Digitaalinen ja reaaliaikainen menetelmä kameran ja näyttölaitteen värikalibrointiin

Petri Piirainen

18.6.2004

Joensuun yliopisto
Tietojenkäsittelytiede
Pro gradu -tutkielma

Tiivistelmä

Työssä on kehitetty tekniikka, jonka avulla voidaan toteuttaa reaaliaikainen värikalibrointi ccd-kameralle ja näyttölaitteelle. Reaaliaikaisuudella tarkoitetaan sitä, että suurin viive ruutujen välillä videokuvassa on 30 ms. Lisäksi algoritmien toteutus on sellainen, että niitä voidaan käyttää monissa erilaisissa laitteistoissa. Tutkielman pääpainona on värikalibrointialgoritmien optimoiminen. Algoritmien optimoinnissa käytetään modernien x86-arkkitehtuurin prosessorien multimediaominaisuuksia, jotka soveltuvat hyvin väritutkimuksen sovelluksiin. Lisäksi tutkielmassa esitellään reaaliaikaisen kuvausjärjestelmän Linux-käyttöjärjestelmässä toimiva prototyyppi.

ACM-luokat (ACM Computing Classification System, 1998 version): B.8, D.4.8,I.4

Avainsanat: kuvanotto, reaaliaikaisuus, digitaalinen, värikalibrointi, MMX, optimointi

Esipuhe

Haluan esittää suuret kiitokset työni ohjaajalle, InFotonics Center Joensuun johtajalle, FT Markku Hauta-Kasarille ohjauksesta ja mielenkiintoisesta projektista, sillä tämä projekti on tarjonnut todella paljon työtä tuotekehityksen parissa. Lisäksi haluan kiittää myös FT Raimo Silvennoista, joka on ollut mukana hankkeessa. Lopuksi haluan kiittää äitiä ja isää siitä tuesta, jota he ovat tarjonneet opintoihini.

Joensuussa 18.6.2004

Petri Piirainen

Sisältö

Määritelmät

SDL = Simple DirectMedia Layer. On kirjasto joka tarjoaa pääsyn esim. tietokoneen näytönohjaimeen. SDL dokumentaatio `http://sdldoc.csn.ul.ie/`, kotisivu `http://www.libsdl.org`

V4L = Video for Linux. On Linuxin ytimessä oleva rajapinta, jonka avulla päästään käyttämään esim. tietokoneesen liitettyjä kameroita tai muita videolähteitä. `http://linux.bytesex.org/v4l2/`

GTK = The GIMP Toolkit. Työkalu ikkunaohjelmien tekemiseen X Window-järjestelmään. Dokumentaatio `http://www.gtk.org/api/`, kotisivu `http://www.gtk.org/`

RGB = Värikoordinaatisto, jossa yksi väri koostuu kolmesta komponentista: punainen (R), vihreä (G) ja sininen (B).

SDL-piirtopinta = Englanniksi SDL_Suface. On SDL-rajapinnan luoma komponentti, jossa ruudulle piirrettävä kuvadata säilytetään.

SIMD = Single Instruction Multiple Data, tarkoittaa tekniikkaa jossa yhden operaation aikana käsitellään useaa dataa.

MMX =Multimedia Extensions. Intelin kehittämä SIMD käskykanta kokonaisluvuille.

1 Johdanto

Monissa sovelluksissa, jotka käyttävät digitaalista videokuvausta, olisi usein tärkeää saada mahdollisimman tarkka väriesitys kohteesta. Spektristä väritutkimusta hyödyntäen on mahdollista korjata videoesitys erittäin tarkaksi väriesityksen kannalta. Kuitenkin ongelmana on laskennasta aiheutuva viive, koska väriesityksen korjaaminen vaatii runsaasti laskutoimituksia. Esimerkiksi lääketieteellisissä sovelluksissa videokuvan pitää olla reaaliaikaista, eli kuvausnopeuden tulee olla 30 ruutua sekunnissa. Viiveen poistaminen voidaan toteuttaa käyttämällä tehokkaampaa laitteistoa tai heikentämällä väriesityksen korjausta. Monissa sovelluksissa väriesitys onkin puutteellinen, koska muuten laitteistokustannukset olisivat liian kalliit. Esimerkiksi mobiilisovelluksissa laitteiston tehokkuutta joudutaan rajoittamaan virransäästön ja tilankäytön takia, kuitenkin esimerkiksi kamerakännykän kuvan laatua, voidaan parantaa huomattavasti, kun käytetään värikalibrointia. Tällöin joudutaan hakemaan kompromissia korjauksen tarkkuuden ja laitteistovaatimusten väliltä.

Tässä tutkielmassa esitellään menetelmä digitaaliseen ja reaaliaikaiseen kameran ja näyttölaitteen värikalibrointiin. Lähtökohtana oli kehittää menetelmä, joka soveltuisi käyttäväksi tavallisessa PC-laitteistossa ja olisi myös mahdollista siirtää käytettäväksi sulautettuihin järjestelmiin. Koska reaaliaikaisuuden vaatimus on se, että kuvausnopeus on vähintään 30 ruutua sekunnissa, tällöin värikalibroinnin suorittaminen saa kestää enintään 1 millisekunnin. Värikalibroinnin lähtökohdaksi otettiin se, että värikalibrointi suoritetaan RGB-avaruudessa. Värikalibroinnissa tarvittavat referenssiarvot muodostetaan spektridatan perusteella [10]. Menetelmässä on tärkeää myös se, että kalibrointidatan mittaaminen on yksinkertaista. Tässä menetelmässä käytetään kalibroinnissa mallina kameran ja näyttölaitteen systeemifunktion vaikutuksen minimoinnista [4]. Systeemifunktiolla tarkoitetaan sitä poikkeamaa, jonka järjestelmä aiheuttaa käsiteltävään dataan ideaaliseen järjestelmään verrattuna. Kameran tapauksessa systeemifunktion määrittäminen tapahtuu siten, että kuvataan kameralla värireferenssejä, joista mitataan myös spektridata. Systeemifunktio saadaan selville kun lasketaan spektridatasta laskettujen tarkkojen RGB-väriarvojen ja kameran esittämien väriarvojen erotus. Systeemifunktio selvitetään koko kameran dynamiikkaa käyttäen, eli jokaiselle RGB-komponentille lasketaan 255 arvoa. Näyttölaitteen systeemifunktion määrittämiseen käytetään samantyylistä menetelmää, siinä muodostetaan näytölle koko dynamiikan kattava RGB-esitykset ja mitataan todellinen RGB-esitys näytöstä ja laske-

taan näiden erotus [11]. Reaaliaikainen kalibrointi saadaan aikaiseksi kun taulukoidaan systeemifunktion arvot, jolloin värikalibrointi voidaan suorittaa pelkästään yhteenlaskuoperaatioiden avulla. Tällöin algoritmi on helppo optimoida monille laitteistoalustoille. Tällä värikalibrointitavalla saavutetaan mahdollisimman tarkka väriesitys RGB-väriavaruutta käyttäen.

Luvussa 2 käsitellään värikalibrointiin liittyvää signaalinkäsittelyä, sekä esitellään systeemifunktion määrittämiseen perustuva kalibrointimenetelmä. Luvussa 3 esitellään värikalibroinnin kannalta tärkeät väriavaruudet ja niiden ominaisuudet. Luvussa 4 esitellään menetelmä kameran systeemifunktion määrittämiseen ja esitellään kameran reaaliaikainen värikalibrointialgoritmi, lisäksi esitellään menetelmät kalibrointidatan muodostamiseen. Luvussa 5 esitellään näyttölaitteen reaaliaikainen värikalibrointimenetelmä, johon liittyvät näyttölaitteen systeemifunktion määrittäminen ja kalibrointidatan laskeminen. Luvussa 6 esitellään värikalibrointialgoritmien optimointimenetelmiä. Näistä tärkein on PC-prosessorien MMX-teknologian hyödyntäminen, MMX-teknologia tehostaa suorista noin kolminkertaiseksi. Vaikka MMX-teknologia on varsin vanha, niin sillä tehdyt ohjelmat toimivat sekä AMD:n että Intelin prosessoreissa. Luvussa 7 esitellään värikalibrointimenetelmää varten kehitetyn prototyypin toteutusta ja toimintaa. Tutkielmassa kehitetyn kalibrointimenetelmän tehokkuutta pohditaan luvussa 8 ja luvussa 9 esitetään tutkielman yhteenveto.

2 Järjestelmän kalibrointi

Järjestelmän kalibroinnissa on tavoitteena saavuttaa mahdollisimman virheetön ulostulo, eli pyritään minimoimaan järjestelmän sisäänmenosignaalille aiheutuneet virheet. Tässä tutkielmassa pyritään lisäksi minimoimaan järjestelmästä aiheutuvat viiveet, joten kalibroinnissa joudutaan hakemaan kompromissi virheen ja viiveen minimoinnin väliltä.

2.1 Konvoluutio ja dekonvoluutio

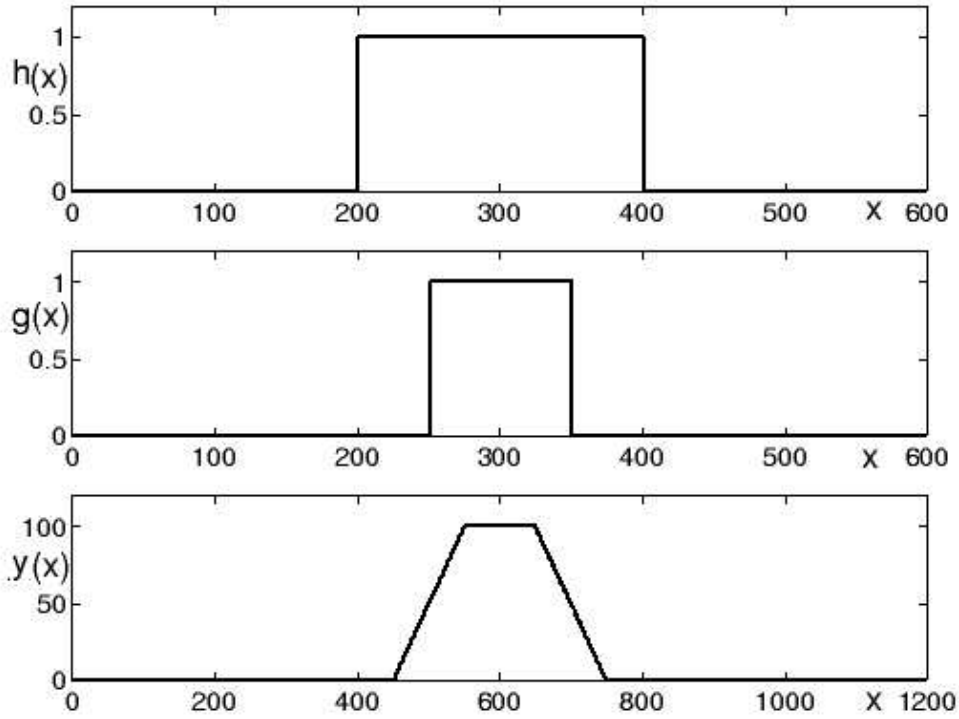
Signaalinkäsittelyssä käytetään konvoluutiota mallintamaan järjestelmässä tapahtuvaa virhettä. Konvoluutiossa järjestelmästä ulostuleva signaali on sisäänmenosignaalin ja järjestelmän systeemifunktion konvoluutio. Konvoluutiota käsitellään taajuustasossa. Kun järjestelmän systeemifunktio saadaan määritettyä taajuustasossa voidaan alkuperäinen signaali palauttaa [1]. Konvoluutio määritellään diskreettinä tapauksena seuraavasti, y on ulostulosignaali, h on sisäänmenosignaali ja g on järjestelmän systeemifunktio, x tarkoittaa diskreettiä aikaa.

$$y[x] = \sum_{n=1}^{\infty} h[x] * g[n - x] \tag{1}$$

Signaalien konvoluutiosta käytetään merkintää:

$$y = h \otimes g \tag{2}$$

Kuvassa 1 on esitetty kahden suorakaidepulssin konvoluutiokuvaaja. Tämä tarkoittaa sitä, että järjestelmän systeemifunktio ja sisäänmenevä signaali ovat suorakaidepulsseja, mutta järjestelmän ulostuleva signaali ei ole reunoiltaan suorakaidepulssi, joka sen tulisi olla. Tällöin ulostuleva signaali on virheellinen ja häiriön on aiheuttanut järjestelmän systeemifunktio.

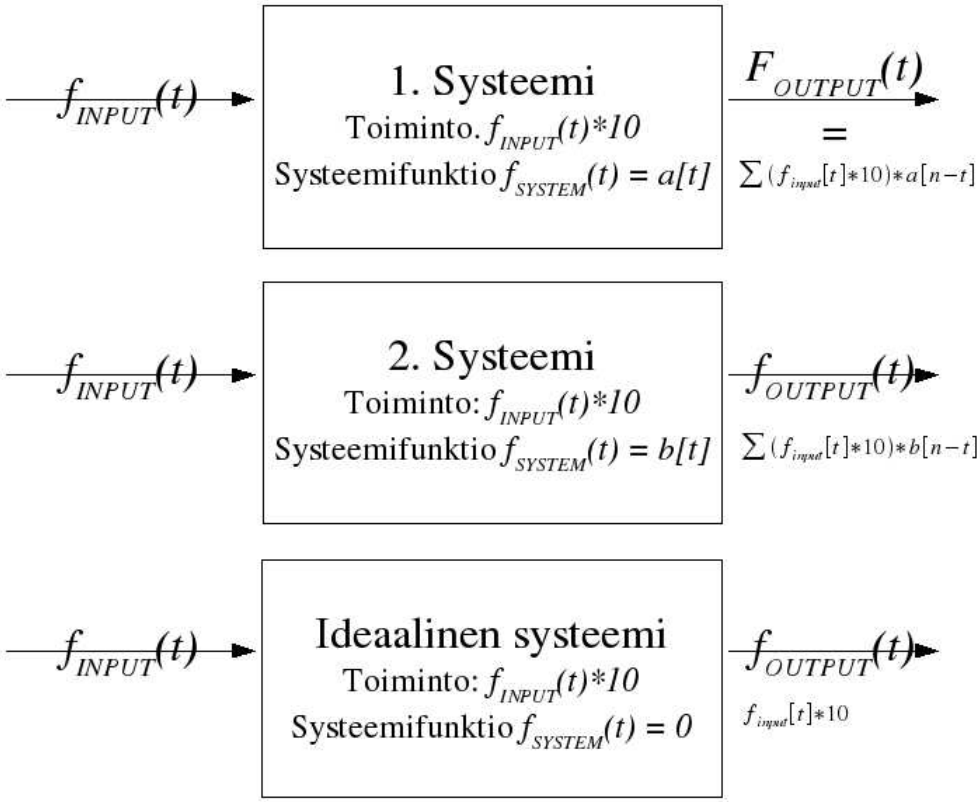


Kuva 1: Kahden suorakaidepulssin konvoluutiokuvaaja.

Systeemifunktion määrittäminen voidaan toteuttaa käyttämällä sisäänmenossa sellaista signaalia jonka taajuusvaste tunnetaan, esim. kanttiaalto, HeNe-laser. Tällöin vertaamalla ulostulosignaalia sisäänmenosignaaliin löydetään järjestelmän systeemifunktio. Kun systeemifunktio ja ulostulosignaali tunnetaan, voidaan systeemifunktion aiheuttama vääristymä korjata. Tämän toteuttaminen on kuitenkin hankalaa, koska taajuustaso tarkastelussa joudutaan käyttämään Fourier-muunnosta ja sen käänteismuunnosta. Menetelmän käyttäminen on lisäksi hankalaa käytännön reaaliaikasovelluksissa [2]. Järjestelmän systeemifunktiolle voidaan kuitenkin eaaliaikasovellusta varten voidaan laatia laskennallisesti yksinkertaisempi malli, konvoluutio-periaatteen mukaisesti.

Järjestelmän systeemifunktiolla tarkoitetaan sitä virhettä, jonka järjestelmä aiheuttaa sisäänmenosignaaliin. Tämä virhe määritetään vertaamalla tarkkaa ulostuloa vastaavaa esitystä eli referenssidataa, järjestelmästä ulostuloina saatuihin arvoihin. Referenssin määritys on oltava sellainen, että järjestelmän toiminta saadaan tutkittua mahdollisimman laajalla dynamiikalla. Tässä tutkielmassa kalibrointia sovelletaan ccd-kameraan ja näyttölaitteeseen, joten referenssidatan tulisi käsittää näiden järjestelmän toiminta-alue mahdollisimman laajasti. Kuvassa 2 on esimerkkinä kolmen järjestelmän toiminta, jossa järjestelmä vahvistaa sisäänmenosignaalia. Systeemeissä 1 ja 2 lopputulos eroaa

ideaalisesta systeemistä, eli on virheellinen, järjestelmien systeemifunktion vaikutuksesta.



Kuva 2: Systeemifunktion vaikutus järjestelmän toimintaan.

