualParameters-ominaisuuden asettaminen metodin avulla	
Paluuarvo	void	
Parametrit	UNSIGNED LONG ShowColours	0 = mustavalkokuva 1 = R-G-B 2 = R-G-IR 3 = R-B-IR 4 = G-B-IR

Nimi	Snap
Kuvaus	Kaappaa yksittäisen kuvan
Paluuarvo	void
Parametrit	-

Tapahtumat

Nimi	Event	
Kuvaus	TVIGC:n laukaisema tapahtuma, jonka asiakasohjelma vastaanottaa. Mukana toimitetaan koodi parametrin välityksellä.	
Parametrit	0	Valmis, kaapattu ja käsitelty kuva odottamassa TVIGC:ssä.
	1000	Ok-kuittaus (ei virhettä)
	1001…*	Virhekoodi