Reunantunnistusalgoritmeista

Mika Hannula

Tampereen yliopisto
Informaatiotieteiden yksikkö
Tietojenkäsittelyoppi
Pro gradu -tutkielma
Ohjaaja: Martti Juhola
Kesäkuu 2014

Tampereen yliopisto
Informaatiotieteiden yksikkö
Tietojenkäsittelyoppi
Mika Hannula: Reunantunnistusalgoritmeista
Pro gradu -tutkielma, 40 sivua
Kesäkuu 2014

Reunantunnistuksessa pyritään etsimään digitaalisista kuvista erilaiset alueet erottavaa rajaviivaa. Tyypillisesti tällainen raja on kohdassa, jossa värissä tapahtuu huomattava muutos. Reunantunnistus on tärkeä esiprosessointivaihe monissa digitaalisia kuvia käsittelevissä tehtävissä.

Tässä tutkielmassa käyn läpi reunantunnistuksen perusteita, esittelen tunnetuimmat reunantunnistusalgoritmit ja selvitän, miten reunantunnistusalgoritmeja on vertailtu ja millaisiin tuloksiin on päädytty.

Avainsanat ja -sanonnat: digitaaliset kuvat, reunat, reunantunnistus, algoritmit, algoritmien vertailu.

Sisällys

1. Johdanto

Reunantunnistus on eräs kuvanprosessoinnin osa-alue. Reunantunnistuksessa pyritään etsimään digitaalisesta kuvasta reunaviivat, jotka erottavat erilaiset alueet toisistaan. Kohdassa, jossa alue muuttuu toiseksi, tapahtuu muutos pikseleiden väreissä ja yksinkertaisimmillaan etsitään vain sellaiset kohdat kuvasta. Kuvassa 1.1 on esimerkki valokuvasta ja siitä etsityistä reunoista.

Yksinkertainen värin muutoksen tarkastelu ei kuitenkaan tuota hyvää lopputulosta, jos kuvassa on häiriötekijöitä. Alueiden välinen raja saattaa olla häilyvä, jos väri muuttuu toiseksi asteittain. Entä jos kuvassa on mies, jolla on ruudullinen paita? Onko hyvä tulos silloin miehen ääriviivat vai myös ruudut paidassa?

Reunantunnistus on ongelmana erittäin haasteellinen ja sitä on tutkittu jo yli 50 vuotta. Reunojen löytämisellä kuvasta ei ole sellaisenaan juurikaan arvoa, mutta reunantunnistusta käytetään esiprosessointina monissa sovelluksissa ja se tekee siitä tärkeän.

Kuva 1.1: Esimerkki valokuvasta ja siitä tunnistetuista reunoista. Alkuperäinen kuva tunnetaan nimellä Lena siinä esiintyvän Lena Söderbergin mukaan ja se on yksi käytetyimmistä testikuvista kuvanprosessoinnin alalla. Tämä kuva on kopioitu USC-SIPI [2014] kuvatietokannasta ja sitten muutettu harmaasävykuvaksi. Kuva reunoista on tuotettu prosessoimalla vasenta kuvaa GNU Octavella [2013].

Tämän tutkielman luvussa 2 tutustutaan reunantunnistuksen peruskäsitteisiin ja termeihin. Luvussa 3 käydään läpi tunnetuimmat reunantunnistusalgoritmit ja esitellään lyhyesti muutama muu. Luvussa 4 vertaillaan eri reunantunnistusalgoritmeja. Lopuksi luvussa 5 on yhteenveto tutkielman käsittelemistä asioista.

2. Peruskäsitteitä

2.1. Digitaalisista kuvista

Digitaalinen kuva on kaksi- tai kolmiulotteinen matriisi, jonka kullakin alkiolla eli *pikselillä* on jokin diskreetti arvo tai vektori, jonka alkiot ovat diskreettejä. Esimerkiksi värivalokuvassa voi kullakin pikselillä olla erikseen oma arvonsa punaiselle, vihreälle ja siniselle värille, joiden yhdistelmänä saadaan aikaan muut värit. Tässä työssä käsitellään lähinnä vain kaksiulotteisia harmaasävykuvia, joita voidaan kutsua myös musta-valkoisiksi kuviksi.

Digitaalinen kuva on usein näyte jostain reaalimaailman ilmiöstä. Esimerkiksi otettaessa valokuva muodostuu kameran linssin läpi suodattuvasta valosta kaksiulotteinen kuva, jonka jokaisen pikselin värikoodi vastaa valonsäteiden aallonpituuksia kyseisessä kohdassa näkymää. Kamerat pystyvät muodostamaan tarkkoja kuvia, joissa on miljoonia pikseleitä, mutta reaalimailmassa ei ole edes tällaisia jakoja. Tällaista kuvan jakamista pikseleiksi sanotaan *näytteistämiseksi* [Gonzales and Woods, 2008].

Vastaavasti pikseleiden värikoodit muodostetaan valonsäteiden aallonpituuksien mukaan, jotka ovat myös jatkuvia. Digitalisoitaessa myös niistä tehdään diskreettejä koodeja ja erilaisia koodeja voi olla miljoonia, esimerkiksi RGB-koodauksessa 16.8 miljoonaa erilaista väriä, mutta värejä on silti rajallinen määrä. Gonzales ja Woods [2008] käyttävät tästä operaatiosta termiä *kvantisointi*.

Kaikkia digitaalisia kuvia ei muodosteta näkyvästä valosta. Niitä voidaan muodostaa myös esimerkiksi röntgensäteistä, ääniaalloista ja mikroaalloista. Olipa mitattava kohde mikä hyvänsä, ilmiön digitalisoinnin periaate on sama. Ilmiö jaetaan pikseleihin ja kullekin pikselille määritellään diskreetti arvo, joka vastaa ilmiöstä mitattua arvoa.

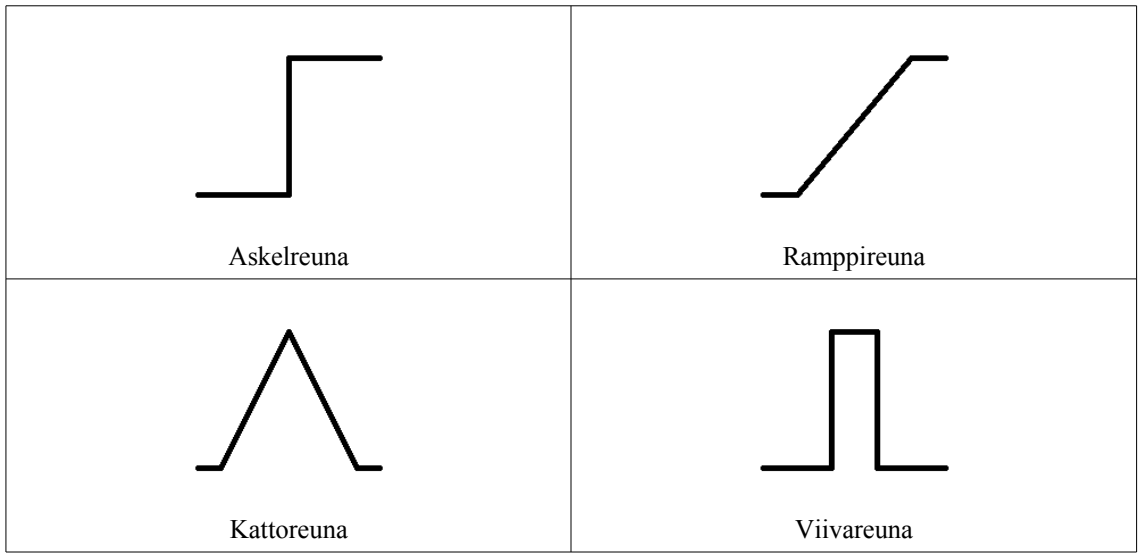
Mitatusta arvosta käytetään nimitystä *intensiteetti*. Intensiteetti on fysiikan suure, jolla mitataan energian määrää, joka vaikuttaa tiettyyn pinta-alaan tietyssä ajassa.