2.2 Ohjelmallisen kalibroinnin toteuttaminen reaaliaikaisesti

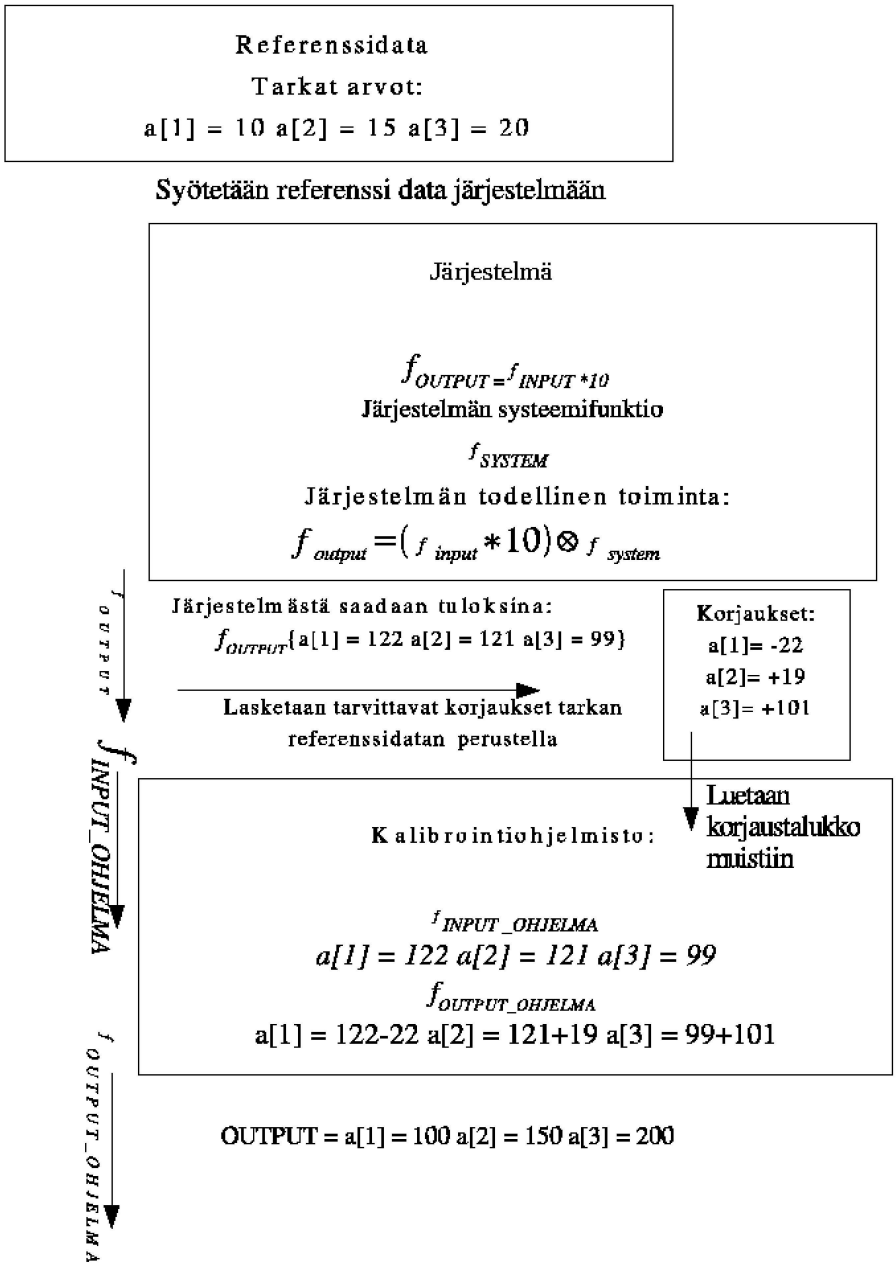
Kun järjestelmän systeemifunktio on määritetty, saadaan tieto niistä virheistä joita järjestelmä aiheuttaa käsiteltävään signaaliin. Järjestelmän kalibrointi tapahtuu tällöin siten, että systeemifunktion vaikutus poistetaan ulostulosignaalista. Systeemifunktio määritetään siten, että se kattaa järjestelmän koko dynamiikan. Esimerkkinä järjestelmään syötetään referenssiaallonpituus x, jonka arvo a, ideaalisesta järjestelmästä ulostulona pitäisi saada arvo a. Tämän jälkeen syötetään aallonpituus x, jonka arvo on myöskin a. Kuitenkin järjestelmästä saadaan ulostulona arvo b. Tällöin b on virheellinen arvo ja virhe on syntynyt järjestelmän systeemifunktion vaikutuksesta. Tämä virhe voidaan kuitenkin korjata laskemalla arvojen a ja b ero d ja tällöin tiedetään että aallonpituudella x järjestelmän systeemifunktion arvo on d. Vastaavat mittaukset suoritetaan koko järjestelmän dynamiikalle. Tällöin kalibrointi ohjelma saa syötteenä systee-

mifunktion arvot järjestelmän toiminta-alueella ja systeemifunktion vaikutus voidaan poistaa järjestelmästä [4].

Reaaliaikasovelluksessa pyritään minimoimaan kalibroinnin suorituksessa syntyvä viive. Tällöin joudutaan hakemaan kompromissi kalibroinnin tarkkuuden ja laskennassa syntyvän viiveen välille. Käytännössä tämä tarkoittaa sitä, että systeemifunktion tarkkuutta joudutaan rajoittamaan, jolloin ei saavuteta ideaalista lopputulosta, mutta reaaliaikaisuus pystytään toteuttamaan. Viiveen minimointi voidaan toteuttaa esimerkiksi taulukoimalla systeemifunktio siten, että taulukon alkiot vastaavat järjestelmän dynamiikkaa, sopivalla tarkkuudella. Koska kysymyksessä on ohjelmallinen kalibrointi, on kaikki käsiteltävä data digitaalisessa muodossa. Kalibroinnin nopeuden kannalta on tärkeää, että systeemifunktion tallennus tapahtuu sellaisessa muodossa, että korjauksen laskentaan kuluva aika saadaan mahdollisimman pieneksi. Kuvassa 3 esitetään periaate ohjelmallisesta kalibroinnista.

Esimerkkinä nopeasta toteutuksesta voidaan mainita se, että systeemifunktio on tallennettu samassa muodossa järjestelmän ulostulodatan kanssa, jolloin korjauksen laskeminen voidaan toteuttaa yhteen- ja vähennyslaskujen avulla, nämä operaatiot ovat erittäin nopeita kaikilla prosessorityypeillä. Kehittyneillä prosessorityypeillä tätäkin lasketaan voidaan vielä optimoida, esimerkiksi lisäämällä käskyjen suorituksen rinnakkaisuutta [5].

Tässä tutkielmassa keskitytään ccd-kameran ja näyttölaitteen värikalibrointiin RGB-väriavaruudessa. Kalibrointi voidaan optimoida varsin helposti, koska kameran ja näyttölaitteen systeemifunktio voidaan taulukoida RGB-arvoina, joka dynamiikan alueella on välillä 0-255. Tällöin jokaisen RGB-kanavan virheet voidaan korjata yksilöllisesti. Systeemifunktio voidaan selvittää mittaamalla vastaavat RGB-arvot referenssidatasta ja vertaamalla mittaustuloksia tarkkoihin referenssin RGB-arvoihin. Taulukoidut kokonaislukuarvot voidaan muuntaa vastaamaan järjestelmän RGB-datan bittiesitystä, jolloin korjauksen laskenta saadaan suoritettua yhteen- ja vähennyslaskujen avulla, sekä optimointi voidaan viedä erittäin laitteistoläheiseksi.



Kuva 3: Kaavio ohjelmallisesta kalibroinnista.

3 Väriavaruudet

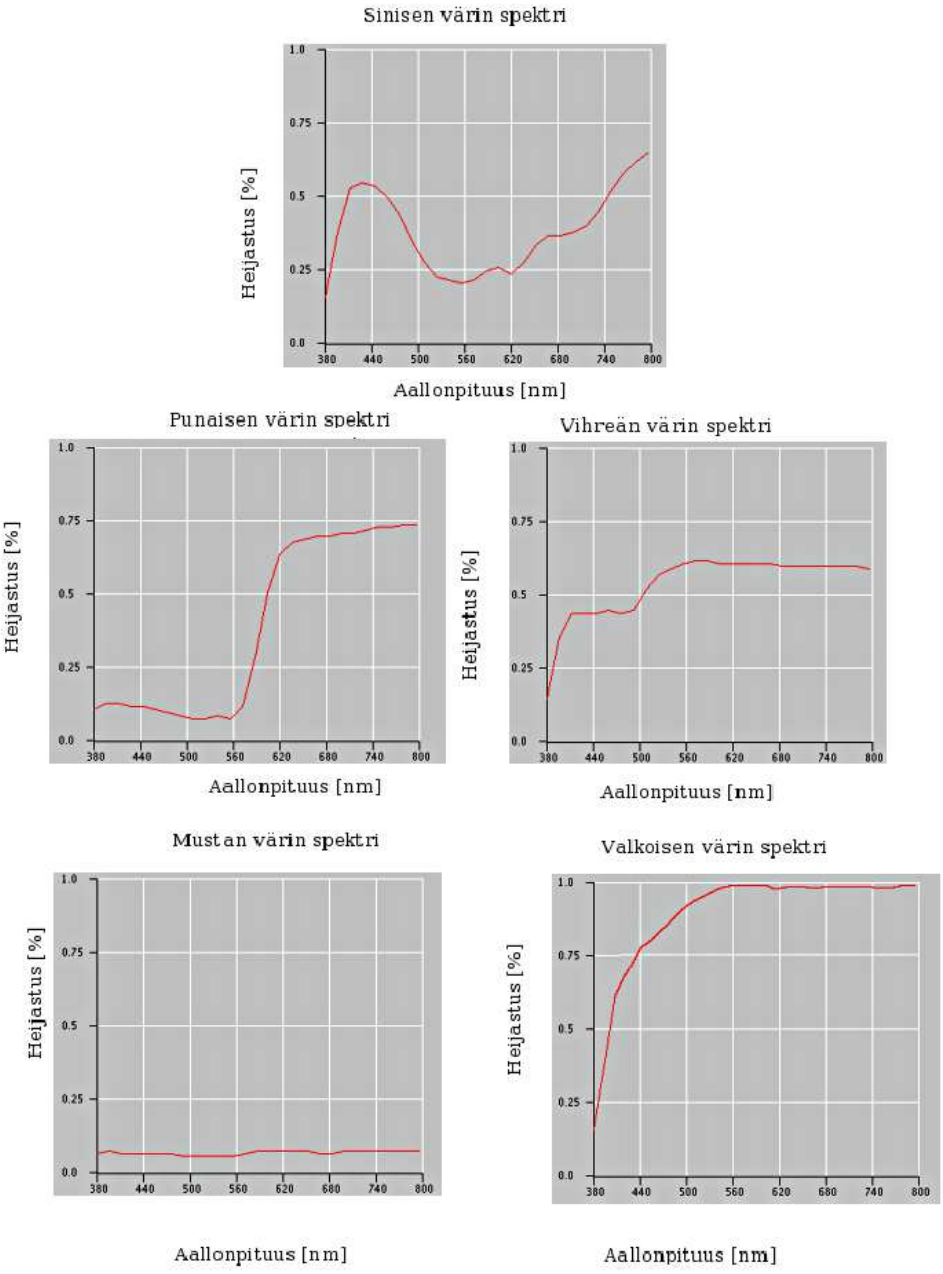
Tarkin väriesitys saavutetaan kun mitataan värin spektri, eli aallonpituuksien intensiteettijakauma. Tämä aallonpituusjakauma aistitaan värinä. Fysikaalisesti mitattava värispektri määritellään seuraavasti:

$$s(\lambda) = [s(\lambda_1), s(\lambda_2), ..., s(\lambda_n)], \tag{3}$$

missä λ on aallonpituus. Jos värispektri $s(\lambda)$ on mitattu näkyvän valon alueella (400 nm -700 nm) 10 nm välein, niin silloin $n = 31$, eli spektrissä on 31 komponenttia.

Kohteesta saatava väri on sähkömagneettista säteilyä alueella 380 nm - 780 nm, jonka intensiteetti voidaan mitata aallonpituutta kohden. Käytännön sovelluksissa käytetään aallonpituusalueena 400 nm - 700 nm. Sähkömagneettisen säteilyenergiajakauma, eli kohteessa nähtävä väri voidaan mittausten jälkeen esittää esimerkikis kuvan 4 mukaisesti sinisen, punaisen, vihreän, mustan ja valkoisen komponettien osalta [31].

Vaikka kohteen värin esittäminen spektrinä on tarkin menetelmä, niin tämä vaatii erittäin paljon dataa, joka vaikeuttaa spektriesityksen käyttämistä käytännön sovelluksissa, lisäksi spektridatan mittaaminen vaatii monimutkaisen laitteiston. Käytännön sovelluksissa väriesityksenä käytetäänkin usein kolmiulotteista väriavaruutta. Tässä tutkielmassa keskitytään RGB-väriavaruuteen, jota käytetään ccd-kameroissa ja näyttölaitteissa. Lähes kaikki väriavaruudet muodostetaan XYZ-koordinaateista, jotka taas voidaan laskea värin spektristä [4]



Kuva 4: Sinisen, punaisen, vihreän, mustan ja valkoisen värin spektrit.

3.1 Kolmiulotteinen väriesitys

Kolmiulotteiset värinäkömallit perustuvat ihmisen värinäköjärjestelmään, jossa on kolmen tyyppisiä erilaisia fotoreseptoreita. Väri esitetään näissä kolmen parametrin avulla ja useimmat värikoordinaatistot perustuvat CIE:n (Commission Internationale de l'Eclairage) XYZ-värikoordinaateihin [31]

$$X = k \int s(\lambda)\bar{x}E(\lambda)d\lambda \tag{4}$$

$$Y = k \int s(\lambda)\bar{y}E(\lambda)d\lambda \tag{5}$$

$$Z = k \int s(\lambda)\bar{z}E(\lambda)d\lambda \tag{6}$$

missä $s(\lambda)$ on värispektri, $\bar{x}, \bar{y}, \bar{z}$, ovat ihmissilmän herkkyysfunktiot, $E(\lambda)$ on valaistusspektri ja k on normitustekijä. Normitustekijä valitaan seuraavasti:

$$k = \frac{100}{\int E(\lambda)\bar{y}d\lambda} \tag{7}$$

CIE suosittelee, että käytännön laskuissa integraalit korvataan summilla, jolloin

$$[XYZ] = k \sum_{\lambda} E(\lambda)s(\lambda)[\bar{x}(\lambda)\bar{y}(\lambda)\bar{z}(\lambda)] \tag{8}$$

sekä

$$k = \frac{100}{\sum_{\lambda} E(\lambda)\bar{y}d\lambda} \tag{9}$$

3.2 CIE 1931 standardihavaitsija

CIE 1931 standardihavaitsija perustuu seitsemällä koehenkilöllä tehtyihin värisovitusfunktioiden, $r(\lambda), g(\lambda), b(\lambda)$ määrityksiin, joissa koehenkilöille näytettiin erivärisiä monokromaattisia ärsykkeitä 2 asteen katselukumassa. Koehenkilöiden tuli säätää viereisen kentän väri samaksi sekoittamalla kolmea primäärivärä R,G ja B. Joten voidaan ajatella, että väriärsyke on muotoa

$$\bar{F} = \bar{r}(\lambda)\bar{R} + \bar{g}(\lambda)\bar{G} + \bar{b}(\lambda)\bar{B} \tag{10}$$

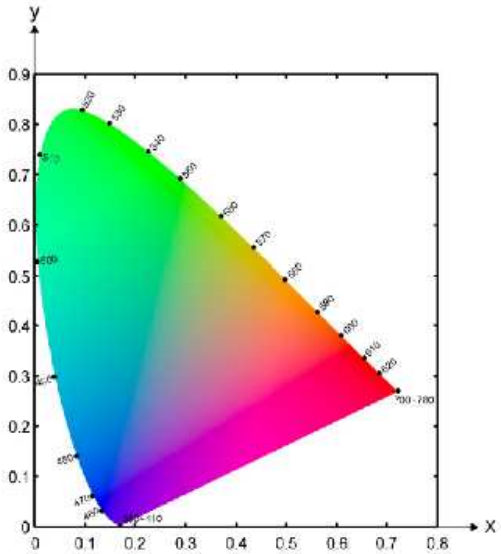
missä, $\bar{r}(\lambda)$,$\bar{g}(\lambda)$ ja $\bar{b}(\lambda)$ ovat spektraaliset tristimulus värisovitusfunktiot. Tristimulusarvot eivät kata koko spektriä, vaan aallonpituusalueella 435,8 - 546,1 $\bar{r}(\lambda)$:n arvot ovat negatiivisia. Tällöin valonaistimukseen joudutaan lisäämään punaista, jotta värikentät saataisiin näyttämään samoilta. Sopivalla muunnoksella saadaan värisovitusfunktiot, $\bar{x}(\lambda)$,$\bar{y}(\lambda)$,$\bar{z}(\lambda)$ [31].