Digitaalisia kuvia käsitellään usein kaksiulotteisena funktiona *f(x,y)*, jossa funktion arvo tietyssä pisteessä on vastaavan pikselin intensiteetti tai eri kanavien intensiteetit. Jos kyseessä on normaali valokuva, niin x- ja y-koordinaatit ovat ei-negatiivisia ja funktion arvo pisteessä *(x,y)* on vastaavan pikselin värikoodi eli intensiteetit erikseen kullekin eri värille.

2.2. Reunoista

Reuna on kohta, jossa kuvassa *f(x,y)*, tarkemmin sanottuna kuvan intensiteetissä, tapahtuu merkittävä muutos. Mikä sitten on merkittävää riippuu sovelluksesta. Ideaalisessa kuvassa eri alueiden välillä on selvät terävät erot, jolloin myös muutokset on helppo havaita. Luonnollisissa kuvissa on kuitenkin pääsääntöisesti mukana kohinaa, jolloin myös muutosten havaitseminen vaikeutuu. Gonzalesin ja Woodsin [2008] mukaan ei-triviaalien kuvien segmentointi onkin yksi digitaalisen kuvanprosessoinnin vaikeimmista tehtävistä.

Gonzales ja Woods [2008] jakavat reunat kolmeen eri malliin: askel- (eng. step edge), ramppi- (eng. ramp edge) ja kattoreuna (eng. roof edge). Park ja Murphey [2008] lisäävät vielä neljännen reunatyypin, viivareunan (eng. line edge). Kunkin reunatyypin kuvaajat näkyvät kuvassa 2.2.1.



Kuva 2.2.1: Eri reunatyyppien kuvaajat.

Jos kahden vierekkäisen pikselin intensiteeteissä on huomattava ero, silloin kyseessä on askelreuna. Nimitys tulee siitä, että kun tällaisen muutoksen kuvaajaa katsoo, se näyttää portaiden askelmalta. Askelreunoja esiintyy harvoin luonnollisissa kuvissa, joissa on usein mukana kohinaa. Kuvien esiprosessoinnilla pyritään poistamaan kohinaa ja terävöittämään reunoja, jolloin tämä malli on käyttökelpoinen myös reaalimaailman kuvien kanssa.

Kun intensiteetin kasvu jakautuu usean pikselin leveydelle, kyseessä on ramppireuna. Nimitys tulee jälleen kuvaajasta, jossa intensiteetti nousee tai laskee tasolta toiselle kulmassa, jonka itseisarvo on pienempi kuin 90 astetta. Tällaiset reunat

ovat tyypillisiä luonnollisissa kuvissa, joissa valaistus tai muut olosuhteet sumentavat reunoja.

Kattoreunasta puhutaan, jos intensiteetti ensin nousee ja sitten heti laskee. Tällöin sen kuvaaja muistuttaa harjakattoa. Alkuperäisessä kuvassa tämä kohta on ollut hyvin ohut tai pistemäinen kohta, jonka keskikohdan intensiteetti on suurempi kuin sen sivujen.

Viivareuna on kuin askelreuna, mutta säilyttää uuden arvonsa vain muutaman pikselin verran ja palautuu sitten takaisin alkuperäiselle tasolle. Parkin ja Murpheyn [2008] mukaan askel- ja viivareunat ovat harvinaisia luonnollisissa kuvissa ja kohinan ansiosta muuttuvat käytännössä ramppi- ja kattoreunoiksi.

2.3. Reunantunnistuksesta

Reunantunnistus on operaatio, jossa digitaalisesta kuvasta etsitään erilaisten alueiden ääriviivat. Alueiden erilaisuus määräytyy niiden kirkkauden, värin tai tekstuurin mukaan. Jos kuvassa on tummalla pohjalla vaalea pallo, niin reunantunnistuksen tulos olisi tuon pallon ääriviivat eli ympyrä.

Tämä työ käsittelee reunantunnistusalgoritmeja, jotka saavat syötteekseen kaksiulotteisen kuvan ja tuloksena on kuvasta etsityt reunat. Näistä algoritmeista käytetään myös nimitystä reunantunnistin (eng. edge detector).

Digitaalinen kuva voidaan jakaa osiin alueiden samankaltaisuuden perusteella tai paikallisten muutosten perusteella. Reunantunnistusta käytetään paikallisten muutosten etsimiseen. [Gonzales and Woods, 2008]

Reunantunnistus on tärkeä esiprosessointivaihe useissa konenäkösovelluksissa kuten teollisuuden tarkastuksissa, liikkeentunnistuksessa, kuvan pakkauksessa ja sormenjälkien tunnistuksessa. Korkean tason konenäkösovellusten toimiminen perustuu hyvin ja luotettavasti tehtyyn reunantunnistukseen. [Park and Murphey, 2008]

Gonzalesin ja Woodsin [2008] mukaan reunantunnistus voidaan jakaa kolmeen vaiheeseen:

1. *Kohinan vähentäminen* tasoittamalla kuvaa. Tasoittamisella on kuitenkin ikävä sivuvaikutus, joka sumentaa mahdollisia kuvassa olevia teräviä reunoja.
2. *Reunapisteiden tunnistus*. Paikallinen operaatio, jolla etsitään kuvasta kaikki mahdolliset kandidaatit reunapisteiksi.
3. *Reunapisteiden yhdistäminen*. Valitaan edellisessä kohdassa löydetyistä pisteistä ne, jotka muodostavat todellisen reunan.

Mainittakoon, että yllämainitut vaiheet toistuvat useissa algoritmeissa, mutta jotkin algoritmit eivät tasoita kuvaa ollenkaan, vaan yrittävät erottaa kohinan oikeista reunapisteistä muilla keinoin. On myös algoritmeja, jotka eivät yhdistä reunapisteitä. Reunapisteiden tunnistus luonnollisesti tehdään kaikissa algoritmeissa.

2.4. Suodatus ja kernelit

Suodatus (eng. filtering) on keskeinen menetelmä kuvien prosessoinnissa. Suodattamalla saadaan tehtyä esimerkiksi kuvan tasoittaminen tai terävöittäminen. Suodatuksessa kuvan jokainen pikseli käydään läpi ja niiden arvoa muutetaan *kernelin* määräämällä tavalla.

Kernelit ovat pieniä matriiseja, jotka ovat usein kokoa $n \times n$, jossa n on pariton luku. Kernelin koko määrittelee, kuinka suuri alue kuvan pikselin ympäriltä otetaan huomioon. Kernelistä käytetään myös muita nimityksiä kuten maski (eng. mask), suodatin (eng. filter) tai ikkuna (eng. window) [Gonzales and Woods, 2008]. Kernelin alkiot ovat kertoimia, joita käytetään suodatuksessa.

Kerneleistä puhutaan usein myös *operaattoreina*. Matematiikassa operaattori on funktio, joka muuttaa toista funktiota. Kuvankäsittelyssä kuvia käsitellään kaksiulotteisina funktioina, joten on luontevaa käyttää termiä operaattori kuvafunktiota muokkaavista kerneleistä. Tämä ei tosin ole täysin matemaattisen määritelmän mukaista, koska kerneli määrittelee vain kertoimet ja vaikutusalueen. Vastaavasti voitaisiin sanoa, että yhtälössä $A+B=C$, A on operaattori, mikä ei tietenkään ole totta. Operaattori-termin käyttö on vakiintunut joissakin yhteyksissä, esimerkiksi Prewittin operaattoreiden tapauksessa (luku 3). Sekaannusten välttämiseksi tässä työssä käytetään termiä kerneli. Niissä yhteyksissä, joissa termi operaattori on yleisesti käytössä alan kirjallisuudessa, tämä mainitaan erikseen.