3.3 CIE xy-koordinaatisto

Olkoon $E(\lambda)$ valonlähteen spektri ja $s(\lambda)$ kohteen läpäisy tai heijastuminen. Tarkastellaan tilannetta jossa $s(\lambda) = 1$, eli täysin valkoinen valo. Kohteen XYZ-arvot lasketaan kaavojen, 4,5,6 avulla, joista saadaan x, y, z arvot seuraavasti:

$$x = \frac{X}{X+Y+Z}, y = \frac{Y}{X+Y+Z}, z = \frac{Z}{X+Y+Z} \tag{11}$$

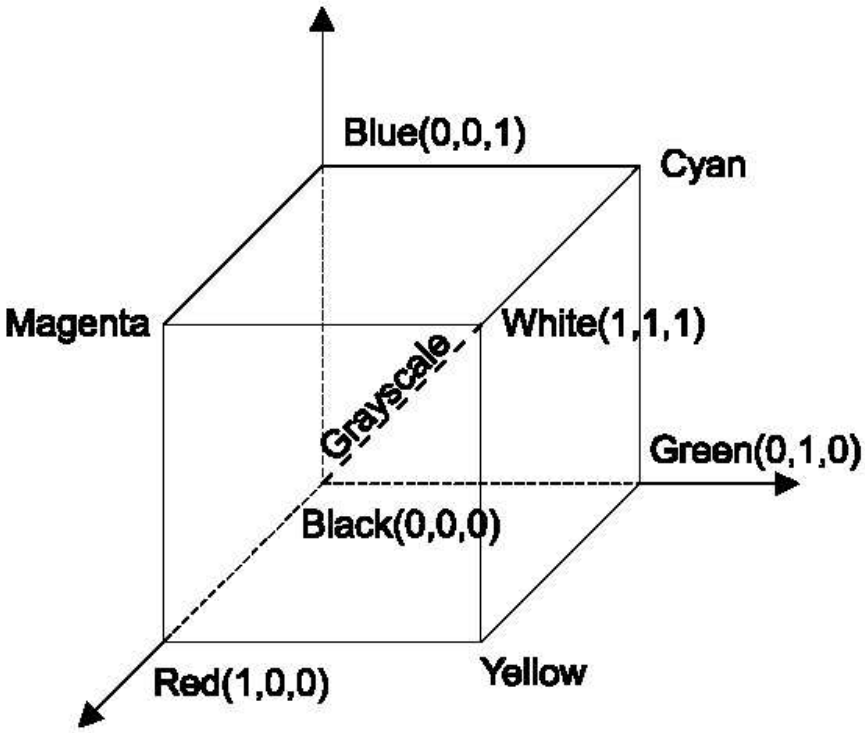
Edellisestä kaavasta huomataan, että $z = 1-x-y$. Kaavassa 7 oleva normitustekijä antaa Y:lle arvon Y=100, kun $s(\lambda)$ = 1 $\forall\lambda$. Y vastaa luminanssia, CIE xy-koordinaatiston on esitetty kuvassa 5.



Kuva 5: CIE xy-värikoordinaatisto.

3.4 RGB-väriavaruus

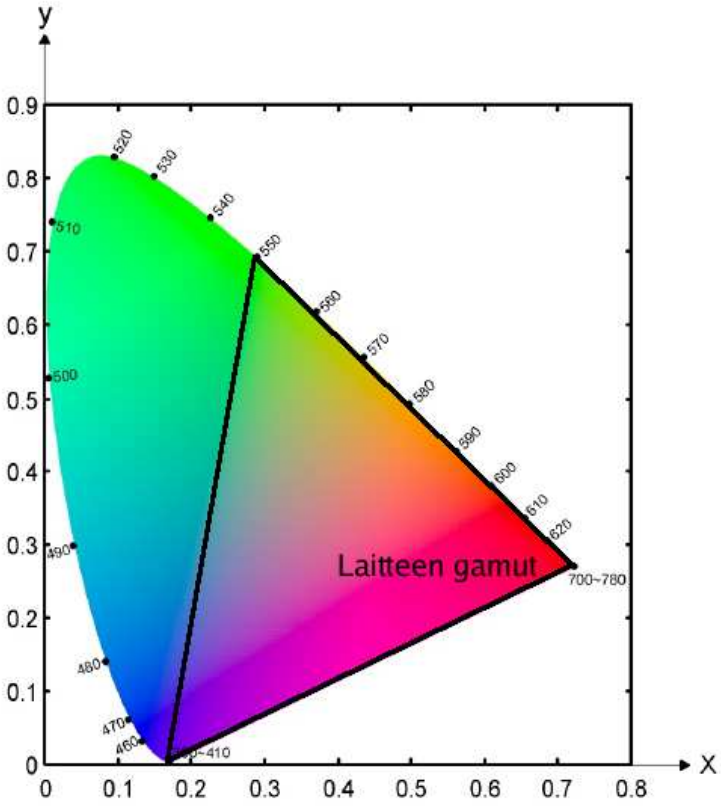
Digitaalisissa kuvasjärjestelmissä ja näyttölaitteissa käytetään RGB-väriavaruutta. RGB-väriavaruudessa jokainen väri esitetään punaisen, vihreän ja sinisen komponentin sekoituksena. Väriavaruutena RGB on erittäin yksinkertainen ja tehokas. RGB-väriavaruus on lineaarinen. Lineaarisuus tarkoittaa sitä, kun kaksi valoa lisätään toisiinsa additiivisesti, saadaan piste väriavaruudessa, joka on niiden pisteiden vektorisumma, jotka vastaavat kahta kyseistä komponentti valoa. RGB-väriavaruudessa komponentit saavat arvot välillä 0-255, RGB-arvo [0,0,0] vastaa mustaa ja [255,255,255] vastaa valkoista [9]. Yleensä RGB-primäärit oletetaan monokromaattisiksi. Näin ollen voidaan R:n G:n ja B:n ajatella olevan keskenään ortogonaaliset ja RGB-koordinaatisto voidaan esittää kolmiulotteisena karteesisena koordinaatistona kuvassa 6 [31].



Kuva 6: RGB-yksikkökuutio [31].

RGB-väriavaruuden yksinkertaisuus aiheuttaa kuitenkin ongelmia tarkan väriesityksen kannalta. Tässä tutkielmassa pyritään värikalibroinnin avulla parantamaan RGB-väriesityksen tarkkuutta ohjelmallisesti tietyissä valaistusolosuhteissa [9]. Syy näihin ongelmiin on se, että kahdella primäärivärillä saadaan tuotettua additiivisesti xyY-

koordinaatistossa ne värit, jotka ovat primärivärien kautta piirretyllä suoralla. Kolmella monokromaattisella primäärivärillä toteutetulla RGB-koordinaatistolla voidaan esittää suurin osa xyY-värikoordinaatistosta, eli ne värit jotka jäävät primärivärien rajaaman kolmion (gamut) sisään. Kuvassa 7 on RGB-primäärit valittu seuraavasti: $\lambda_R = 700nm, \lambda_G = 550nm$ ja $\lambda_B = 430nm$, tällaisilla primääreillä varustettu laite pystyy esittämää piiretyn kolmion sisään jäävät värit. Kuvasta 7 huomataan, että erityisesti vihreiden sävyjen esityksessä on puutteita RGB-väriavaruudessa. Käytännön sovelluksissa tämä ei aiheuta suurta ongelmaa, koska xy-värikoordinaatisto on epälineaarinen, jolloin värierot ovat pienempiä vihreällä alueella kuin muilla alueilla xy-värikoordinaatistossa.



Kuva 7: RGB-primäärien rajaama kolmio (gamut) xy-koordinaatistossa.

Muunnos tristimulusarvoista RGB-arvoiksi suoritetaan seuraavan muunnosmatriisin avulla [31]:

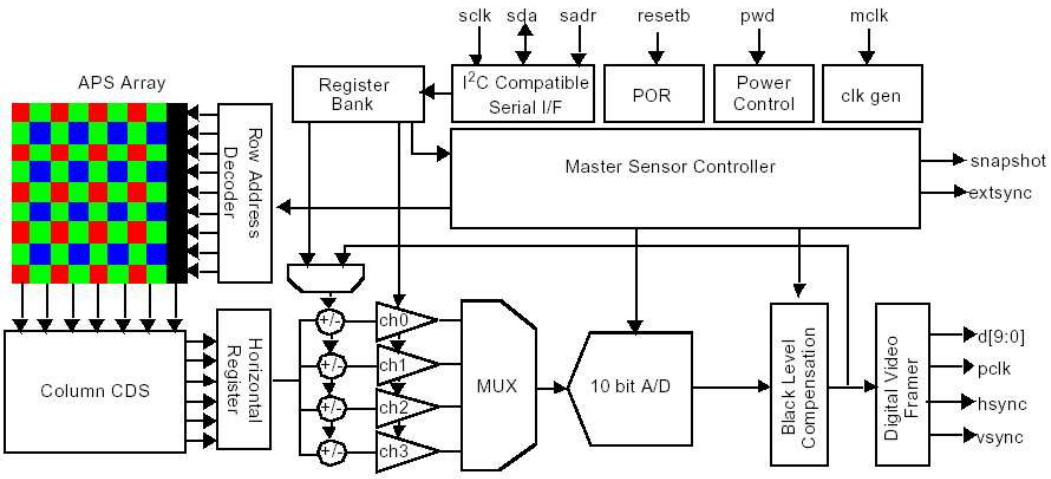
$$\left[\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 3.2410 & -1.5374 & 0.4986 \\ -0.9692 & 1.8760 & 0.0416 \\ 0.0556 & -0.2040 & 1.0570 \end{bmatrix} * \begin{bmatrix} X \\ Y \\ Z \end{bmatrix}\right] \tag{12}$$

4 Ccd-kameran reaaliaikainen värikalibrointi

Tutkielmassa esitellään tekniikka, jolla saavutetaan reaaliaikainen värikalibrointi ccd-kameralle. Reaaliaikaisuudella tarkoitetaan sitä, että kuvausnopeudeksi saadaan 30 ruutua sekunnissa, joka tarkoittaa 30 ms viivettä ruutujen välillä. Käytännön toteutuksessa suurin osa viiveestä aiheutuu datan siirrosta ccd-kameralta siihen järjestelmään, joka esittää videokuva näyttölaitteella, joten käytännössä värikalibroinnin laskemiseen voi jää aikaa alle 1 ms. Värikalibrointialgoritmin toteutuksessa on pyritty myös siihen, että algoritmi voidaan toteuttaa useilla erilaisilla laitealustoilla sekä tarvittaessa myös sulautettuna järjestelmänä. Myös systeemifunktion mittaaminen on toteutettu siten, että mittaus voidaan suorittaa automatisoidusti.

4.1 Kuvadatan muodostuminen ccd-kamerassa

Ccd-kenno toimii siten, että jokainen pikseli muodostuu neljästä RGB-komponentista (punainen, 2 vihreää ja sininen). Kennolle saapuvan valon aallonpituusjakauman mukaan kennolta saadaan erilaisia jännitetasoja, jotka muunnetaan digitaalisiksi RGB-arvoiksi, esimerkiksi 8 bittiä / komponentti [7]. Ccd-kennon tarkka rakenne vaihtelee eri valmistajilla, kuvassa 8 on esitetty National Semiconductorin valmistaman ccd-kennon rakenne. Kuvan muodostuminen alkaa ccd-sirussa, (ASP-array), jossa erivärisillä suodattimilla varustetut varausparit muodostavat jännite-eron, joka riippuu tietyn aallonpituuden voimakkuudesta. Jännite-erot muunnetaan digitaalisiksi sanoiksi 10-bittisessä AD (Analog Digital)-muuntimessa. Nämä sanat tallennetaan rekistereihin, joista data siirretään esim. tietokoneeseen, sopivaa väylää käyttäen. Tietokoneisiin liitettävissä kameroissa väylänä käytetään I^2C väylää, jolla data siirretään USB-väylän ohjaimeen [21].



Kuva 8: Esimerkki ccd-kennon rakenteesta [30].

4.2 Ccd-kamerassa tapahtuvat RGB-väriavaruuden virheet

Reaaliaikaisessa toteutuksessa huomioidaan kolme erityyppistä tekijää, jotka muodostavat ccd-kameran systeemifunktion. Ccd-kameran elektroniikassa olevat vuotovirrat aiheuttavat kohinaa kuvadataan. Ccd-kennon RGB-komponenttien taajuusvaste ei ole lineaarinen, vaan vaihtelee eri taajuusalueilla. Edelliset virheet korostuvat erityisesti, silloin kun ccd-kennoa ei ole jäähdytetty. Kuvasympäristön valaistus vaikuttaa aina ccd-kamerasta saatavaan kuvadataan [5]. Digitaalisessa kuvadatassa nämä virheet tarkoittavat sitä, että jokainen ccd-kameralta luettu pikselin arvo on muodostunut ccd-kennon AD-muunnoksesta saadusta datasta, sekä ccd-kennon virheiden aiheuttamista virheistä em. dataan [7,8].

4.3 Ccd-kameran systeemifunktion määrittäminen

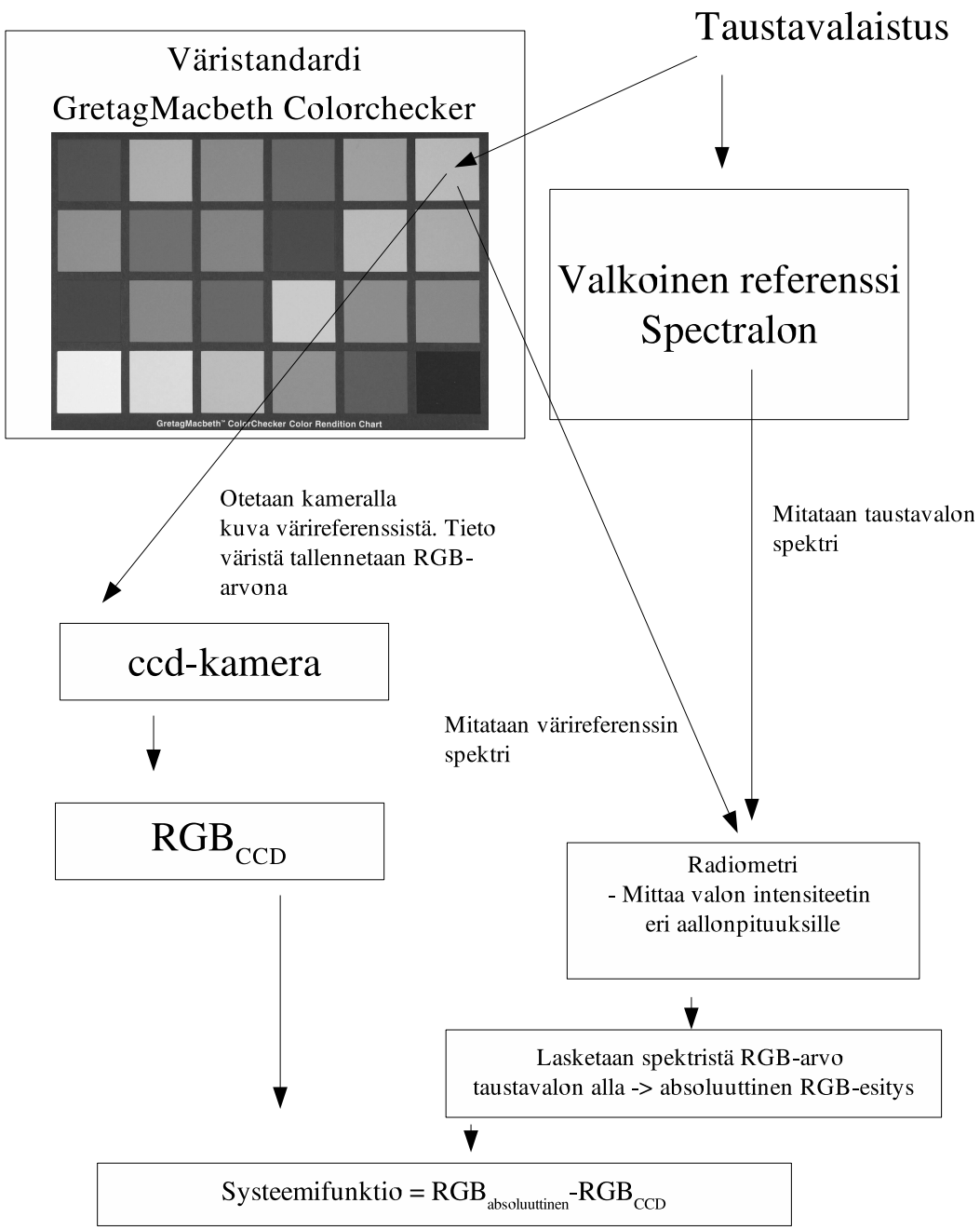
Ccd-kameran systeemifunktion määrittäminen voidaan suorittaa siten, että kuvataan kameralla värireferensseistä RGB-kuvat. Yleensä väristandardi on GretagMacbeth jossa on 24 erilaista värireferenssiä, joten saadut tulokset joudutaan interpoloimaan siten, että jokaiselle RGB-arvolle 0-255 saadaan vastaava systeemifunktion arvo [7]. Interpoloidut arvot järjestetään suuruusjärjestykseen kameran RGB-datan mukaisesti. Kameran jokaisen referenssin RGB-datan arvoista lasketaan keskiarvot ja niitä verrataan väristandardin tarkkoihin RGB-arvoihin. Tarkkojen RGB-arvojen ja kameran RGB-

arvojen eroista muodostetaan taulukko, jossa jokaisella RGB-komponentilla on vastaava systeemifunktion arvo. Tällä menetelmällä koko ccd-kameran dynamiikan käsittävä systeemifunktio saadaan esitettyä kokonaislukutaulukkona, joka voidaan antaa syötteenä kalibrointiohjelmistolle. Korjaustaulukon arvot järjestetään suuruusjärjestykseen mittausarvojen intensiteetin mukaisesti. Esimerkit korjaustaulukon arvoista on esitetty taulukossa 1. Saatu taulukko on yksilöllinen tietylle ccd-kameralle. Tämä systeemifunktion määrittäminen voidaan toteuttaa helposti automatisoituna. Lisäksi tämä systeemifunktion esitystapa mahdollistaa värikalibroinnin suorittamisen pelkästään kokonaislukujen yhteenlaskun avulla.

Taulukko 1: Korjausarvot kameran RGB-arvoille.

Vastaava arvo	R	G	B
0	10	-12	5
1	23	-33	4
..	.	.	.
255	10	-12	5

Kuvassa 9 on esittety mittausjärjestely kameran systeemifunktion määrittämiseksi.



Kuva 9: Kameran systeemifunktion mittausjärjestely.