Kerneleitä käytetään suodatuksessa kahdella eri tavalla, joista *korrelaatio* lienee yleisempi. Korrelaatiossa kernelin keskimmäinen alkio ikään kuin asetetaan kuvan pikselin päälle. Sitten kernelin alkioilla kerrotaan alle jäävien pikselien arvot ja lasketaan kaikki yhteen. Tulojen summa on operaation tulos kyseiselle pikselille. Sama kaavana:

$$w(x,y) cor\, f(x,y) = \sum_{s=-a}^{a} \sum_{t=-a}^{a} w(s,t) f(x+s, y+t) \qquad (2.4.1)$$

Yllä olevassa kaavassa *cor* on korrelaatio-operaattori, jota käytetään kernelille w ja kuvalle f. Kernelin w rivit ja sarakkeet on numeroitu siten, että keskimmäisen alkion rivi

ja sarake on *(0,0)*. Näin ollen 3×3 -kokoisen kernelin sarakkeet ja rivit olisivat -1, 0 ja 1. *a* on kernelin maksimi rivin tai sarakkeen numero. Siis *n*×*n* -kernelin tapauksessa *n=2a+1*. Kun operaatio suoritetaan kuvan kaikille pikseleille saadaan uusi suodatettu kuva.

Toinen tapa on *konvoluutio*, joka poikkeaa korrelaatiosta vain siten, että ensin kerneliä kierretään 180 astetta ja sitten tehdään korrelaatio [Gonzales and Woods, 2008]. Konvoluutio voidaan tehdä myös siten, että kierretäänkin kuvan pikseleistä operaation aikana. Tällöin kaava saadaan seuraavaan muotoon, missä *conv* on konvoluutio-operaattori [Gonzales and Woods, 2008].

$$w(x,y)conv\, f(x,y) = \sum_{s=-a}^{a} \sum_{t=-a}^{a} w(s,t) f(x-s, y-t) \tag{2.4.2}$$

Jos kerneli on kokoa *n*×*n* ja symmetrinen keskipisteensä suhteen, niin silloin korrelaatio ja konvoluutio antavat saman tuloksen. Shapiron ja Stockmanin [2001] mukaan kuvan prosessoinnin alalla on tyypillistä puhua konvoluutiosta vaikka oikeasti tarkoitetaan korrelaatiota. Kernelitkin saatetaan esittää jo valmiiksi kierrettyinä, jolloin korrelaatio antaa oikean tuloksen. Myös Gonzales ja Woods [2008] valittavat alalla esiintyvää epätäsmällisyyttä näiden termien suhteen.

2.5. Kuvan gradientti

Funktion derivaatasta nähdään milloin funktion arvo muuttuu ja kuinka jyrkkä muutos on. Gradientit ovat kuin vektoriavaruuksien derivaattoja, joista nähdään mihin suuntaan pinta kaareutuu ja kuinka jyrkästi. Kuvan *f(x,y)* gradientti ∇f määritellään sen osittaisderivaattojen vektorina. Tämä vektori osoittaa suurimman muutoksen suuntaan pisteessä *(x,y)*.

$$\nabla f \equiv grad(f) \equiv \begin{bmatrix} g_x \\ g_y \end{bmatrix} = \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{bmatrix} \tag{2.5.1}$$

Vektorin ∇f pituus kertoo muutoksen jyrkkyyden. Mitä pidempi vektori on sitä suurempi muutos. Englanninkielisessä kirjallisuudessa pituudesta käytetään termiä *magnitude* eli itseisarvo. Olkoon siis *M(x,y)* gradientin pituus pisteessä *(x,y)*. Gradientin pituus lasketaan seuraavalla kaavalla.

$$M(x,y)=mag(\nabla f)=\sqrt{g_x^2+g_y^2} \tag{2.5.2}$$

Gradientin pituuden laskenta kaavan 2.5.2 mukaan on raskasta siinä olevien neliöön korotuksien ja neliöjuuren vuoksi. Pitää ottaa huomioon, että näitä kaavoja sovelletaan aineistoihin, joissa voi olla miljoonia pisteitä. Gonzalesin ja Woodsin [2008] mukaan gradientin pituutta usein approksimoidaan gradienttikomponenttien g_x ja g_y itseisarvojen summalla.

$$M(x,y)\approx|g_x|+|g_y| \tag{2.5.3}$$

Gradientin suunta saadaan laskettua seuraavasti

$$\alpha(x,y)=\arctan\left(\frac{g_y}{g_x}\right) \quad . \tag{2.5.4}$$

Gonzalesin ja Woodsin [2008] mukaan gradienttivektorin komponentit g_x ja g_y eli osittaisderivaatat kuvalle *f(x,y)* saadaan alla olevilla kaavoilla. Tämä on digitaalinen likiarvo osittaisderivaatoista.

$$g_x=\frac{\partial f(x,y)}{\partial x}=f(x+1,y)-f(x,y) \tag{2.5.5}$$

$$g_y=\frac{\partial f(x,y)}{\partial y}=f(x,y+1)-f(x,y) \tag{2.5.6}$$

2.6. Muita termejä

Tässä luvussa käydään läpi muita termejä.

Kynnystäminen (eng. thresholding) on operaatio, jossa funktion arvot asetetaan sen perusteella ylittävätkö ne jotain annettua kriteeriä eli kynnystä. Kynnyksiä voi olla myös useampia kuin yksi. Alla on esimerkki, jossa kynnystetään funktio binääriseksi.

$$g(x,y)=\begin{cases}1 & jos\ f(x,y)>kynnys\\ 0 & jos\ f(x,y)\leqslant kynnys\end{cases} \tag{2.6.1}$$

Pikselin *naapureilla* (eng. neighbors) tarkoitetaan pikselin ympärillä olevia muita kahdeksaa pikseliä. Käytetään merkintää $N_8(p)$, kun tarkoitetaan kaikkia pikselin *p*

kahdeksaa naapuria. Jos tarkoitetaan vain horisontaalisia ja vertikaalisia naapureita, käytetään merkintää $N_4(p)$. Diagonaalisista naapureista käytetään merkintää $N_D(p)$. Naapureille pätevät seuraavat lauseet.

$$N_4(p) \cup N_D(p) = N_8(p) \tag{2.6.2}$$

$$N_4(p) \cap N_D(p) = \emptyset \tag{2.6.3}$$

Pikseleiden *vierekkäisyydestä* (eng. adjacency) puhutaan silloin, kun kahden naapuripikselin intensiteetit kuuluvat samaan joukkoon V. V voi olla mikä tahansa intensiteettien osajoukko. Esimerkiksi kaikki intensiteetit, jotka ylittävät jonkin tietyn kynnyksen. Vastaavasti kuin naapureista, käytetään myös käsitteitä *4-vierekkäisyys* ja *8-vierekkäisyys*, kun määritellään mistä naapureiden joukosta vierekkäisiä pikseleitä etsitään. Näiden lisäksi on olemassa vielä *sekavierekkäisyys*, joka määritellään seuraavasti.

Pikselit p ja q, joiden intensiteetit kuuluvat joukkoon V, ovat sekavierekkäisiä, jos

1. q kuuluu joukkoon $N_4(p)$ tai
2. q kuuluu joukkoon $N_D(p)$ ja joukossa $N_4(p) \cap N_D(q)$ ei ole yhtään pikseliä, joiden arvo kuuluu joukkoon V.

Pikseleiden p ja q välillä on *polku* (eng. path), jos *vierekkäisiä* pikseleitä seuraamalla päästään kulkemaan pikselistä p pikseliin q. Jälleen voidaan puhua 4-, 8- tai sekapolusta riippuen käytettävästä vierekkäisyydestä.

Olkoon S kuvan pikseleiden osajoukko. Jos pikseleiden p ja q välillä on polku joukossa S, niin silloin sanotaan, että ne on *yhdistetty* (eng. connected). Riippuen käytettävästä vierekkäisyydestä, puhutaan 4-, 8- tai sekayhdistetyistä pikseleistä.

3. Reunantunnistusalgoritmeja

Tässä luvussa käydään yksityiskohtaisesti läpi tunnetuimmat reunantunnistusalgoritmit. Nämä käydään läpi useimmissa kuvanprosessoinnin oppikirjoissa ja näihin on lukuisia viittauksia tieteellisissä artikkeleissa. Monissa kuvanprosessoinnissa käytettävissä ohjelmistoissa nämä ovat valmiita työkaluja. Esimerkiksi MATLABissa [2014] on valmiina Roberts, Sobel, Prewitt, LoG ja Canny -algoritmit.

Lisäksi käydään lyhyesti läpi luvussa 4 tehtävissä vertailuissa esiintyviä algoritmeja. Luvun lopuksi esitellään vielä aallokemuunnos ja eräs aallokemuunnosta käyttävä algoritmi.

3.1. Varhaiset gradienttiin perustuvat algoritmit

Reunantunnistus on ongelmana yli 50 vuotta vanha [Park and Murphey, 2008]. Määritelmän mukaan reunantunnistuksessa pyritään etsimään merkittäviä paikallisia muutoksia. Näin ollen olikin luontevaa, että alan pioneerit L. G. Roberts, J. M. S. Prewitt ja I. E. Sobel lähtivät etsimään ratkaisua kuvan gradienttien kautta. Robertsin tutkimus on vuodelta 1965 ja Prewittin ja Sobelin tutkimukset vuodelta 1970. Kopiota niistä ei ollut saatavilla, joten tämä luku perustuu lähteisiin Gonzales ja Woods [2008] sekä Park ja Murphey [2008], jotka ovat yhteneviä oleellisin osin.

Nämä kaikki algoritmit perustuvat kuvan suodattamiseen korrelaatiolla. Korrelaatiossa käytettävät kernelit ovat ainoa asia, joka erottaa algoritmit toisistaan. Kussakin algoritmissa on kaksi kerneliä, joista toisella haetaan x-suuntainen gradientti ja toisella y-suuntainen. Jokaisen kernelin kertoimien summat ovat nollia, mistä seuraa se, että jos kuvan kaikilla pikseleillä on sama intensiteetti tarkkailtavassa kohdassa, tulokseksi tulee nolla.

Robertsin kernelit ovat 2×2 -matriiseja. Tämä poikkeaa myöhemmin omaksutusta tavasta käyttää kerneleitä, joissa on rivejä ja sarakkeita pariton määrä. Gonzalesin ja Woodsin [2008] mukaan parittomuus tekee kerneleistä symmetrisiä keskipisteen suhteen, joka antaa lisätietoa reunan suunnasta. Suodatuksessa Robertsin kernelin ensimmäisen rivin ja ensimmäisen sarakkeen alkio asetetaan kuvan *f(x,y)* alkion päälle.

Robertsin kernelit (Robertsin operaattorit).

$$g_x = \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix}, \quad g_y = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}$$

Prewittin kernelit (Prewittin operaattorit).

$$g_x = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}, \quad g_y = \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}$$

Sobelin kernelit (Sobelin operaattorit) ovat muuten samat kuin Prewittin kernelit, mutta keskimmäiset kertoimet ovat saaneet suuremman painoarvon.

$$g_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}, \quad g_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

Reunantunnistus käyttäen yllä olevia kerneleitä tapahtuu seuraavasti.

1. Suodatetaan korreloimalla kuva käyttäen kumpaakin kerneliä erikseen. Näin saadaan kuvan gradientit.
2. Lasketaan gradientin itseisarvot kussakin pisteessä kaavalla 2.5.3. Näin saadaan kuva *M(x,y)*.
3. Kynnystetään kuvaa *M(x,y)* pudottamalla pienimmät itseisarvot pois. Tästä muodostetaan yleensä binäärikuva, jossa kynnyksen alapuolelle jääneet pikselit asetetaan nolliksi ja muut ykkösiksi. Pikselit, joiden arvo on yksi, ovat reunapisteitä.

Kuva 3.1.1: Vasemmalla ylhäällä alkuperäinen kuva. Oikealla ylhäällä kuvasta etsityt reunat käyttäen Sobelin algoritmia kynnysarvolla 0.2. Tulos on alarivin kuvien summa. Vasemmalla alhaalla Sobelin g_x-kernelillä suodatettu kuva ja oikealla alhaalla g_y-kernelillä suodatettu kuva. Alkuperäinen kuva on saatu USC-SIPI [2014] kuvatietokannasta. Kuvat reunoista on prosessoitu alkuperäisestä kuvasta GNU Octavella [2013].

Gradientit ovat hyvin alttiita kohinalle ja onkin tavallista suorittaa jonkinlainen kohinan suodatus ennen reunantunnistusta.

Parkin ja Murpheyn [2008] mukaan yksi ongelma näiden algoritmien käytössä on sopivan kynnyksen valinta. Jos kynnys asetetaan liian korkealle, osa reunapisteistä voi jäädä havaitsematta. Jos se taas asetetaan liian matalalle, silloin vääriä pisteitä, kuten kohinaa, merkataan reunoiksi. Toinen ongelma on se, että nämä muodostavat paksuja reunoja, koska luonnollisissa kuvissa on harvoin askelreunoja. Tällöin havaittuja reunoja tulee ohentaa.

Kuvassa 3.1.1 on selvästi havaittavissa paksuja eli yli yhden pikselin levyisiä reunoja. Jos kynnysarvo on pieni, niin leveitä reunoja havaitaan ramppireunojen alueella, koska silloin pienetkin muutokset hyväksytään reunoiksi. Toisaalta jos kynnys

asetetaan liian korkeaksi, ramppireunoja ei havaita ollenkaan, koska sen alueella ei tapahdu riittävän suurta muutosta intensiteetissä.

Paksut reunat eivät kuitenkaan ole pelkästään ramppireunojen ongelma, vaan täysin ideaalisessa askelreunassakin Prewittin ja Sobelin algoritmit muodostavat vähintään kahden pikselin levyisen reunan. Esimerkiksi suodatetaan binäärikuvaa, jonka vasen puoli on musta eli sen intensiteetti on nolla ja oikea puoli valkoinen eli sen intensiteetti on binäärikuvan tapauksessa yksi. Nyt kun kuvaa suodatetaan esimerkiksi Prewittin g_x-kernelillä lähestyen mustan ja valkoisen alueen välistä reunaa vasemmalta, niin kun kernelin oikea reuna menee valkoisen alueen päälle tulee gradientin itseisarvoksi kolme.

$$-1*0+-1*0+-1*0+0*0+0*0+0*0+1*1+1*1+1*1=3$$

Kun kerneliä siirretään askel oikealle, niin jälleen itseisarvoksi tulee kolme.

$$-1*0+-1*0+-1*0+0*1+0*1+0*1+1*1+1*1+1*1=3$$

Tässä vaiheessa algoritmi on siis löytänyt kaksi reunapistettä ideaaliselle askelreunalle. Seuraavalla askeleella kerneli siirtyy kokonaan valkoisen alueen ylle, jolloin gradientti saa jälleen arvokseen nollan.