4.4 Reaaliaikainen ccd-kameran värikalibrointialgoritmi

Jotta kalibroinnissa aiheutuva viive saataisiin minimoitua useissa erilaisissa tietokonearkkitehtuureissa, hyödynnetään sitä, että kameran systeemifunktio on esitetty tau-

lukoituina RGB-arvoina, jolloin kalibroinnin suorittaminen saadaan toteutettu mahdollisimman yksinkertaisilla operaatiolla. Ccd-kameran kalibrointi toteutetaan jokaiselle pikselille seuraavasti: Vähennetään aluksi kameralla kuvatun mustan referenssin RGB-arvot kameralta saadusta RGB-datasta. Jaetaan kameralla kuvatun valkean referenssin RGB-arvoilla kameran RGB-data. Lopuksi lisätään pikselin RGB-arvoja vastaavat korjaustaulukon (kalib_data), kuten taulukon 1 mukaisesti, alkioiden RGB-arvot kameran RGB-dataan.

Seuraavana värikalibrointialgoritmi esitetään pseudokielisenä toteutuksena.

Algoritmi 1: Värikalibrointi

```
 for each pixel in kamera_data
{
kamera_data_rgb[x,y] =
kamera_data_rgb[x,y] - kamera_musta_rgb[x,y];

kamera_data_rgb[x,y] =
kamera_data_rgb[x,y] /
(kamera_valkea_rgb[x,y] - kamera_musta_rgb[x,y]);

kamera_data_rgb[x,y] =
kamera_data_rgb[x,y] + kalib_data[kamera_data_rgb[x,y]];

}
```

4.5 Kalibrointidatan laskeminen

Kalibrointidatan laskeminen tapahtuu seuraavaan algoritmin mukaisesti.

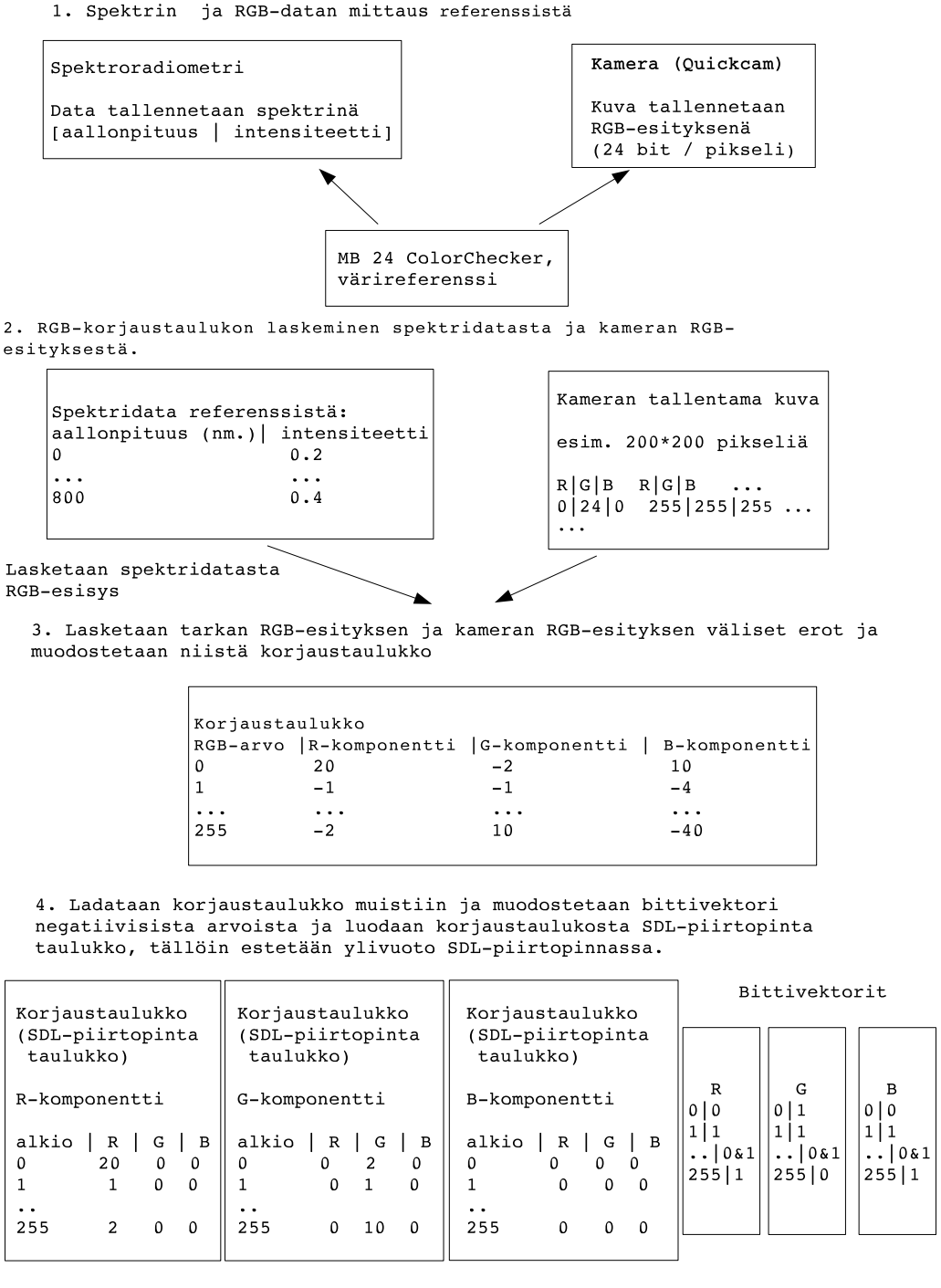
Algoritmi 2: Kalibrointidatan laskeminen

```
     lataa musta_referenssi
lataa valkea_referenssi
for each väri_referenssi
{
lataa väri_referenssi
väri_referenssi = väri_referenssi - musta_referenssi
väri_referenssi = väri_referenssi / valkea_referenssi
tallenna väri_referenssi taulukkoon
}

lataa musta_spektridata
lataa valkea_spektridata
valkea_spektridata = valkea_spektridata - musta_spektridata
for each spektridata
{
lataa spektridata
spektridata = spektridata - musta_spektridata
spektridata = spektridata / valkea_spektridata
tallenna spektridata taulukkoon
}

     järjestetään spektridatat suurusjärjestykseen
järjestetään väri_referenssit spektridatan
järjestyksen mukaan
lasketaan spektridatataulukon ja
väri_referenssien erotukset
tallennetaan erotukset taulukkoon
interpoloidaan erotus_taulukot välille 0-255
```

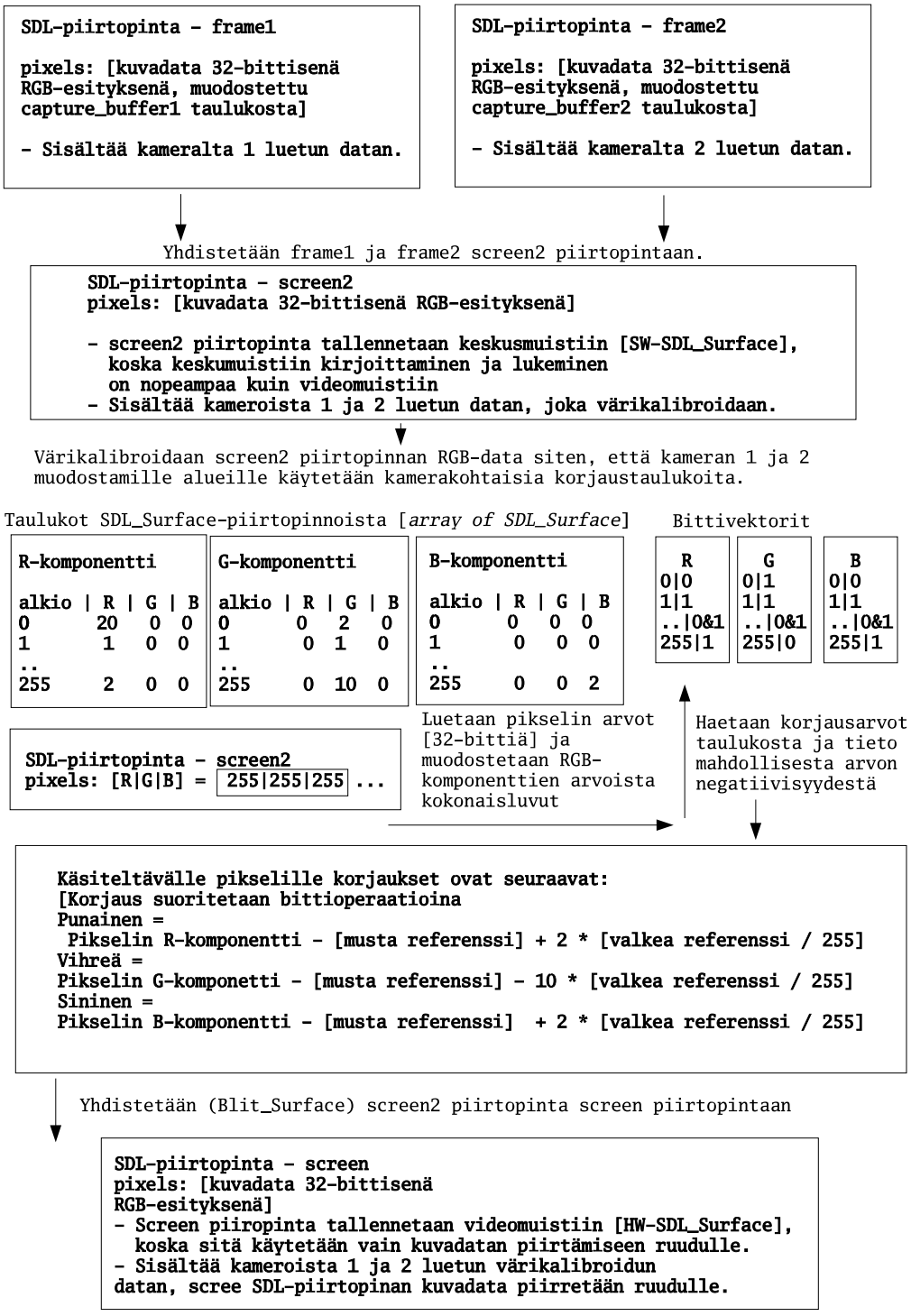
Värireferenssillä tarkoitetaan esimerkksi GretagMacBeth 24 värireferenssin jotakin väriä. Kalibrointidatan laskeminen on esitelty kuvassa 10. Simple Direct Layer (SDL)-piirtopintaa käytetään esimerkki tallennusmuotona.



Kuva 10: Kalibrointitaulukon muodostaminen.

4.6 Esimerkkitapaus ccd-kameran värikalibroinnin suorittamisesta

Tässä esimerkissä on käytetty grafiikka-rajapintana Simple Direct Layer (SDL)-rajapintaa, Linux-käyttöjärjestelmässä. Esimerkissä käytetään kahta kameraa, jotta menetelmää voidaan käyttää myös stereokuvaukseen. Värikalibroinnissa luetaan korjaustaulukko tekstitiedostosta muistiin. Korjaustaulukko on laskettu Matlab-ohjelmalla. Korjausdatan laskemiseen tarvittava, kameroilla kuvattu data, tallennetaan BMP-muodossa videodatasta. BMP-muotoiset kuvat tallennetaan videodatasta SDL-rajapintaan käyttäen. Värikalibroinnissa käytetään 32-bittistä tallennusmuotoa kuvadatalle, koska 32-bittinen esitys on nopeampaa kuin 24-bittinen, mikäli X-windows järjestelmässä käytetään 32-bittistä tilaa. Kuvadata luetaan kameroilta 24-bittisenä. Muunnos eri tallennusmuotojen välillä tehdään SDL-rajapinnan avulla. Korjaustaulukon arvot on tallennettu 32-bittisenä RGB-esityksenä SDL-piirtopintaan. Korjaustaulukossa voi olla myös negatiivisia arvoja, mutta ne tallennetaan positiivisina arvoina korjaustaulukkoon. Tiedot niistä taulukon kohdista, joissa on negatiivisia arvoja on tallennettu bittivektoriin. Jokaisella RGB-komponentilla on oma bittivektorinsa. Korjaustaulukon käsittelyä varten kuvadatan RGB-komponenttien arvot muutetaan kokonaisluvuiksi, tähän käytetään SDL-piirtopinnan sisältämiä tietoja eri komponenttien pituuksista bitteinä. [18] Värikalibroinnin toteutus on esitetty kuvassa 11.



Kuva 11: Värikalibroidun videokuvan muodostaminen [18].

5 Näyttölaitteen reaaliaikainen värikalibrointi

Seuraavaksi tutkielmassa esitellään toteutustapa näyttölaitteen värikalibroinnille, jossa kalibrointi saadaan suoritettua reaaliaikaisesti siten, että kohdassa 2 esitelty ccd-kameran värikalibrointi ja näyttölaitteen värikalibrointi voidaan suorittaa ilman, että ruutujen välinen viive kasvaisi yli 30 ms. Näyttölaitteessa muodostetaan digitaalisesta RGB-datasta valoa, joka havaitaan. Taustavalaistus vaikuttaa hyvin paljon, siihen miten näyttölaitteen muodostama valo havaitaan. Tässä tutkielmassa esitetyn värikalibrointialgoritmin avulla pyritään muodostamaan mahdollisimman virheetön näyttölaitteen esittämä RGB-esitys erilaisissa valaistusolosuhteissa. Lisäksi pyritään minimoimaan näyttölaitteen taajuusvasteen epälineaarisuus, käsittäen koko RGB-väriavaruuden dynamiikka. Näyttölaitteen systeemifunktion määrittämiseen sovelletaan samaa periaatetta kuin ccd-kameroiden systeemifunktion selvittämisessä.

5.1 Näyttölaitteen (CRT / TFT) toimintaperiaate lyhyesti

Kun järjestelmässä tuotettu videodata halutaan näyttää näyttölaitteella, tarvitaan näyttölaitteen lisäksi myös näytönohjausjärjestelmä, joka muokkaa prosessorilta tulevan datan näyttölaitteelle sopivaksi. Yleisesti videodata käyttää RGB-väriavaruutta, jolloin data koostuu kolmesta 8 bitin komponentista. Teoriassa voidaan siis muodostaa 16.8 miljoonaa erilaista värisävyä yhdelle pikselille. CRT-näytössä digitaalisesta informaatiosta muodostetaan valoa, joka aistitaan väreinä, ohjaamalla elektroneja fosforipäällysteeseen, josta emittoituu valoa. Valon voimakkuutta säädetään muuttamalla elektronisuihkun voimakkuutta. Aktiivimatriisinäyttö rakentuu kahdesta hyvin lähekkäin olevasta lasilevystä, joiden väliin laitetaan nestekidemassaa. Ylemmän levyn päälle tulee RGB-kaavan mukaisesti väritetty värisuodatin (punainen, sininen ja vihreä) ja polarisoiva kalvo. Alemman levyn alapuolelle sijoitetaan omalle lasilevylle matriisiin ohutkalvotransistoreita (TFT - Thin Film Transistor) yksi jokaista osapikseliä kohden. Jokainen osapikseli on kytketty aiemmin mainittuun matriisiin ja pikseleitä ohjataan aktivoimalla haluttu rivi ja varaus lähetetään oikeaan sarakkeeseen. Tarkalla jänniteohjauksella jokaisesta väristä saadaan tuotettua 256 eri sävyä eli kaikki kolme väriä ja niiden sävyt yhdistämällä saadaan 16,8 miljoonaa eri värisävyä. Nestekidenäytön toiminta perustuu nestekiteen valoa heijastavien ominaisuuksien muuttamiseen jännitteen avulla. Jännitteen suuruus määrää kuinka paljon nestekiteet kääntyvät ja kuinka paljon

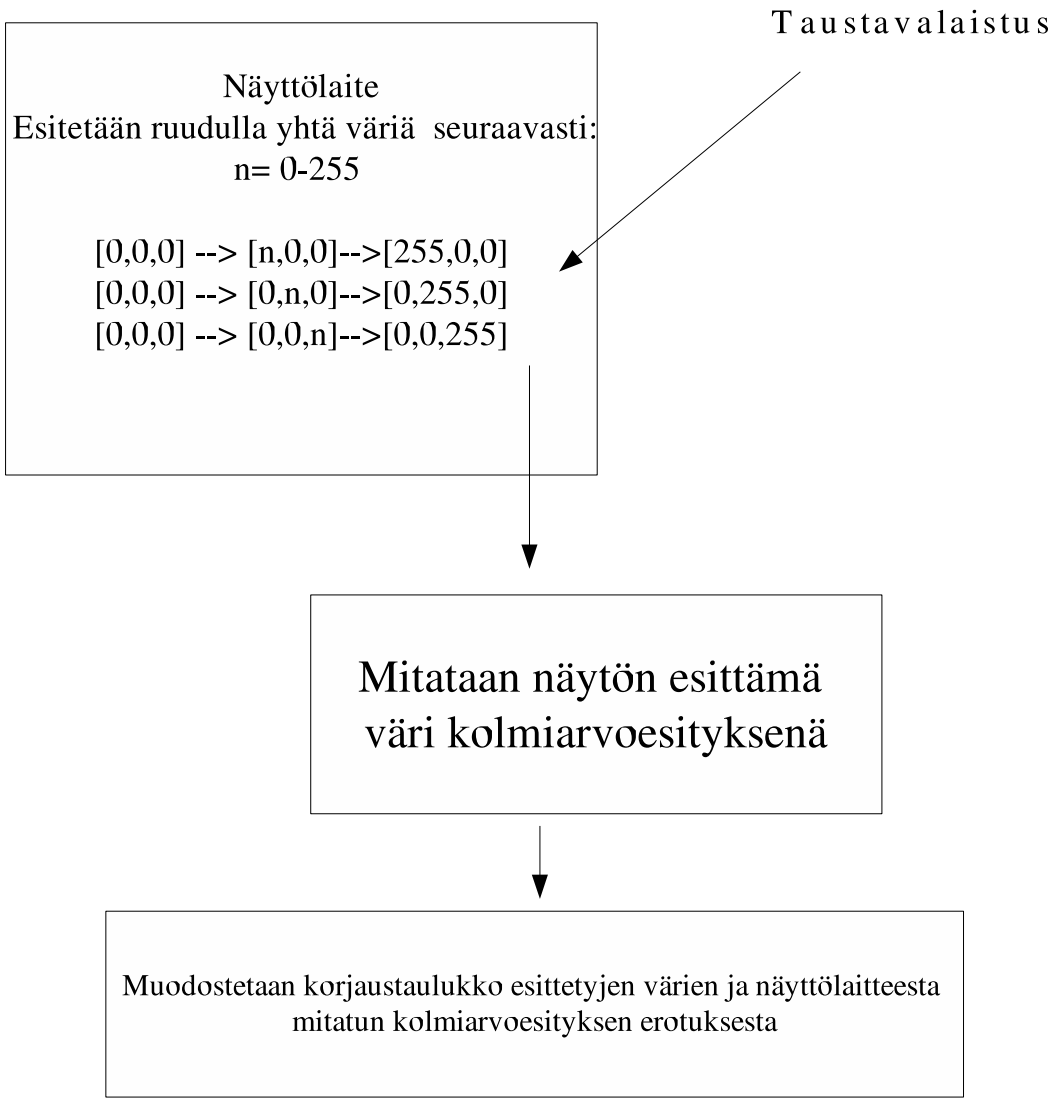
valoa pääsee läpi. Seuraavaksi valo osuu värisuodattimeen, jonka jälkeen luodaan yksi pikseli kolmea pääväriä hyväksi käyttäen [11].