Edellisessä esimerkissä gradientit saavat arvoikseen suurempia lukuja kuin mikä oli alkuperäisen kuvan intensiteetin maksimi. Jos tämä on sovelluksen kannalta ongelmallista, voidaan kerneleiden kertoimia normalisoida siten, että tulos on aina nollan ja maksimin välissä. Esimerkiksi Prewittin kernelille voisi antaa kertoimeksi 1/3.

Tässä luvussa esiteltyjen Prewittin ja Sobelin kerneleiden lisäksi niistä on olemassa variaatiot, joissa kerneleitä on kierretty 45 astetta. Niillä voidaan paremmin havaita diagonaalisia reunoja. Joissakin ratkaisuissa [Robinson, 1977] tätä on viety vieläkin pidemmälle ja kerneleistä on tehty kahdeksan eri variaatiota, yksi kuhunkin pää- ja väli-ilmansuuntaan. Tästä käytetään nimitystä *kompassireunantunnistin* (eng. compass gradient edge detector). Kompassireunantunnistimen periaate on sama kuin muiden tässä luvussa esiteltyjen reunantunnistimien. Prewittin kerneleistä saman akselin vastakkaiset parit tuottavat vain erimerkkisen gradientin, joten kaikkia kahdeksaa kerneliä on turha käyttää ellei gradientien etumerkillä ole merkitystä. Sen sijaan Kirschin [1971] kerneleillä on erilaiset painoarvot kussakin suunnassa ja niistä käytetään kaikkia kahdeksaa. Hieman poiketen tässä luvussa esitellyistä muista algoritmeista, Kirschin algoritmissa ei käytetä kaavaa 2.5.3 laskemaan gradienttien komponenttien summaa, vaan valitaan gradientiksi suoraan se, jonka itseisarvo on suurin. Alla on muutama Kirschin kerneli esimerkkinä. Loput saadaan kiertämällä kerneliä 45 asteen askelin.

$$g_1=\begin{bmatrix} 5 & 5 & 5 \\ -3 & 0 & -3 \\ -3 & -3 & -3 \end{bmatrix},\ g_2=\begin{bmatrix} 5 & 5 & -3 \\ 5 & 0 & -3 \\ -3 & -3 & -3 \end{bmatrix},\ \ldots\ ,\ g_8=\begin{bmatrix} -3 & 5 & 5 \\ -3 & 0 & 5 \\ -3 & -3 & -3 \end{bmatrix}$$

3.2. Marr-Hildreth

Edellisessä luvussa käsitellyt gradientit perustuivat funktion ensimmäiseen derivaattaan, mutta ne muodostavat liian paksuja reunoja. Tästä seuraa kysymys, missä on oikea reuna. Tähän ongelmaan etsitään ratkaisua toisesta derivaatasta. Käymme seuraavaksi läpi Marr-Hildreth -algoritmin, joka käyttää toisen derivaatan lisäksi tasoittavaa Gaussin operaattoria.

Marrin ja Hildrethin [1980] mukaan eräs suurimmista vaikeuksista luonnollisten kuvien kanssa on, että niissä intensiteetin muutoksien vaihteluväli on suuri (vertaa askel- ja ramppireunat). Heidän mukaansa yksikään staattinen kerneli, kuten esimerkiksi Sobel, ei toimi optimaalisesti monella vaihteluvälillä. Näin ollen pitäisi pyrkiä mukautumaan paikallisiin muutoksiin. Tämä johti heidät ajatukseen tasoittaa ensin kuvaa paikallisella keskiarvolla ja sitten etsiä intensiteetin muutokset.

Marr ja Hildreth [1980] asettivat tasoittavalle operaattorille kaksi tavoitetta. Ensiksi sen piti pienentää intensiteettien vaihteluvälejä ja toiseksi olla spatiaalisesti paikallinen. Heidän mukaansa nämä tavoitteet olivat valitettavasti ristiriidassa keskenään, koska ne olivat eri avaruuksista. Gaussin jakauma oli optimaalisin kompromissi ongelmaan.

Gaussin kerneli on myös ainoa kerneli, joka kasvattaa paikallista maksimia ja pienentää paikallista minimiä, kun kernelin kokoa kasvatetaan. Tästä seuraa, että kernelin kokoa voidaan kasvattaa ilman, että hukataan reunoja kasvattamisen seurauksena. [Badaud et al., 1986]

Basun [2002] mukaan Gaussin kerneli on laajimmin käytetty tasoittava kerneli. Marrin ja Hidrethin esittelemä Gaussin kerneli oli käänteentekevä keksintö matalan tason kuvanprosessoinnin alalla [Basu, 2002].

Gaussin funktiosta muodostetaan Gaussin kerneli (3.2.1).

$$G(x,y)=\mathrm{e}^{-\frac{x^2+y^2}{2\sigma^2}} \tag{3.2.1}$$

Funktion ensimmäisestä derivaatasta nähdään, milloin funktion arvo muuttuu. Funktion toisesta derivaatasta puolestaan nähdään, milloin ensimmäisen derivaatan arvo muuttuu. Toisin sanoen funktion toinen derivaatta on nolla silloin, kun funktion arvo ei muutu tai se muuttuu tasaisesti. Funktion toinen derivaatta saa positiivisen arvon silloin, kun funktion arvo alkaa kasvamaan ja negatiivisen vastaavasti silloin, kun funktion arvo

alkaa laskemaan. Kuvafunktioiden yhteydessä toinen derivaatta saa positiivisen arvon, kun intensiteetti alkaa kasvamaan ja negatiivisen arvon, kun intensiteetti alkaa laskemaan. Intensiteetin muutos taas merkitsee reunaa. Jos kyseessä on askelreuna, niin toisen derivaatan kuvaajassa näkyy positiivinen tai negatiivinen piikki kyseisessä kohdassa. Jos taas kyseessä on ramppireuna, silloin rampin alussa ja lopussa on erimerkkiset piikit. Ramppireunat ovat ongelmallisia ensimmäisen derivaatan gradienteille, koska ne merkkaavat koko rampin reunaksi. Toinen derivaatta sen sijaan ilmoittaa rampin alun ja lopun. Vakiintunut käytäntö on etsiä kohta, jossa toisen derivaatan etumerkki vaihtuu (eng. zero crossing) ja merkitä reuna siihen pisteeseen. Kyseisen kohdan etsimiseen kuvafunktiosta on omat menetelmänsä ja palaamme niihin myöhemmin. Jos ajatellaan toisen derivaatan kuvaajaa, silloin nollan ylitys tapahtuu pisteessä, jossa suora, joka on vedetty negatiivisen piikin kärjestä positiivisen piikin kärkeen, leikkaa x-akselia.

Marr ja Hildreth [1980] päättivät käyttää toista derivaattaa etsiessään reunoja, mutta he halusivat välttää kerneleiden määrittelyä erikseen joka suuntaan. He päätyivät käyttämään *Laplacen operaattoria*, joka on Gonzalesin ja Woodsin [2008] mukaan yksinkertaisin *isotrooppinen* eli kierron suhteen invariantti toisen derivaatan operaattori. Laplacen operaattori voidaan määritellä kuvalle *f(x,y)* seuraavasti.

$$\nabla^2 f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} \tag{3.2.2}$$

Digitaalisten kuvien diskreetissä ympäristössä Laplacen operaattorin osittaisderivaatat voidaan laskea seuraavasti.

$$\frac{\partial^2 f}{\partial x^2} = f(x-1, y) - 2\mathrm{f}(x, y) + f(x+1, y) \tag{3.2.3}$$

$$\frac{\partial^2 f}{\partial y^2} = f(x, y-1) - 2\mathrm{f}(x, y) + f(x, y+1) \tag{3.2.4}$$

Osittaisderivaattojen yhdistelmää voidaan approksimoida seuraavalla kernelillä [Park and Murphey, 2008], mutta myös muita variaatioita on.

$$\nabla^2 f(x, y) = \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix} \tag{3.2.5}$$