5.2 Näyttölaitteen systeemifunktion määrittäminen

Systeemifunktio sisältää tiedon näyttölaiteen RGB-esityksen virheistä, verrattuna ideaaliseen tapaukseen. Systeemifunktion selvittäminen aloitetaan mittaamalla kromametrillä kolmiarvoesitykset näytössä esiintyville väreille. Näytölle muodostetaan värit, RGB-arvoina 0-255, kaikille kolmelle komponentille siten, että kaksi komponenttia ovat 0 ja kolmannen arvoa kasvatetaan välillä 0-255. Näin saadaan selville näytön taajuusvaste [11]. Kromametrillä mitatut arvot eivät yleensä ole RGB-arvoja vaan ne ovat jonkin muun kolmiulotteisen väriavaruuden arvoja, joten kromametriltä saadut arvot joudutaan muuttamaan RGB-väriavaruuteen. Tässä tapauksessa näytölle muodostetut värit toimivat referenssiarvoina, joihin mitattuja arvoja verrataan. Mitatut arvot sisältävät myös tiedon valaistusolosuhteista. Mitattujen arvojen ja referenssiarvojen erotuksesta saadaan muodostettua näyttölaitteen systeemifunktion arvot, kattaen koko näyttölaitteen dynamiikan, ilman interpolointia. Saadut erotukset taulukoidaan, jolloin jokaista RGB-arvolle saadaan yksilöllinen korjausarvo. Korjaustaulukon arvot järjestetään suuruusjärjestykseen mittausarvojen intensiteetin mukaisesti. Esimerkit korjaustaulukon arvoista on esitetty taulukossa 2. Kaavio näyttölaitteen systeemifunktion määrittämisestä on esitetty kuvassa 12.

Taulukko 2: Korjausarvot näyttölaitteen RGB-arvoille

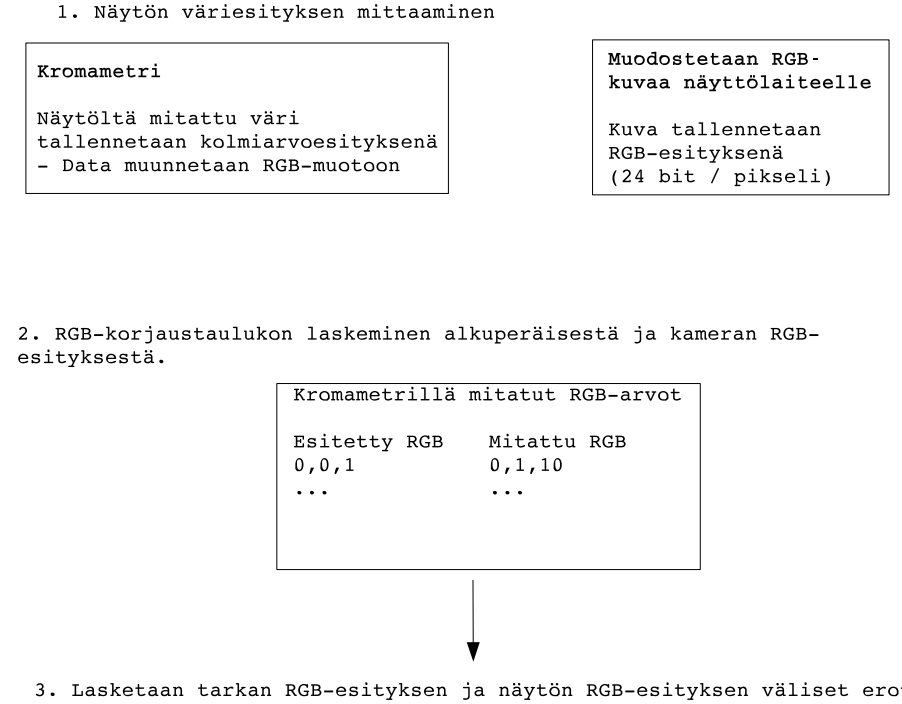
Vastaava arvo	R	G	B
0	10	-12	5
1	23	-33	4
..	.	.	.
255	10	-12	5



Kuva 12: Näyttölaitteen systeemifunktion mittausjärjestely.

5.3 Kalibrointidatan laskeminen

Kalibrointidatan laskeminen on esitelty kuvassa 13. SDL-piirtopintaa käytetään esimerkki tallennusmuotona.



Kuva 13: Kalibrointitaulukon muodostaminen

5.4 Reaaliaikainen näyttölaitteen värikalibrointialgoritmi

Jotta kalibroinnissa aiheutuva viive saataisiin minimoitua, useissa erilaisissa tietokonearkkitehtuureissa hyödynnetään sitä, että näyttölaitteen systeemifunktio on esitetty taulukoituina RGB-arvoina, jolloin kalibroinnin suorittaminen saadaan toteutettu mahdollisimman yksinkertaisilla operaatiolla. Näyttölaitteen kalibrointi toteutetaan jokaiselle pikselille seuraavasti: Lisätään pikselin RGB-arvoja vastaavat korjaustaulukon (kalib_data) alkioiden RGB-arvot näyttölaitteen RGB-dataan, algoritmin 3 mukaisesti.

Seuraavana värikalibrointi algoritmi esitetään pseudokielisenä toteutuksena.

Algoritmi 3: Näyttölaiteen värikalibrointi

```
 for each pixel in näyttölaite_data
{

näyttölaite_data_rgb[x,y] =
näyttölaite_data_rgb[x,y] + kalib_data[näyttölaite_data_rgb[x,y]];

}
```

Muuttuja kalib_data on muodostettu kuvan 13 mukaisesti.

5.5 Esimerkkitapaus näyttölaitteen värikalibroinnin suorittamisesta

Tässä esimerkissä on käytetty grafiikka-rajapintana Simple Direct Layer (SDL)-rajapintaa, Linux-käyttöjärjestelmässä. Värikalibroinnissa luetaan korjaustaulukko tekstitiedosta muistiin. Värikalibroinnissa käytetään 32-bittistä tallennusmuotoa kuvadatalle, koska 32-bittinen esitys on nopeampaa kuin 24-bittinen, mikäli X-windows järjestelmässä käytetään 32-bittistä tilaa. Korjaustaulukon arvot on tallennettu 32-bittisenä RGB-esityksenä SDLpiirtopintaan. Korjaustaulukossa voi olla myös negatiivisia arvoja, jotka kuitenkin tallennetaan arvot positiivisina arvoina korjaustaulukkoon. Tiedot niistä taulukon kohdista, joissa on negatiivisia arvoja on tallennettu bittivekto-

riin. Jokaisella RGB-komponentilla on oma bittivektorinsa. Korjaustaulukon käsittelyä varten kuvadatan RGB-komponenttien arvot muutetaan kokonaisluvuiksi, tähän käytetään SDL-piirtopinnan sisältämiä tietoja eri komponenttien pituuksista bitteinä.

Värikalibroinnin toteutus on esitetty kuvassa 14.

Kuva 14: Värikalibroidun videokuvan muodostaminen.

6 Värikalibrointialgoritmien tehokkuus ja optimointi

Värikalibrointialgoritmien korkeantason optimoinnissa keskitytään tehostamaan c-kielen kääntäjän toimintaa siten, että c-kielen kääntäjän tuottama konekielinen koodi on mahdollisimman tehokasta käytettävälle prosessorityypille. Lisäksi pyritään optimoimaan piirtorutiinit siten, että grafiikan piirto ruudulle olisi mahdollisimman tehokasta. Käyttöjärjestelmän optimoinnista tuodaan esille, Linux-käyttöjärjestelmän viritttäminen reaaliaikakäyttöön sopivaksi.

Käytännössä kuvadata on n:n pituisessa vektorissa, jossa jokainen pikseli sisältää 3 arvoa. Tällöin algoritmin aikavaativuus on $3n$. Yleisesti voidaan olettaa, että jos prosessori käyttää ajan t kokonaisluvun yhteenlaskuun niin kokonaisluvun kertolaskuun kuluu aika $4t$ ja jakolaskuun $8t$. [5] Tältä osin algoritmit toimivat varsin tehokkaasti, koska systeemifunktiot ovat jo sopivassa muodossa valmiiksi laskettuina. Lisäksi algoritmien yksinkertaisuus mahdollistaa niitten käyttämisen myös yksinkertaisissa prosessoreissa, joita käytetään sulautetuissa järjestelmissä. Lisäksi algoritmeja voidaan helposti rinnakkaistaa, jolloin niiden tehokkuutta voidaan huomattavasti kasvattaa, esimerkiksi moderneissa x86-arkkitehtuurin prosessoreissa. Laskennan rinnakkaistaminen voidaan toteuttaa käyttämällä prosessorien Single Instruction Multiple Data (SIMD)-käskykantoja. Algoritmin yksinkertainen toteutus mahdollistaa myös sen että konekielinen koodi saadaan optimoitua mahdollisimman tehokkaaksi, jolloin modernien prosessorien liukuhihnaus-ominaisuudet saadaan mahdollisemman tehokkaaseen käyttöön [11,12].

6.1 Värikalibrointialgoritmien korkeantason optimoiminen

Korkeantason optimoinnin tavoittaneena on auttaa c-kääntäjää luomaan käytettävälle prosessorille optimoitu konekielinen ohjelma, joka toimii mahdollisimman tehokkaasti ilman viiveitä. Tämä saavutetaan optimoimalla ohjelmakoodissa olevat silmukat siten, että Loop-Unrolling tapahtuu mahdollisimman tehokkaasti. Tällä menetelmällä tarkoitetaan sitä, että liukuhihnauksen sisältävät prosessorit pystyvät toteuttamaan mahdollisimman tehokkaan käskyn sisäisen rinnakkaistamisen. Lisäksi pyritään välttämään liukuhihnassa tapahtuvia pysähdyksiä [19,23]. Värikalibroinnit tapahtuvat silmukan sisässä, joten tämä optimointi toteutetaan käytännössä seuraavasti: Kirjoitetaan matemaattiset ja loogiset operaatiot korkeantason ohjelmakoodiin mahdollisemman

laitteistonläheisesti. Lisäksi minimoidaan silmukan sisällä tapahtuvat laskutoimitukset, eli taulukoidaan kaikki operaatiot, jotka voidaan laskea ennakkoon. Käytännössä tämä tarkoittaa sitä, että muokataan kaikki data samaan muotoon, jotta voidaan käyttää bittioperaatiota laskutoimituksiin. Grafiikan piirron optimoinnissa on tärkeintä se, että käytetään mahdollisimman tehokasta grafiikkarajapintaa, joka sisältää ominaisuudet näytönohjaimen laitteistotason muistinkäsittelyyn, jolloin kaksoispuskuroinnin toteutus saadaan mahdollisemman tehokkaaksi. Lisäksi pyritään siihen, että grafiikka piirretään kokonaisuudessa keskusmuistiin ennen kuin se siirretään videomuistiin, koska videomuistin käyttäminen on hidasta [25]. Datan piirtämisen tehokuutta saadaan lisättyä käyttämällä pelkästään osoittimia kun grafiikkaa piirretään muistiin. Tällöin saadaan vähennetty ALU-operaatioita verrattuna siihen jos pikselien indeksoinnissa käytettäisiin kokonaislukuja [23].

6.2 Käyttöjärjestelmän optimoiminen

Käyttöjärjestelmän optimoinnissa keskitytään Linux-käyttöjärjestelmän optimointiin, koska Linuxin avoimuuden takia tehokas optimointi on helpoin suorittaa siinä. Linux on suunniteltu tarjoamaan jokaiselle suoritettavalle tehtävälle , sen suoritustasosta riippumatta, tasapuolisesti suoritinaikaa ja muita sen tarvitsemia resursseja [27] siten, että järjestelmän kokonaissuorituskyky olisi mahdollisimman hyvä. Reaaliaikaisuutta vaativissa järjestelmissä vastaan tästä on haittaa, sillä niissä on tarkoitus palvella tarvittaessa muutamaa prosessia mahdollisimman nopeasti ja kaikin mahdollisin keinoin, muiden prosessien kustannuksella.

Linux toteuttaa POSIX 1003.13 määrityksen mukaisen standardin; Miten monenkäyttäjän reaaliaikajärjestelmän tulee toimia muistin lukituksen, ajoituksen sekä reaaliaikasignaalien osalta. Määrityksen mukaan reaaliaikaisella suoritustasolla ajettava prosessi voidaan lukita keskusmuistiin, jotta sitä ei suoritusaikana sivutettaisi virtuaaliseen muistiin, kuten esimerkiksi kovalevylle. Ajoitin (sheduler) puolestaan huolehtii siitä, että prosessit tulevat aina suoritettua ennalta tunnetussa järjestyksessä. Vaikka POSIX 1003.13 - määrityksen ansiosta järjestelmän suorituskyky paranee huomattavasti, ei käyttöjärjestelmä silti saavuta kaikkia kovaa reaaliaikaisuutta vaativien prosessin vaatimuksia, sillä korkeamman suoritustason prosessi voi tulla syrjäytetyksi ytimessä tapahtuvien toimenpiteiden takia [27]. Tämä tarkoittaa myös sitä, että ytimen suorittaessa jotain tehtävää ei mitään prosessien pyyntöjä pystytä käsittelemään, jolloin myös

reaaliaikaprosessit joutuvat odottamaan.

Linuxin ytimen kykenemättömyys reaaliaikaisen käyttöjärjestelmän vaatimuksiin johtuu käytännössä siis siitä, että muiden käyttöjärjestelmien tapaan Linuxin ydin estää ajoittain kaikkien keskeytysten käsittelyn hyvinkin pitkäksi aikaa. Tästä aiheutuu se, että vaikka Linuxissa monien muiden käyttöjärjestelmien tapaan on tarjolla prosesseille reaaliaikainen suoritustaso, jonka omaava prosessi tulee aina suoritettua ennen muita suoritustasoja, tämäkin prosessi tulee kaikesta huolimatta syrjäytetyksi ytimen toimesta.

Mikäli reaaliaikainen värikalibrointi toteutetaan Linux-käyttöjärjestelmässä, joudutaan Linux-ytimen toimintaa muuttamaan prosessien ajoituksen kannalta. Käytännössä tämä tarkoittaa sitä, että minimoidaan käytettävien prosessien määrä ja muutetaan ajoittimen prosessille antamaan suoritinaikaa. Toisen hyvin suositun menetelmän, jolla Linuxin reaaliaikaongelma voidaan poistaa tai, jolla sitä voidaan ainakin merkittävästi parantaa, on niin sanottu pienempään viiveeseen (low-latency) pyrkivät ytimen päivitykset. Tähän ryhmään voidaan laskea kaikki menetelmät, joilla Linuxin ytimen kykyä vaihtaa suoritettava tehtävä parannetaan siten, että viive korkeamman suoritustason prosessien suoritukselle olisi mahdollisimman lyhyt. Käyttämällä ytimen versiota 2.6, joka sisältää uudistetun version ajoittimesta, saadaan huomattava tehonlisäys. Tämä uusittu ajoitin osaa säädellä tehtävän prioriteetin tehtävän käyttäytymisen mukaan. Lisäksi Linux-ydin tulee optimoida käyttämään sopivia prosessorin käskykannan laajennuksia [13,14].

6.3 Värikalibrointialgoritmien matalantason optimoiminen

Matalantason optimoinnilla tarkoitetaan sitä, että algoritmit toteutetaan käyttäen jotain prosessorin käskykannanlaajennuksia. Värikalibrointialgoritmien toteutuksen optimoinnissa tärkeimmät käskykannanlaajennukset ovat SIMD käskyt, joissa yhdessä käskyssä käsitellään useaa dataa. Optimoinnit suoritetaan käyttäen x86-arkkitehtuurin MMX-laajennusta, koska sille löytyy tuki Intelin ja myös AMD valmistamissa moderneissa prosessoreissa [5,12].

6.4 MMX-teknologian perusteet

MMX (Multimedia Extensions)-teknologian perusta on SIMD-tekniikka (Single Instruction Multiple Data), joka pystyy prosessoimaan monia sanoja samanaikaisesti yhdellä käskyllä. MMX sisältää 57 uutta käskyä, jota on suunniteltu nimenomaan videon, audion ja grafiikan prosessointiin. Nämä tuovat lisätehoa toistuvissa suorituksissa. MMX sisältää lisäksi neljä uutta tietotyyppiä ja uuden 64-bittisen MMX rekisterin, joita on käytössä kahdeksan. Multimediatoimintojen nopeuttaminen perustuu juuri siihen, että kaikille MMX rekisteriin pakatuille sanoille voidaan tehdä sama operaatio yhdellä käskyllä. Monet operaatiot, joihin MMX-komennoilla tarvitaan vain yksi kellojakso vaativat tavallisilla komennoilla useita kellojaksoja. Usein multimedia sovelluksissa käytetään toistuvia silmukoita. Saattaa olla, että silmukkaan osallistuu vain alle 10% koko ohjelmasta, mutta sen suoritus käyttää 90% koko suoritusajasta. SIMD tuo lisäsuorituskykyä nimenomaan tällaisissa tilanteissa, kun se mahdollistaa saman funktion suorittamisen useammalle tavulle [16].

MMX:n perustietotyyppi on pakattu 64-bittinen kokonaislukutyyppi, johon on pakattu monia lyhyempiä kokonaislukuja. Näitä 64-bittisiä sanoja käsitellään 64-bittisissä MMX-rekistereissä, joita on yhteensä kahdeksan. Neljä MMX-tietotyyppiä ovat:

- Packed byte: Kahdeksan tavua pakattuna 64-bitin kokonaisuudeksi
- Packed word: Neljä 16-bittistä sanaa pakattuna 64 bitin kokonaisuudeksi
- Packed doubleword: Kaksi 32-bittistä kaksoissanaa (doubleword) pakattuna 64 bitin kokonaisuudeksi
- Quadword: Yksi 64-bitin kokonaisuus

Kun värikalibrointi toteutetaan RGB-väriavaruudessa pikselit on esitetty 8-bittisinä kokonaislukuina. MMX-teknologia pystyy pakkaamaan 8 tällaista pikselitavua yhdeksi 64-bittiseksi kokonaisuudeksi ja käsittelemään sen MMX-rekisterissä. Prosessoitaessa tavuja ne voidaan siirtää kerralla rekisteristä, prosessoida samanaikaisesti ja palauttaa tulos takaisin rekisteriin. Tähän perustuu osa MMX:n tuomasta suorituskyvynlisäyksestä.

MMX-käskyt käsittää mm. seuraavat funktiot:

- Perus aritmeettiset operaatiot kuten yhteenlasku, vähennyslasku, tulo, vaihto ja kerro-lisää (multiply-add)
- Vertailuoperaatiot
- Konversiot uusien tietotyyppien välillä, tiedon pakkaus ja purku
- Loogiset operaatiot AND, NAND, NOT, OR ja XOR
- Vaihto-operaatiot (shifting)
- Tiedonsiirto käskyt, joilla voidaan käsitellä tietoa rekistereiden välillä [16]

6.5 Värikalibrointialgoritmien toteutus MMX-teknologiaa käyttäen

MMX tuo mukanaan kahdeksan uutta rekisteriä (MM0-MM7) sekä uusia komentoja, jotka käyttävät näitä rekisterejä. Nämä perustuvat olemassa oleviin liukulukurekistereihin (FP0-FP7). Moniajoa käyttävä käyttöjärjestelmä tallettaa (FSAVE) MMX-rekisterin tiedot liukulukurekistereihin, kun MMX-käskyt ovat käytössä [16,17]. MMX-dokumentaation mukaan ohjelmat tulee suunnitella siten, että liukulukulaskentaa ei käytetä kovin lähekkäin. Värikalibrointialgoritmin toteutuksessa tämä ei aiheuta ongelmaa, koska värikalibrointi toteutetaan kokonaisluvuilla. MMX-laajennettu toteutus eroaa tavallisesta toteutuksesta siinä, että pikselin jokaisen RGB-komponentin arvot käsitellään yhdellä käskyllä, koska muuttujat on sijoitettu eli pakattu MMX-rekistereihin. Lopuksi laskettu arvo puretaan MMX-rekisteristä ja tallennetaan takaisin muuttujaan.

MMX-teknologiaa käyttävän ccd-kameran värikalibrointialgoritmin pseudokielinen toteutus, ja näyttölaitteen kalibrointialgoritmi on toteutettu seuraavasti algoritmin 5 mukaisesti.

Algoritmi 5: Ccd-kameran värikalibrointi MMX-laajennettuna

```
Otetaan MMX-käyttöön

for each pixel in kamera_data
{
pakataan kamera_musta_rgb[x,y] MM0-rekisteriin
pakataan kamera_valkea_rgb[x,y] MM1-rekisteriin
pakataan kalib_data[x,y] MM2-rekisteriin
pakataan kamera_data_rgb[x,y] MM3-rekisteriin
MM5 = MM1 - MM0

MM4 = MM3 - MM0
MM4 = MM3 / MM5

MM6 = MM3 + MM2

puretaan MM6 takaisiin kamera_data_rgb[x,y] muuttujaan

}
Poistetaan MMX-käytöstä
```

Algoritmissa 5 käytetään MM6 ja MM5 rekisterejä datan säilytykseen, jotta liukuhihnaus saadaan toimimaan mahdollisemman tehokkaasti.

MMX-teknologian hyöty seuraa siitä, että ei tarvita 3 matemaattista operaatiota yhtä pikseliä kohden , vaan pikselin RGB-komponenttien arvot saadaan laskettua yhdellä käskyllä. Käytännössä tämä tarkoittaa sitä, että matemaattisia käskyjä joudutaan suorittamaan kolmasosa tavallisen toteutuksen määrästä. Tavallisessa toteutuksessa joudutaan lataamaan pikselin komponenttien arvoja kolme kertaa rekistereihin yhden pikselin käsittelyn aikana. Tämä saadaan myös vähennettyä kolmasosaan kun käytetään MMX-laajennusta. Voidaan todeta, että MMX-teknologian käyttäminen lisää huomattavasti tehoa tässä tutkielmassa toteutettuihin värikalibrointialgoritmeihin [12,16]. Moderneissa prosessoreissa, esimerkiksi AMD Athlon ja Pentium 4,MMX-teknologiaa

on kehitetty lisäämällä siihen liukuhihnausta, jolloin suorituksen tehokkuus lisääntyy. Käytännössä tämä tarkoittaa sitä, että värikalibrointiin liittyvät operaatiot suoritetaan tehokkaammin rinnakkain.

7 Värikalibrointijärjestelmän prototyypin toteutus

Ohjelmiston toteutuksessa tärkeimpänä päämääränä oli mahdollisimman tehokkaan laskennan saavuttaminen Pentium 4 prosessoria käyttävässä tietokoneessa. Ohjelmoinnissa pyritään siihen, että korkeantason c-kieli saadaan optimoitua gcc-kääntäjällä mahdollisimman tehokkaaksi, eli ohjelmansuoritus tietokoneessa olisi optimaalisesti rinnakkaistettu. Tähän on päästy käyttämällä lyhyitä käskyjä, jotta Pentium 4-prosessorin RISC-ominaisuudet saadaan tehokkaasti käyttöön sekä välttämällä ulkoisten funktioiden käyttöä suurinta laskentateho vaativissa funktioissa, lisäksi laskentaa vaativat operaatiot toteutetaan MMX-teknologiaa käyttäen, jolloin yhteen pikseliin kohdistuvat operaatiot voidaan suorittaa yhdellä operaatiolla, eikä RGB-komponenteittain. Videodatan tallentamiseen käytetään globaaleja muuttujia, jotta suorituksenaikaisia viiveitä saadaan vähennettyä. SDL ja GTK-rajapintojen yhdistämistä ei ole dokumentoitu kummassakaan rajapinnassa, vaan ohjelmistossa käytetty tapa rajapintojen yhdistämiseen on löydetty eri vaihtoehtojen tutkimisen jälkeen, samoin kahden V4L-rajapinnan kameran yhdistämistä ei ole dokumentoitu V4L-rajapinnan dokumentaatiossa. Ohjelmistossa käytetty tapa on eräs mahdollisuus, tässä toteutuksessa on pyritty saamaan mahdollisimman hyvä ajonaikainen rinnakkaisuus. Prototyypissä käytetään kahta kameraan, jotta sitä voidaan soveltaa myös stereokuvaukseen.

7.1 Yleistä

Ohjelmiston tarkoituksena on lukea dataa USB-väyliin liitetyistä kameroista ja piirtää haettu data ruudulle värikalibroituna. Ohjelmistossa ruudulle piirtäminen tapahtuu reaaliaikaisesti, tällä tarkoitetaan sitä että kuvausnopeus on vähintään 30 ruutua sekunnissa, jolloin ihminen havaitsee videon reaaliaikaisena. Jotta kuvausnopeudeksi saataisiin 30 ruutua sekunnissa, on ohjelmistossa sekä Linux järjestelmässä toteutettu useita eri optimointeja. USB-väylän (versio 1) nopeus on varsin hidas 12 Mbit/s, joten kuvadatan hakeminen kameroilta vie suurimman osan siitä ajasta, jota on käytettävissä yhtä ruutua kohden. Datan lukeminen USB-väylän kautta kameroista on optimoitu seuraavasti: Lukeminen toteutetaan siten, että kamerat kytketään omiin USB-väyliin, jolloin kummallakin kameralla on käytössä 12 megabitin kaistanleveys. Data kamerasta suoraan 24 bittisenä RGB-esityksenä. Algoritmin optimoiminen perustuu laskennan suorittamiseen rinnakkaisesti MMX-teknologiaa hyödyntäen. Lisäksi optimoidaan

käyttöjärjestelmää, jotta viive saadaan mahdollisimman pieneksi.

7.2 Korjaustaulukon laskeminen

Matlab-ohjelmalla lasketaan spektridatasta RGB-esitykset, RGB-esityksien laskemisessa käytettävästä spektridatasta on poistettu valonlähde jakamalla spektridatat valkoisen referenssin spektrillä. Kameran ja spektrien RGB-arvot tallennetaan 24 soluiseen taulukkoon. Spektridatan RGB-arvot järjestetään nousevaan järjestykseen intensiteetin mukaisesti jokaiselle RGB-komponentille erikseen. RGB-esitykset normitetaan välille 0-1. Kameran RGB-taulukko järjestetään spektridatan järjestelemisessä olleiden solujen siirtojen mukaisesti. Tämän jälkeen lasketaan tarkan spektridatasta lasketun ja kameran RGB-arvojen väliset erot, eli muodostetaan korjaustaulukko, koska mitattuja arvoja on vain 24 kappaletta ja RGB-värikalibrointiin tarvitaan 256 kappaletta arvoja joudutaan korjaustaulukko interpoloimaan 256 soluiseksi. Tämä suoritetaan käyttämällä spline-sovitusfunktiota. Kameran RGB-kuvat on tallennettu bmp-formaatissa.

7.3 Kalibroinnin suorittaminen

Värikalibroinnissa luetaan korjaustaulukko tekstitiedosta muistiin. Korjaustaulukko on laskettu Matlab-ohjelmalla. Korjausdatan laskemiseen tarvittava, kameroilla kuvattu data, tallennetaan BMP-muodossa videodatasta. BMP-muotoiset kuvat tallennetaan videodatasta SDL-rajapintaan käyttäen. Värikalibroinnissa käytetään 32-bittistä tallennusmuotoa kuvadatalle, koska 32-bittinen esitys on nopeampaa kuin 24-bittinen, mikäli X-windows järjestelmässä käytetään 32-bittistä tilaa. Kuvadata luetaan kameroilta 24-bittisenä. Muunnos eri tallennusmuotojen välillä tehdään SDL-rajapinnan avulla. Korjaustaulukon arvot on tallennettu 32-bittisenä RGB-esityksenä SDLpiirtopintaan. Korjaustaulukossa voi olla myös negatiivisia arvoja, jotka kuintenkin tallennetaan arvot positiivisina arvoina korjaustaulukkoon. Tiedot niistä taulukon kohdista, joissa on negatiivisia arvoja on tallennettu bittivektoriin. Jokaisella RGB-komponentilla on oma bittivektorinsa. Värikalibrointi suoritetaan bittioperaatioina, eli siinä käytetään vain bittien yhteen- ja vähennyslaskua sekä bittien siirtämistä. Tällöin laskenta voidaan suorittaa mahdollisimman pienellä viiveellä, koska hitaita operaatioita, esim. kertolasku tai jakolasku, ei tarvitse suorittaa. Korjaustaulukon käsittelyä varten kuvadatan RGB-komponenttien arvot muutetaan kokonaisluvuiksi, tähän käytetään SDL-

piirtopinnan sisältämiä tietoja eri komponenttien pituuksista bitteinä. Värikalibroinnin toteutus on esitetty kuvassa 11. Kummaltakin kameralta tuleva väridata kalibroidaan erikseen, omilla korjaustaulukoilla. Kalibrointidataan tulevat laskutoimitukset esim. valkoisella referenssillä jakamisen jälkeen normitus, on laskettu valmiiksi valkoiseen referenssidataan.

7.4 Ohjelmiston pääosat

7.4.1 MMX-laajennuksen käyttäminen värikalibroinnissa

MMX-laajennettu toteutus eroaa tavallisesta toteutuksesta siinä, että pikselin jokaisen RGB-komponentin arvot käsitellään yhdellä käskyllä, koska muuttujat on sijoitettu eli pakattu MMX-rekistereihin. Lopuksi laskettu arvo puretaan MMX-rekisteristä ja tallennetaan takaisin muuttujaan. MMX-operaatiot on toteutettu inline-komentoina c-kieliseen lähdekoodiin. Esimerkiksi GNU C-compiler kääntäjää käytettäessä tämä tarkoittaa seuraavaa:

Esimerkki 1: MMX-käskyn käyttäminen GNU C-compiler kääntäjässä

```
__asm__("Assembler käsky"   : tulos_muuttujat :
syöte_muuttujat : mmx_rekisterit);
```

Esimerkkinä MMX-toteutusta operaatiosta on esitetty kuvassa 15, jossa toteutetaan kahden SDL-piirtopinnan bittien yhteenlasku, 8-bitillä. MMX-teknologiasta hyödynnetään saturoimisoperaatioita, eli mikäli yhteenlaskun tulos ylittää kokonaislukuna 255, niin ei tapahdu ylivuotoa vaan tulokseksi tulee 255.

```
1 int SDL_AddMMX(unsigned char *Src1, unsigned char *Src2, unsigned char *Dest, int length)
2 {
3     asm volatile
4      ("pusha                  \n\t" "mov  %2, %%eax \n\t"// Ladataan Src1 osoite eax-rekisteriin
5       "mov          %1, %%ebx \n\t"     // Ladataan Src2 osoite ebx-rekisteriin
6       "mov          %0, %%edi \n\t"     // Ladataan Dest osoite  edi-rekisteriin
7        "mov          %3, %%ecx \n\t"    // Ladataan laskuri muuttuja (SIZE)  ecx-rekisteriin
8       "shr          $3, %%ecx \n\t"     // laskuri/8 (MMX lataa 8 tavua kerralla)
9        ".align 16               \n\t"    // 16 tavun varaus silmukkaa varten
10      ".L1010:                \n\t" "movq    (%%eax), %%mm1 \n\t" // Lataa 8 tavua Src1:stä
  mm1-rekisteriin
11      "paddusb (%%ebx), %%mm1 \n\t"     // mm1=Src1+Src2 (8 tavun yhteenlasku, saturaatiolla)
12      "movq    %%mm1, (%%edi) \n\t"     // Tallennetaan tulos Des:ii
13       "add          $8, %%eax \n\t"    // Kasvatetaan Src1, Src2 ja Dest osoittimia
14      "add          $8, %%ebx \n\t"     // kahdeksalla
15      "add          $8, %%edi \n\t" "dec              %%ecx \n\t" // pienennetään laskuria
16      "jnz             .L1010 \n\t"     // Tarkastetaan päätttyykö silmukka
17       "emms                    \n\t"    // Poistutaan MMX-tilasta
18      "popa                   \n\t":"=m" (Dest) // %0
19      :"m"(Src2),               // %1
20      "m"(Src1),                // %2
21      "m"(length)               // %3
22        );
23    return (0);
24 }
```

Kuva 15: Yhteenlaskun toteuttaminen MMX-teknologialla

7.4.2 Datan lukeminen kameroilta (Video4Linux)

Ohjelman käynnistyttyä avataan tiedostokahvat kameroille. Seuraavaksi luetaan kameroiden asetukset, joissa on tallennettu valotusaika, kuvadatan bittisyys, sekä kuva-alueen koko. Asetusten hakemisen jälkeen muokataan asetuksia ja tallennetaan asetukset kameraan. Valotusaika säädetään kirkkaus/kontrasti-säädöillä. Kuvadatan lukeminen tapahtuu ioctl-protokollaa käyttäen. Merkkijonolaitteesta (/dev/videoX) luetaan halutun kuvan kokoinen alue, jonka koko on $3 * leveys * korkeus$ (24-bittinen data). Luettu data tallennetaan 32-bittiseen kokonaislukutaulukkoon. Kummallakin kameralla on omat muuttujat kaikkia operaatiota varten. Ohjelmaa lopetettaessa suljetaan tiedostokahvat kameroihin. 24-bittisestä datasta muuttaminen 32-bittiseen tapahtuu automaattisesti SDL-rajapinnassa, tällöin esitys on RGBA, jossa A on alpha-kanava eli läpinäkyvyys, muunnoksen jälkeen A-komponenttien arvo on 0 [18].