Kun saadulla kernelillä suodatetaan kuva, reunapisteet löytyvät kohdista, joissa tapahtuu nollan ylitys eli suodatetun kuvan intensiteetti vaihtuu positiivisesta negatiiviseksi tai päinvastoin. Tämä on Laplacen reunantunnistusoperaattori ja sen isotrooppisuudesta johtuen se pystyy havaitsemaan reunat joka suunnasta, toisin kuin Prewittin ja Sobelin kernelit, jotka piti määritellä erikseen tarvittaville suunnille. Kuten ensimmäisen derivaatan gradienttioperaattorit, myös Laplacen reunantunnistusoperaattori on hyvin herkkä kohinalle ja sitä harvoin käytetään ilman kohinan suodatusta. [Park and Murphey, 2008]

Kun yhdistetään Gaussin ja Laplacen kernelit konvoloimalla ensimmäinen jälkimmäisellä, saadaa Marrin ja Hildrethin [1980] kehittämä Gaussin-Laplacen -kerneli (operaattori), josta käytetään lyhennettä LoG. LoG tulee menetelmän englanninkielisestä nimestä Laplacian of a Gaussian.

LoG-kerneli voidaan muodostaa kaavalla (3.2.6). Kaavassa σ on säätöparametri, jolla säädetään tasoitettavan alueen kokoa. Mitä suurempi σ on, niin sitä enemmän kuvaa tasoitetaan ja pienet yksityiskohdat katoavat. Jos $\sigma < 0.5$, niin silloin LoG -operaattori muuttuu käytännössä Laplacen operaattoriksi, koska Gaussin funktion vaikutus on niin pieni. Digitaalisten kuvien diskreetissä ympäristössä σ vastaa yhtä pikseliä. Gonzalesin ja Woodsin [2008] mukaan $n \times n$ LoG -kernelin koko kannattaa valita siten, että n on pienin pariton kokonaisluku, joka on suurempi kuin 6σ. Näin on siksi, että Gaussin jakaumassa 99.7% todennäköisyysmassasta on $\pm 3\sigma$ päässä keskipisteestä.

$$\nabla^2 G(x,y) = \left[\frac{x^2+y^2-2\sigma^2}{\sigma^4}\right] e^{-\frac{x^2+y^2}{2\sigma^2}} \qquad (3.2.6)$$

LoG-kerneli voidaan siis muodostaa valmiiksi ja sitten suodattaa sillä kuva. Vaihtoehtoinen tapa on suodattaa kuva ensin Gaussin funktiolla muodostetulla kernelillä ja sitten Laplacen kernelillä. Marr-Hildreth -algoritmi voidaan siis tiivistää seuraavasti:

1. Suodata kuva Gaussin funktiolla muodostetulla kernelillä.
2. Suodata saatu kuva Laplacen kernelillä.
3. Etsi kohdassa 2 saadusta kuvasta kohdat, joissa tapahtuu nollan ylitys.

Yksi tapa etsiä nollan ylitykset on tarkastella pikselin N_8-naapureita. Tarkastellaan pikselin vastakkaisilla puolilla olevia pikseleitä ja jos niiden etumerkit eroavat, pikselin

kohdalla on nollan ylitys. Tällöin tarvitsee tehdä neljä eri vertailua: vaaka- ja pystysuoraan ja diagonaalit. Yleinen käytäntö on laskea lisäksi naapureiden erotuksen itseisarvo ja kynnystää sitä. Toisin sanoen, jos erotuksen itseisarvo ei ylitä tiettyä raja-arvoa, niin pikselin kohdalla ei ole nollan ylitystä. N_8-naapureista saadaan siis neljä eri erotusta, joten luonnollisesti näistä valitaan vertailtavaksi se, jonka itseisarvo on suurin. [Gonzales and Woods, 2008]

Eräs menetelmä on suodattaa kuva eri kokoisilla LoG-kerneleillä ja etsiä kustakin nollan ylitykset. Sitten valitaan ne nollan ylitykset, jotka löytyvät kaikista tuloksista. Tämä menetelmä voi tuottaa käytännöllistä informaatiota, mutta se on laskennallisesti raskas ja sitä käytetäänkin yleensä vain, jotta löydettäisiin sovellukseen parhaiten sopiva kernelin koko. [Gonzales and Woods, 2008]

Basun [2002] esittämän kritiikin mukaan LoG-kerneli ei toimi nurkissa tai reunoissa, joiden muoto ei ole lineaarinen, esimerkiksi ympyrä. Sellaisissa reunoissa LoG-kerneli sijoittaa reunapisteet väärään paikkaan.

3.3. Canny

Seuraavaksi tutustumme Cannyn [1986] kehittämään algoritmiin, johon on lukuisia viittauksia alan tieteellisissä artikkeleissa. Usein, kun esitellään uusi algoritmi, sen suorituskykyä verrataan juuri Cannyn algoritmiin. Parkin ja Murpheyn [2008] mukaan Cannyn algoritmi on yksi eniten käytetyistä reunantunnistusalgoritmeista.

Työssään "A Computational Approach to Edge Detection" Canny [1986] asetti seuraavat kolme kriteeriä reunantunnistimelle, joista mainittakoon myös, että ne ovat alan kirjallisuudessa yleisesti hyväksytyt hyvän reunantunnistimen kriteerit [Lopez-Molina et al., 2013]:

1. Hyvä havaitsemiskyky. Reunantunnistimella tulee olla mahdollisimman suuri todennäköisyys havaita oikeat reunapisteet ja vastaavasti pieni todennäköisyys havaita vääriä pisteitä.
2. Hyvä paikallistamiskyky (eng. localization). Löydettyjen reunapisteiden tulee olla mahdollisimman lähellä oikean reunan keskustaa.
3. Vain yksi pikseli löydettyä reunaa kohden, toisin sanoen mahdollisimman ohut reuna.

Canny [1986] loi matemaattiset yhtälöt kahdelle ensimmäiselle kriteerille ja sitten päätyi etsimään ratkaisua, jossa molemmat saavat mahdollisimman suuren arvon yhtä aikaa. Vastaavasti hän loi yhtälön kolmannelle kriteerille. Cannyn [1986] mukaan tälle yhtälöryhmälle oli vaikeaa ellei jopa mahdotonta löytää ratkaisua, mutta ratkaisua

pystyttiin helposti arvioimaan numeerisesti. Näin löytyi optimaalinen funktio, mutta sen laskeminen oli raskasta. Canny [1986] havaitsi, että optimaalinen funktio oli lähellä Gaussin funktion ensimmäistä derivaattaa. Gaussin funktion ensimmäinen derivaatta antoi 20% huonompia tuloksia kuin optimaalinen funktio, mutta virhe oli niin pieni, että se oli vielä hyväksyttävissä. Canny [1986] vertasi tulostaan Marrin ja Hildrethin [1980] käyttämään Laplacen Gaussin funktioon ja väitti Gaussin funktion ensimmäisen derivaatan olevan parempi kaksiulotteisessa avaruudessa, koska se tunnisti ja paikallisti reunapisteitä paremmin. Myös Basun [2002] mukaan LoG-kerneli paikallistaa reunapisteet väärin, kun reunat eivät ole lineaarisia.