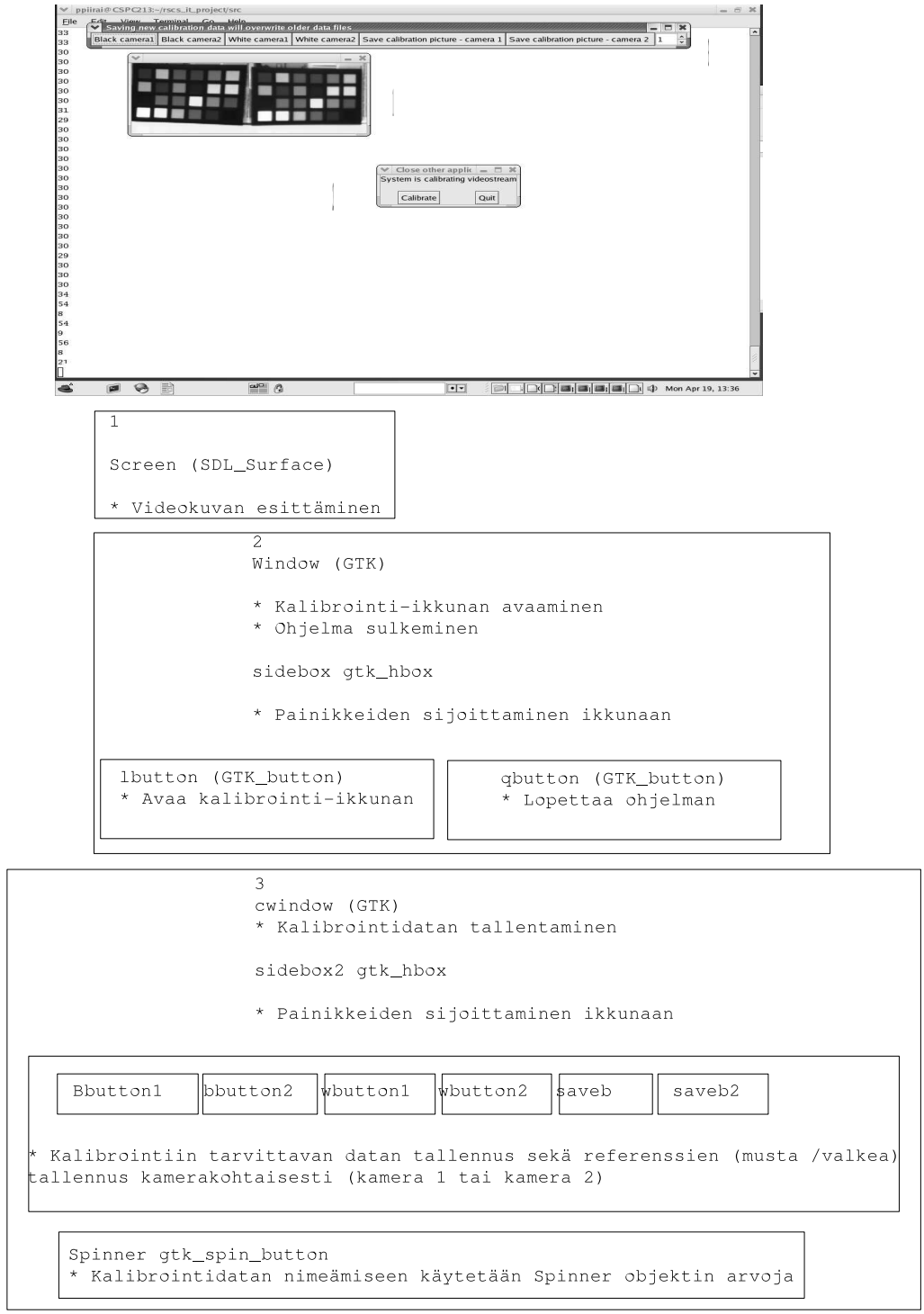
7.4.3 Videon piirtäminen ruudulle (SDL)

SDL-rajapintaa käytetään videokuvan piirtämisen ruudulle, sekä kalibrointiin tarvittavan kameran kuvadatan tallentamiseen. SDL-rajapinta tallentaa kuvadatan SDL_Surface-piirtopintoihin, jotka sisältävät myös on tiedot datan koosta ja bittien lukumäärästä. SDL_Surface-piirtopinnat tallennetaan keskusmuistiin, lukuun ottamatta sitä piirtopintaa, joka esittää videokuvaa ruudulla, joka on tallennettu videomuistiin. Tämä on tehty siksi,että videomuistin käsitteleminen on hidasta. Kuva kameroil-

ta haetusta datasta muodostetaan siten, että kopioidaan merkkitaulukko johon V4L-rajapinnalla kameroilta haettu data on tallennettu SDL_Surface-piirtopintaan pikselidatan tilalle. Tässä käytettävä värisyvyys on 24-bittiä, kun ruudulle piirrettävät elementit ovat 32-bittisiä. Kameroiden datasta muodostetaan aluksi omat SDL_Surface elementit, värikalibrointia varten nämä elementit yhdistetään uudeksi SDL_Surface elementiksi, tämä tapahtuu käyttämällä SDL_BlitSurface-metodia, joka yhdistää kahden elementin pikselien RGB-arvot. Kuvan piirtäminen ruudulle on esitetty kuvassa 11.

7.4.4 Graafinen käyttöliittymä (GTK)

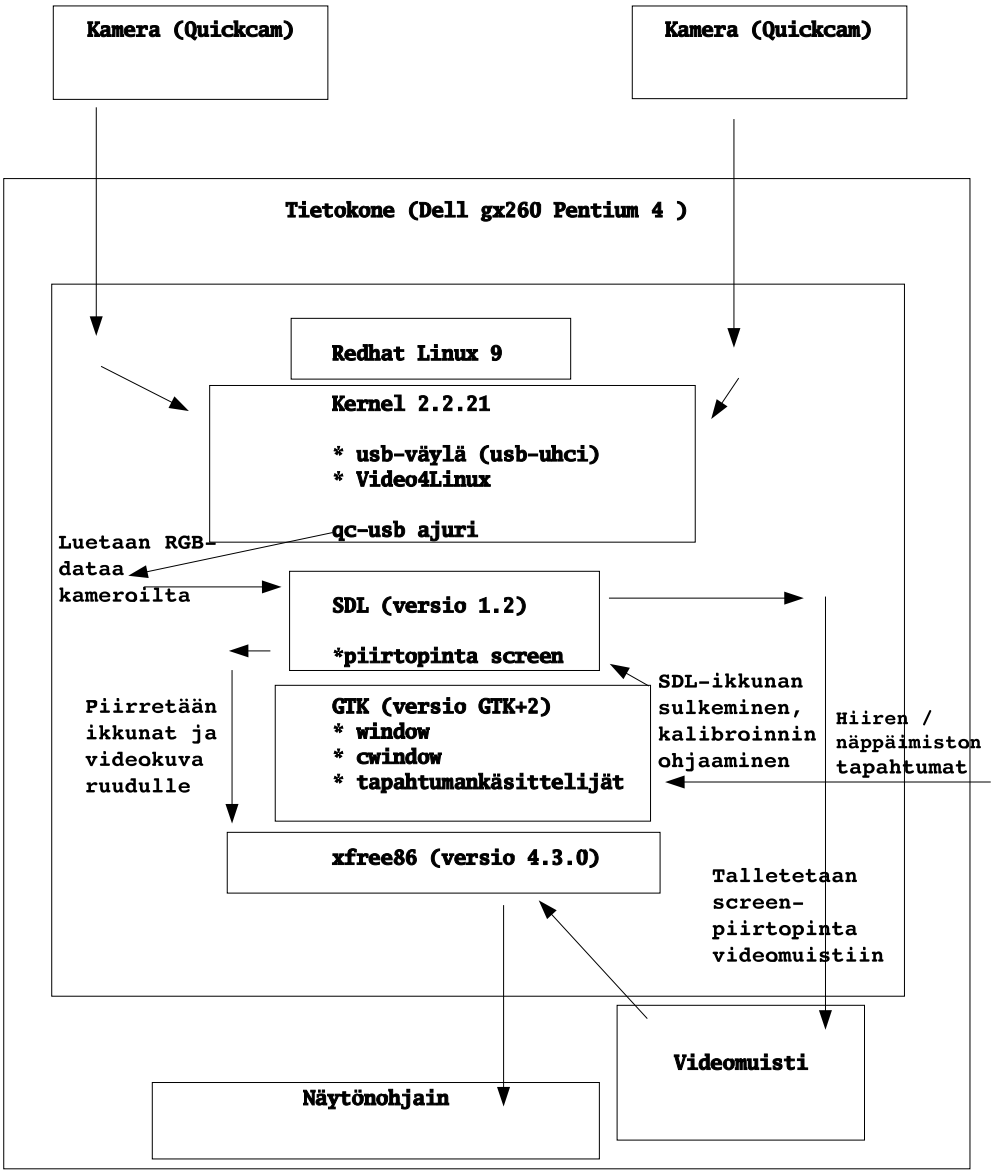
Ohjelman graafinen käyttöliittymä on toteutettu käyttäen GTK-rajapintaa, käyttöliittymä sisältää kaksi ikkunaa joissa on painonappeja sekä nimiöitä. Kuvien nimeämistä varten käytetään liukuvaa valintapainiketta. Painonappien tapahtumankäsittelijät ohjaavat järjestelmän toimintaa, ohjelman sulkemista, kalibrointiin tarvittavien kuvadatojen tallennuksen. Tapahtumankäsittelijät ohjaavat järjestelmää lippumuuttujien avulla, joiden arvoa muutetaan 0 tai 1. Graafisen käyttöliittymän rakenne on esitelty kuvassa 16. SDL-rajapinta on yhdistetty GTK-rajapintaan siten, että GTK-tapahtumasilmukassa kutsutaan jatkuvasti SDL-tapahtumasilmukkaa. Tähän käytetään GTK-rajapinnan gtk_idle_add-funktiota.



Kuva 16: Graafisen käyttöliittymän rakenne [18].

7.5 Laitteisto

Laitteiston ja ohjelmistossa käytettyjen rapapintojen yhteys on esitetty kuvassa 17



Kuva 17: Laitteisto sekä ohjelmistorajapinnat.

8 Esitellyn värikalibrointimenetelmän arviointi

Tutkielmassa esitelty menetelmä mahdollistaa värikalibroinnin suorittamisen siten, että videokuvassa suurin ruutujen välinen aika on 30 ms. Nykyisillä PC-tietokoneilla voidaan suorittaa myös monia muita kuvankäsittelyyn pohjautuvia menetelmiä, joilla pyritään saavuttamaan mahdollisimman hyvälaatuinen värikuva [31, 32]. Mikäli nämä menetelmät optimoidaan esimerkiksi käyttämää SIMD-käskykantoja, voidaan toteuttaa monia videosovelluksia, joiden väriesitystä on korjattu. Tällaisia menetelmiä on valkotasapainon säätäminen ja histogrammin tasapainottaminen tai gamma-korjaus [32]. Kuitenkin näissä tekniikat perustuvat siihen, että kuvaa korjataan kuvan sisältämän RGB-väriesityksen pohjata. Lisäksi näillä tekniikoilla pyritään siihen, että kuva näyttäisi mahdollisimman hyvältä, esimerkiksi että värit ovat tasapainossa. Tällöin ei kuitenkaan saavuteta tarkkaa väriesitystä siitä, miltä alkuperäinen kohde näytti kuvausolosuhteissa.

Tässä tutkielmassa esitellyt värikalibrointialgoritmit pyrkivät siihen, että kuvatun kohteen värit pyritään esittämään mahdollisimman tarkasti digitaalisessa muodossa. Lisäksi algoritmit eroavat kuvankäsittelymenetelmistä siinä, että niissä otetaan käyttöön ulkopuolista informaatiota, jotka ei sisälly kameralta saatuun kuvadataan. Tällöin voidaan yksilöidä erilaisten kuvauslaitteiden virheet väriesityksessä ja korjata ne. Korjauksessa pyritään siihen, että saavutettaisiin mahdollisimman tarkka väriesitys kuvattavasta kohteesta. Tarkkuus saavutetaan sillä, kun referenssiarvoina käytetään kohteen spektristä laskettuja RGB-arvoja, tällöin voidaan myös huomioida kohteessa oleva valaistus. Kun kalibrointidata on selvitetty, lisätään sen sisältämä informaatio kuvadataan kun suoritetaan värikalibrointi, eli tällöin voidaan laajentaa kuvauslaitteen ominaisuuksia.

Tutkielmassa toteutetussa prototyypissä oli tarkoitus osoittaa se, että edullisella laitteistolla voidaan saavuttaa ohjelmallisesti sellaisia ominaisuuksia, joiden avulla prototyyppi vastaa kalliimpaa laitteistokokonaisuutta. Prototyypissä parannettiin halpojen ccd-kameroiden kuvauslaatua värikalibroimalla niiden luoma data, tällöin poistetaan ohjelmallisesti ne häiriöt, jotka muuten vaatisivat parempilaatuista laitteistoa. Lisäksi prototyypissä toteutettiin reaaliaikainen signaalinkäsittely tavallisen prosessorin avulla, käyttäen hyödyksi prosessorin multimediakäyttöön tarkoitettuja ominaisuuksia. Nämä ominaisuudet soveltuvat hyvin värikalibrointiin, koska siinä käytetyt operaatiot ovat tietokoneen kannalta samanlaisia kuin esimerkiksi tietokonepelien visuaaliset

efektit.

MMX-laajennusta käytettäessä saavutetaan huomattava tehonlisäys perinteisiin menetelmään verrattuna. Esimerkiksi tarkastellaan tilannetta yhden 320*200 pikselin kokoisen 24-bittisen kuvan värikalibroinnin osalta. Taulukossa 3 on esitelty käskyjen teoreettiset määrät kuvan värikalibroinnin suorittamista varten, latauskäskyssä huomioidaan myös kalibrointidatan lataaminen [16].

Taulukko 3: Käskyjen teoreettiset lukumäärät Intel Pentium prosessorilla

Käsky	Operaatioita	Ei MMX	MMX käytössä
Lataa	320*200*3*2	384000	96000
Avaus (UnPack)	–	–	96000
Kertolasku	320*200*3	192000	48000
Jakolasku	320*200*3	192000	48000
Tallenna	320*200*3	192000	48000
Yhteensä		960000	336000

Taulukosta 3 huomataan, että MMX-teknologiaa käytettäessä käskyjä tarvitaan vain 35% tavallista suorituksesta. Eli MMX-teknologiaa käyttäen saadaan värikalibroinnin nopeus kolmikertaiseksi. MMX on jo teknologiana varsin vanha, kuitenkin se on SIMD-laajennuksista ainoa, jolla toteutetut ohjelmat toimivat sekä Intelin ja AMD:n prosessoreissa.

Kameran värikalibroinnin toimintaa voidaan tarkastella kuvaamalla esimerkiksi Macbeth 24 referenssin värireferenssiä ja värikalibroimalla tätä kuvadataa. Värikalibroinnin toiminnan tehokkuus saadaan selville kun värikalibroitua kuvaa verrataan Macbeth 24 värireferenssin spektrikuvasta, samoilla valaistustiedoilla, laskettuun RGB-kuvaan. Kuvassa 18 on esitetty tulokset prototyypin kameroilla kuvastusta ja värikalibroidusta Macbeth 24 värireferenssistä ja vastaavasta spektrikuvan RGB-esityksestä.

Värireferenssin RGB-arvot on esitetty kuvan 19 tilastoissa. Kun tarkastellaan kuvan 19 tilastoja havaitaan, että prototyypin värikalibrointi toimii, koska vertaamalla värikalibroidun kuvan RGB-arvoja spektrinkuvan RGB-arvoihin havaitaan, että värikalibroinnilla saavutetaan oikean valaistusolosuhteen saavuttaminen RGB-kuvaan. Spektrikuvan RGB-esityksen ja värikalibroidun kuvan RGB-arvojen keskiarvojen erotus on alle 20. Vaikka RGB-erot on kuvista havaittavissa, on tärkein, että oikean valaistuksen

Kuva 18: Värikalibroinnin tulokset kameralle.

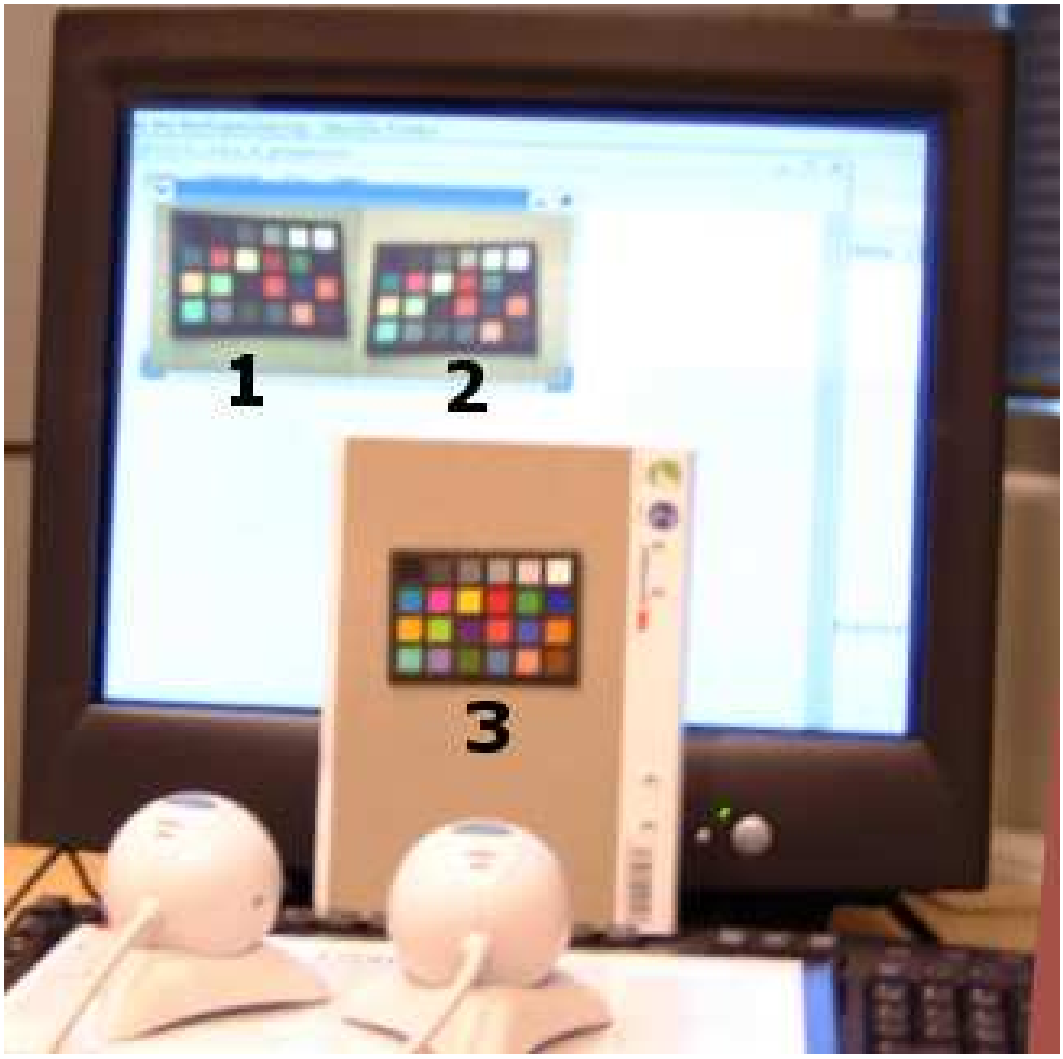
saavuttaminen kuitenkin onnistunut, joten voidaan todeta, että prototyypin kameroiden värikalibrointi toimii oikein.

	Alkuperäinen kuva R G B	Spektrikuvasta laskettu RGB-kuva R G B	Värikalibroitu kuva R G B	Värikalibroidun kuvan virhe R G B
1.	33 21 9	111 89 68	98 73 49	13 16 19
2.	255 155 63	197 166 129	209 167 121	-12 -1 8
3.	81 89 49	119 127 122	112 122 114	7 5 8
4.	58 68 22	105 116 73	100 112 68	5 4 5
5.	133 106 64	147 139 140	144 134 131	3 5 9
6.	144 219 92	156 190 150	133 201 126	23 -11 24
7.	255 128 39	195 137 76	201 135 67	-6 2 9
8.	50 50 41	100 102 120	90 94 113	10 8 7
9.	255 84 36	172 110 98	184 109 94	-12 1 4
10.	45 28 17	90 75 83	80 65 76	10 10 7
11.	214 248 68	182 191 84	193 208 75	-11 -17 9
12.	255 225 64	218 177 85	233 188 71	-15 -11 14
13.	25 23 23	66 69 96	48 50 76	18 19 20
14.	79 131 40	117 147 80	102 141 63	15 6 17
15.	213 50 22	139 78 66	155 66 52	-16 12 14
16.	255 255 95	237 214 99	254 235 98	-17 -21 1
17.	252 83 52	164 111 124	184 103 110	-20 8 14
18.	57 95 59	90 127 122	74 117 106	16 10 16
19.	255 255 228	255 253 215	255 255 225	0 -2 -10
20.	255 254 153	221 212 181	234 227 178	-13 -15 3
21.	214 182 82	177 169 142	182 172 127	-5 -3 15
22.	112 96 42	136 131 109	141 135 110	-5 -4 -1
23.	53 45 20	95 91 75	76 73 53	19 18 22
24.	22 18 7	58 56 45	49 46 36	9 10 9

Kuva 19: Värireferenssien RGB-arvojen keskiarvot kameran värikalibroinnissa.

Seuraavaksi tarkastellaan prototyypin näyttölaitteen värikalibrointimenetelmän toimintaa. Kun kameran kuvadata on värikalibroitu vastaamaan oikeaa väriesitystä. Näyttölaitteen värikalibroinnin toiminnan arviointi suoritettiin kuvassa 20 esitetyllä koejärjestelyllä, jossa kuvataan kameralla yhtäaikaisesti Macbeth 24 värireferenssiä ja näyttölaitteelta myös värikalibroitua kuvaa samasta referenssistä. Näyttölaitteen värikalibroinnin tulokset on esitetty kuvassa 21, sekä kuvan 21 värireferenssien RGB-arvojen keskiarvot on esitetty tilastoina kuvassa . Lisäksi tilastoissa ja kuvassa 22 on esitetty spektrikuvasta laskettu sRGB-esitys, johon värikalibroinnin tulosta voidaan verrata.

Kun värikalibroimattomalla kameralla kuvataan näyttölaitetta ja referenssiä ei kuvaan saadaan oikeaa väriesitystä vaan kuva sisältää käytetyn kameran systeemifunktion, kuitenkin kuvasta voidaan tarkastella RGB-arvojen eroa ilman värikalibrointia ja värika-



Kuva 20: Koejärjestely näyttölaitteen värikalibroinnin tulosten selvittämiseksi. Kamera 1 on vasemmalla ja kamera 2 oikealla.

Kuva 21: Värikalibroinnin tulokset näyttölaitteelle.

librointia käyttäen. Tästä saadaan selville se, kuinka värikalibrointi korjaa kuvaa vastaamaan todellisen värireferenssin värejä samassa valaistussa. sRGB-esitys vastaa sitä, mihin näyttölaitteen valmistaja on pyrkinyt kalibroimaan näytön, tosin tähän vertaamisessa joudutaan huomioidaan kuvaamiseen käytetyn kameran systeemifunktion vaikus. Kun tuloksia tarkastellaan havaitaan, että värikalibroinnin avulla saavutetaan valaistusta vastaava väriesitys näyttölaitteella. Suuret virheet RGB-arvojen erotuksissa johtuvat näytön kirkkauden vaikutuksesta kuvauksessa käytettyyn kameraan. Kuitenkin tuloksista voidaan todeta, että näyttölaitteen värikalibrointi toimii halutulla tavalla.

	Alkuperäinen kuva			**Väristandardi**			**Värikalibroitu kuva**			**Värikalibroidun kuvan virhe**			**sRGB**			**sRGB virhe**		
	R	**G**	**B**	**R**	**G**	**B**	R	G	B	R	G	B	R	G	B	R	G	B
1.	81	76	96	51	32	26	63	59	75	12	27	49	59	60	59	4	-1	16
2.	85	83	98	98	67	50	78	64	60	-20	-3	10	90	91	89	-12	-27	-29
3.	108	122	122	137	109	85	110	103	92	-27	-6	7	125	127	124	-15	-24	-32
4.	154	179	158	183	147	119	155	159	145	-28	12	26	160	161	158	-5	-2	-13
5.	233	250	232	234	185	155	212	208	194	-22	23	39	198	200	197	14	8	-3
6.	232	254	253	255	226	183	230	236	231	-25	10	48	237	237	231	-7	-1	0
7.	79	129	148	56	121	151	56	112	136	0	-9	-15	0	142	164	56	-30	-28
8.	209	111	125	244	74	117	183	90	127	-61	16	10	181	95	153	2	-5	-26
9.	233	255	203	255	190	30	233	212	105	-22	22	75	237	198	46	-4	14	59
10.	185	84	91	246	38	27	201	49	50	-45	11	23	172	60	68	29	-11	-18
11.	90	150	125	93	142	43	79	132	70	-14	-10	27	84	154	76	-5	-22	-6
12.	78	72	102	40	40	132	47	43	101	7	3	-31	53	72	151	-6	-29	-50
13.	248	220	155	255	159	24	228	178	92	-27	19	68	227	159	56	1	19	36
14.	164	246	171	181	181	35	162	204	96	-19	23	61	168	186	68	-6	18	28
15.	83	75	98	101	35	84	76	60	92	-25	25	8	97	71	112	-21	-11	-20
16.	232	108	103	255	75	58	199	88	93	-56	13	35	187	93	104	12	-5	-11
17.	79	87	123	82	75	154	68	68	126	-14	-7	-28	80	102	170	-12	-34	-44
18.	244	138	111	255	114	13	213	116	73	-42	2	60	211	121	58	2	-5	15
19.	124	235	200	132	178	139	116	195	159	-16	17	20	107	192	170	9	3	-11
20.	128	138	153	157	115	146	130	116	138	-27	1	-8	131	135	175	-1	-19	-37
21.	85	108	98	102	102	15	79	92	49	-23	-10	34	95	113	74	-16	-21	-25
22.	93	119	130	110	111	131	78	109	126	-32	-2	-5	94	127	155	-16	-18	-29
23.	233	166	142	246	129	87	204	147	127	-42	18	40	194	153	131	10	-6	-4
24.	101	81	92	141	61	14	111	69	51	-30	8	37	116	85	73	-5	-16	-22

Kuva 22: Värireferenssien RGB-arvojen keskiarvot näyttölaitteen värikalibroinnissa.

9 Yhteenveto

Työssä esiteltiin ja toteutettiin digitaalinen menetelmä, jonka avulla korjataan kameran ja näyttölaitteen väriesityksen virheitä värikalibroinnin avulla. Värikalibroinnilla saavutetaan tarkka RGB-väriavaruutta käyttävä väriesitys, jossa huomioidaan myös valaistusolosuhteet, eli pyritään korjaamaan väriesitys todellisuutta vastaavaksi. Värikalibrointi perustuu siihen, että haetaan kameran ja näyttölaitteen systeemifunktiot, eli poikkeamat ideaalisesta tapauksesta, ja pyritään minimoimaan niiden vaikutus. Esitelty värikalibrointimenetelmä on ohjelmallinen ja se voidaan toteuttaa reaaliaikaisesti, eli videokuvan esitysnopeus on värikalibroinnista huolimatta vähintään 30 ruutua sekunnissa. Värikalibroinnin reaaliaikaisuus saavutetaan matalantasonoptimoinnilla. Tutkielmassa toteutetussa prototyypissä tämä optimointi suoritettiin käyttöjärjestelmää virittämällä sekä toteuttamalla ohjelmisto käyttäen SIMD-käskykantoja. Prototyypin toteutuksessa käytettiin MMX-teknologiaa sekä optimoitiin Linux-käyttöjärjestelmää. Värikalibroinnissa saavutettiin valaistusolosuhteet huomioiva korjaus, lisäksi kalibrointiin tarvittavat systeemifunktion mittaukset voitiin toteuttaa varsin yksinkertaisesti, joka mahdollistaa niiden automatisoinnin. Esitellyn värikalibrointimenetelmän ongelmana on se, että siinä käytetään RGB-väriavaruutta, jolloin väriesityksen tarkkuus riippuu siitä, kuinka suuren määrän värejä laite voi RGB-väriavaruudessa tuottaa. Prototyypin ongelmana on se, että tutkielmassa esitelty versio ei sisällä laitetta, jolla valaistusinformaatio voitaisiin tuoda reaaliaikaisesti järjestelmään, vaan prototyypin värikalibrointijärjestelmä toimii vain olosuhteissa, joissa valaistus pysyy vakiona. Jatkotutkimus valaistusinformaation reaaliaikaiseen mittaukseen ja sen yhdistämiseen toteutettuun prototyyppiin on käynnissä.

Viitteet

[1] G. B. Arfken and H. J. Weber, Mathematical methods for physicists (4th edition), Academic Press, 1995.

[2] R. C. Dorf (Editor in Chief), The Electrical Engineering Handbook (Second Edition), CRC Press, IEEE Press, 1997.

[3] F. Elliot (Editor), Handbook of Digital Signal Processing (Engineering Applications), Academic Press, San Diego, 1987.

[4] A. V. Oppenheim, A.S. Willsky and S. Hamid Nawab, Signals & Systems (Second Edition), Prentice Hall, 1995.

[5] J. L. Hennessy, D. A. Patterson, Computer Architectures. A Quantitative Approach, Morgan Kaufmann Publishres, 2002.

[6] G. E. Healey and R. Kondepudy, CCD Camera calibration and noise estimation, IEEE Transactions on Patter Analysis and Machine Intelligence, vol. 16, no. 3, pp. 267 - 276, March. 1994.

[7] G. E. Healey and R. Kondepudy, Radiometric CCD Camera calibration and noise estimation, IEEE Transactions on Patter Analysis and Machine Intelligence, vol. 16, no. 3, pp. 267 - 276, March. 1994.

[8] Y. Chang and J. Reid, RGB Calibiration for Color Image Analysis in Machine Vision, IEEE Transactions on Image Processing, vol. 5, no. 10, pp. 1414 - 1422, October. 1996.

[9] M. Vrhel and H. Trussel, Color Device Calibration A Mathematical Formulation, IEEE Transactions on Image Processing, vol. 8, no. 12, pp. 1796 - 1812, December. 1999.

[10] Y. V. Haegehen and J. M. Naeyaert, An Imaging Sytem with Calibrated Color Image Acquistition for Use in Dermatology,IEEE Transactions on Medical Imaging, vol. 19, no. 7, pp.722 - 730, Jul. 2000.

[11] G. Sharma, LCDs Versus CRTs-Color-Calibration and Gamut Considerations,Proceedings of the IEEE, vol. 90, no. 4, pp.605 - 622, Apr. 2002.

[12] R. Bhargava, L.K. John, B.L. Evans, R. Radhakrishnan, Evaluating MMX technology using DSP and multimedia applications, 31st Annual ACM/IEEE International Symposium on Microarchitecture, pp. 37- 46, 1998.

[13] A. Peleg, U. Weiser, The MMX technology Extension to the Intel Architecture, IEEE Micro, vol 16, no 4, pp. 42 - 50, Aug. 1996.

[14] A. Perera , T. Sundic, A portable electronic nose based on embedded PC technology and GNU/Linux: hardware, software and applications Sensors Journal, IEEE , Volume: 2 , Issue: 3 ,pp. 235 - 246, June 2002

[15] T. Nakajima, M. Iwasaki, S. Ochiai, Issues for making Linux predictable Applications and the Internet (SAINT) Workshops, 2002. Proceedings. 2002 , 28 Jan.-1 Feb., pp. 8 - 14, 2002

[16] O. Lempel, A. Peleg, U. Weiser, Intel's MMX(TM) technology-a new instruction set extension, Compcon '97. Proceedings, IEEE , pp. 255 - 259,Feb. 1997

[17] Greene, M.A., Pentium(R) processor with MMX(TM) technology performance, Compcon '97. Proceedings, IEEE , pp. 263 - 267,Feb., 1997

[18] P.O., Piirainen, Reaaliaikainen stereokuvaus- ja värikalibrointijärjestelmä, Erikoistyö, Joensuun yliopisto, 2003

[19] IA-32 Intel Architecture Optimization, Reference Manual, Intel, 2003

[20] Fast Color Conversion Using Streaming SIMD Extensions and MMX technology, Application note, Intel, 2003

[21] The HP HDCS Family of CMOS IMage Sensors, Product Techinal Specification Revision 3.0, Hewlett-Packartd Company, 1998

[22] M. J. Vrehl H.J Trussel, The Mathematics of Color Calibration,International Conference on Image Processing, 1998

[23] F. Agner, How to optimize for the Pentium family of microprocessors, `www.agner.org/assem/pentopt.pdf`, 2004

[24] AMD Extensions to the 3DNow! and MMX Instruction Sets Manual, Design Guide, Advanced Micro Devices Inc., 2000

[25] Linux Kernel Issue with Systems Using AGP Graphics, Application Note, Advanced Micro Devices Inc., 2002

[26] AMD Athlon Processor x86 Code Optimization Guide, Design Guide, Advanced Micro Devices Inc., 2002

[27] Jr. Ortiz , Embedded OS s gain the inside track, Computer, Vol. 34, No 11, pp. 14 - 16, IEEE Computer Soc. 2001

[28] Lennon, Embedding Linux, IEEE Review, Vol. 47, Issue 3, pp: 33 - 37, 2001

[29] A. Deshpande, Linux Kernel 2.6: the Future of Embedded Computing, www-sivu, `http://www.linuxjournal.com//article.php?sid=7477`, 2004, haettu 1.4.2004

[30] LM9637 Monochrome CMOS Image Sensor VGA 68 FPS, Application Note, National Semiconductor, 2002

[31] G. Wyszecki, W.S. Stiles, Color Science: Concepts and Methods, Quantitative Data and Formulae, 2nd ed., Johd Wiley & Sons, 1982

[32] Camera Imaging Chain, www-sivu, `http://www.nokia.com/nokia/0,,54238,00.html`, Nokia, 2004, haettu 5.4.2004

[33] R.E. Gonzalez, R.E. Woods, Digital Image Processing, Addison-Wesley, 1992.