Yllä olevat tarkastelut Canny teki yksiulotteisessa avaruudessa. Tulos voidaan laajentaa koskemaan kaksiulotteista avaruutta, kun otetaan huomioon, että yksiulotteiset tulokset pätevät reunan normaalin suuntaan. Normaalin suuntaa ei tiedetä etukäteen, jolloin Gaussin funktion ensimmäistä derivaattaa pitäisi käyttää joka suuntaan. Riittävän lähelle oikeaa tulosta päästään, kun ensin suodatetaan kuva korreloimalla se Gaussin funktiolla ja lasketaan gradientit kaikissa pisteissä. Gradienttien laskentaan voidaan käyttää esimerkiksi Prewittin tai Sobelin kerneleitä. Tämän jälkeen käyttäen kaavoja 2.5.2 ja 2.5.4 saadaan gradientin suunta ja itseisarvo, jotka kuvastavat reunan suuntaa ja voimakkuutta kussakin pisteessä. [Gonzales and Woods, 2008]

Gradienttikuvassa *M(x,y)*, joka saatiin edellisessä kohdassa, on tyypillisesti leveitä harjantaitteita paikallisen maksimin ympärillä. Tämä on vastoin Cannyn asettamaa kolmatta kriteeriä, joten havaittua reunaa tulee ohentaa. Ohennus suoritetaan käyttämällä menetelmää nimeltä *nonmaxima suppression* ja siihen liittyy olennaisesti gradientin eli reunan normaalin suunta. Kun tarkastellaan pikseliä p, niin otetaan tarkasteluun mukaan se naapuripikseli, joka on gradientin suunnassa ja se joka on päinvastaisessa suunnassa. Esimerkiksi jos alla olevassa kuvassa pikselin p gradientin suunta olisi 45°, niin otettaisiin siinä suunnassa oleva naapuri p_3 ja vastakkaisessa suunnassa oleva naapuri p_6 mukaan tarkasteluun. [Gonzales and Woods, 2008]

$$\begin{bmatrix} p_1 & p_2 & p_3 \\ p_4 & p & p_5 \\ p_6 & p_7 & p_8 \end{bmatrix}$$

Kun gradientin suuntaiset pikselit on valittu, niin vertaillaan pikselin p gradientin itseisarvoa kahden muun pikselin gradienttien itseisarvoihin. Jos pikselin p gradientin itseisarvo on pienempi kuin kummankan muun, niin tuloskuvassa g_N asetetaan vastaava pikseli nollaksi, muuten sen arvoksi tulee p:n gradientin itseisarvo. [Gonzales and Woods, 2008]

Nonmaxima suppression voidaan tiivistää seuraavasti:

1. Etsi pikselin (x,y) gradientin suuntaiset naapurit (x_A,y_A) ja (x_B,y_B).
2. Jos $M(x,y) < M(x_A,y_A)$ tai $M(x,y) < M(x_B,y_B)$, niin $g_N(x, y) = 0$, muuten $g_N(x,y) = M(x,y)$.

Cannyn algoritmin seuraava vaihe on *hystereesi* (eng. hysteresis) eli kynnystäminen kahdella erisuurella kynnyksellä. Kynnystämisessä suurin vaikeus on valita sopiva kynnys. Liian pienellä kynnyksellä mukaan tulee kohinaa ja liian suurella puolestaan hukataan oikeita reunapikseleitä. Cannyn [1986] mukaan yhden kynnyksen käyttäminen aiheuttaa reunaviivan katkonaisuutta, kun reunan gradientti sopivasti vaihtelee kynnyksen ylä- ja alapuolella. Tätä ongelmaa hän pyrki korjaamaan hystereesillä, jonka etuna on myös yksittäisten kohinaisten pikseleiden parempi suodatus [Canny, 1986].

Hystereesi toimii käytännössä siten, että valitaan kaksi erisuurta kynnystä. Pikselit, joiden gradientti ylittää korkeamman kynnyksen T_H, merkitään suoraan reunapikseleiksi algoritmin viimeisessä vaiheessa. Pikselit, joiden gradientti ylittää matalamman kynnyksen T_L, mutta ei ylitä ylempää kynnystä, merkitään kandidaateiksi. Canny [1986] suositteli, että näiden kynnyksien suhde olisi 2:1 ja 3:1 välillä. Gonzales ja Woods [2008] suosittelevat luomaan kaksi uutta kuvaa g_{NH} ja g_{NL}, jotka kummatkin alustetaan siten, että jokaisen pikselin intensiteetti on nolla. Tämän jälkeen kynnystetään nonmaxima suppression -menetelmällä saatu kuva $g_N(x, y)$ seuraavasti.

$$g_{NH}(x,y) = g_N(x,y) \geqslant T_H$$
$$g_{NL}(x,y) = g_N(x,y) \geqslant T_L$$

Nyt g_{NL} sisältää myös kaikki g_{NH}:n pikselit, joiden intensiteetti on suurempi kuin nolla. Tämä ei ole suotavaa seuraavaa vaihetta ajatellen, joten asetetaan kuvassa g_{NL} kuvien yhteiset pikselit nolliksi. Tämä tehdään yksinkertaisesti vähentämällä kuvasta g_{NL} kuva g_{NH}.

$$g_{NL}(x,y) = g_{NL}(x,y) - g_{NH}(x,y)$$

Algoritmin seuraavassa ja viimeisessä vaiheessa valitaan kuvista g_{NL} ja g_{NH} reunapikselit. Kuvan g_{NH} kaikki pikselit valitaan suoraan reunapikseleiksi, koska ne

kynnystettiin korkeammalla kynnyksellä ja ovat siten vahvoja pikseleitä. Kuvasta g_{NL} valitaan pikselit Gonzalesin ja Woodsin [2008] mukaan seuraavalla menetelmällä:

1. Valitse kuvasta $g_{NH}(x, y)$ seuraava reunapikseli p, jossa ei ole vielä käyty.
2. Merkitse reunapikseleiksi kuvan $g_{NL}(x, y)$ kaikki pikselit, jotka voidaan *yhdistää* pikseliin p. Tässä voidaan käyttää esimerkiksi 8-yhdistämistä.
3. Jos kuvassa $g_{NH}(x, y)$ on vielä reunapikseleitä, joissa ei ole käyty, palaa kohtaan 1.
4. Merkitse nolliksi kaikki kuvan $g_{NL}(x, y)$ pikselit, joita ei ole merkattu reunapikseleiksi.

Lopuksi yhdistetään kuvat $g_{NH}(x, y)$ ja $g_{NL}(x, y)$ ja näin saadaan Cannyn algoritmin lopputulos. Gonzales ja Woods [2008] huomauttavat, että hystereesi voidaan toteuttaa nonmaxima suppressionin aikana ja viimeisessä vaiheessa tehty reunapikseleiden valinta voidaan tehdä suoraan kuvalle $g_N(x, y)$.

Yhteenvetona Cannyn algoritmin vaiheet ovat seuraavat:

1. Tasoita kuva käyttäen Gaussin kerneliä.
2. Laske gradientit ja niiden suunta.
3. Ohenna löydettyjä reunoja käyttäen nonmaxima suppression -menetelmää.
4. Käytä kaksinkertaista kynnystämistä ja yhdistä matalammalla kynnyksellä löydetyt reunapikselit korkeammalla kynnyksellä löydettyihin.

Basun [2002] mukaan Cannyn algoritmin huono puoli on se, että se tunnistaa liian herkästi heikkoja reunoja, joissa intensiteetin muutokset eivät ole suuria. Tästä syystä algoritmi toimii erityisen huonosti reunoissa, joissa reuna on sumentunut ja muodostaa loivan rampin [Basu, 2002]. Todettakoon kuitenkin, että sellaiset reunat ovat haasteellisia mille tahansa algoritmille, mutta Basun [2002] mukaan Cannyn algoritmi havaitsee olemattomia reunoja.

Deriche [1987] esitteli parannuksen Cannyn algoritmiin. Derichen mukaan Cannyn käyttämä Gaussin funktion ensimmäinen derivaatta on 25% huonompi kuin $f(x)=kxe^{-\alpha|x|}$, missä k ja α ovat sopivasti valittavia vakioita. Deriche myös esitteli funktiolleen tehokkaan toteutustavan.

Ding ja Goshtasby [2001] löysivät heikkouden Cannyn käyttämästä nonmaxima suppression -menetelmästä. Heidän mukaansa pitäisi tarkastella myös muita suuntia kuin gradientin suuntaa. Näin siksi, että esimerkiksi risteyskohdissa, joissa usean alueen intensiteetit sekoittuvat toisiinsa, reuna olisi silti löydettävissä, mutta gradientin suunta voi olla väärä. Heidän menetelmässään etsitään ensin pikselit, joiden gradientin

suuntaisilla naapureilla on pienemmät gradientin itseisarvot ja merkitään ne pikselit vahvoiksi. Jos pikseli ei ole vahva, niin tutkitaan onko missään muussa kolmesta suunnasta ehdot täyttävät naapurit. Jos tällaiset naapurit löytyivät, niin pikseli merkitään heikoksi. Muut pikselit merkitään taustaksi eli niiden intensiteetiksi tulee nolla. Dingin ja Goshtasbyn [2001] mukaan löydetyistä heikoista pikseleistä osa on vääriä, kohinan aiheuttamia pikseleitä. Niiden poistamiseksi he lisäsivät menetelmään tarkastelun, joka hyväksyy vain sellaiset heikot pikselit, joiden muodostama polku yhdistyy vahvoihin pikseleihin. Muut heikot pikselit hylätään. Tämän jälkeen jako vahvoihin ja heikkoihin pikseleihin poistetaan ja edetään Cannyn algoritmin seuraavaan vaiheeseen eli hystereesiin. Dingin ja Goshtasbyn menetelmällä on hyvät perusteet, mutta se kasvattaa tarvittavaa laskentaa.

3.4. Muita reunantunnistusalgoritmeja

Luvussa 4 vertaillaan eri algoritmeja. Tässä luvussa esittelen lyhyesti ne algoritmit, jotka esiintyvät luvussa 4, mutta joita ei ole vielä esitelty. Järjestys on kronologinen vanhimmasta uusimpaan.

Nalwan ja Binfordin algoritmissa [1986] sovitetaan *reunakkeita* (eng. edgel) kuvaan ja jos virhe on riittävän pieni, niin piste hyväksytään. Nalwa ja Binford käyttävät reunakkeen mallina hyperbolisen tangentin määräämän tason suoraa. Hyperbolisen tangentin kuvaaja muistuttaa askelreunaa ja oli siksi sopiva. Nalwa ja Binford [1986] käyttävät pienimmän neliösumman menetelmää laskeakseen kuinka hyvin kuva vastaa reunaketta. Kuvasta lasketun gradientin perusteella lasketaan alustava suunta reunakkeelle ja sitten sitä kierretään, kunnes riittävän hyvä vastaavuus löytyy tai todetaan, että virhe on liian suuri. Nalwan ja Binfordin algoritmi [1986] pyöristää reunakkeen suunnan lähimpään viidellä jaolliseen kulmaan asteina. Algoritmi käyttää 5×5 -kokoisia kerneleitä.

Bergholmin [1987] algoritmi suorittaa rekursiivisen reunaan tarkentamisen. Aluksi algoritmi suodattaa kuvan Gaussin kernelillä käyttäen suurehkoa σ:n arvoa. Näin saadaan aluksi hyvin karkea suodatus, josta etsitään maksimigradientit, jotka merkitään reunapisteiksi. Löydetyt reunapisteet välitetään seuraavalle kierrokselle, mutta samalla σ:n arvoa pienennetään. Nyt tällä toisella ja sitä seuraavilla kierroksilla etsitään uusia reunapisteitä vain edellisillä kierroksilla löytyneistä pisteistä ja niiden naapureista. Näin jatketaan kunnes σ:n arvo laskee liian pieneksi, jolloin tulos on valmis. [Bergholm, 1987]

Rothwell ja muut [1995] kuvaavat algoritminsa toimintaa jakamalla sen seuraaviin vaiheisiin:

1. Tasoita kuva käyttäen Gaussin kerneliä.

2. Laske x- ja y-suuntaiset gradientit.
3. Laske gradienttien normaali ja itseisarvo kussakin pisteessä.
4. Laske gradienttikuvan paikalliset maksimit pikselin osien (eng. sub-pixel) tarkkuudella.
5. Laske pikseleille kynnysarvot paikallisesti. Kynnykset luokittelevat pisteet reunapisteiksi tai muiksi pisteiksi.
6. Suorita topologisesti oikea ohennus, joka muuttaa ylimääräiset reunapisteet takaisin tavallisiksi kuvan pisteiksi.
7. Suorita topologisten primitiivien erottelu ja luo topologinen verkko, josta saadaan reunat.

Iversonin ja Zuckerin [1995] algoritmi perustuu Iversonin ja Zuckerin kehittämiin loogisiin/lineaarisiin L/L-operaattoreihin, jotka yhdistävät lineaari-operaattoreiden ja Boolen logiikan elementtejä. Iversonin ja Zuckerin [1995] mukaan reunantunnistimen tulee merkitä pikseli reunapisteeksi vain silloin, kun se on osa jotain kuvassa esiintyvää struktuuria. Heidän mukaansa väärät positiiviset havainnot ovat yleinen ongelma useimmissa lineaarisissa reunantunnistimissa, kuten esimerkiksi Cannyn algoritmissa. He hajottivat ongelman pieniksi loogisiksi operaattoreiksi, joiden yhdistelmä vastaa lineaarista operaattoria. Iversonin ja Zuckerin [1995] mukaan heidän algoritminsa havaitsee hyvin reunoja myös matalan kontrastin alueilta. He esittelevät tuloksia esimerkkikuvien avulla vertaamalla tuloksiaan Cannyn algoritmin tuloksiin. Kuvien perusteella Iversonin ja Zuckerin algoritmi on selvästi parempi, mutta kuten Heathin ja muiden [1997] tutkimuksesta nähdään, Iversonin ja Zuckerin [1995] käyttämät parametrit Cannyn algoritmille eivät ehkä olleet optimaaliset.

Meerin ja Georgescun [2001] kehittämässä algoritmissa on käytännössä kolme vaihetta: gradienttien laskenta, nonmaxima suppression ja hystereesi. Algoritmin erityispiirre on se, että siinä lasketaan todennäköisyydet, kuinka hyvin pikselin kohdalle asetettu reunasapluuna (eng. template) vastaa pikseliä ja sen ympäristöä. Algoritmissa lasketaan myös kullekin pikselille, millä todennäköisyydellä sen gradientti on suurempi kuin kuvan muut gradientit. Näitä kahta todennäköisyyttä käytetään gradienttien sijasta nonmaxima suppression- ja hystereesi-vaiheissa.

3.5. Aallokemuunnoksien käyttö reunantunnistuksessa

Aallokemuunnos on digitaalisen signaalinkäsittelyn menetelmä, joka on korvaamassa diskreetin Fourier-muunnoksen. Aallokemuunnoksessa signaalin muodostavat lokaalit osafunktiot. Aallokemuunnos tehdään suodattamalla syötesignaali käyttäen

kantafunktiosta johdettujen aalloke- ja skaalausfunktioiden eri tavoin skaalattuja ja aika-avaruudessa siirrettyjä variaatioita. [Wikipedia, 2014]

Kuvanprosessoinnissa aallokemuunnos muuntaa kuvafunktion *f(x,y)* toiseksi funktioksi *g(x,y)*, jonka maalijoukon alkioilla on kaksi arvoa: skaalaus (eng. scale) ja siirtymä (eng. shift). Yksinkertaistettuna skaalaus voidaan mieltää eräänlaisena zoomauskertoimena eli tavallaan kuinka läheltä kuvaa tarkastellaan. Siirtymää puolestaan käytetään paikan määrittämiseen *x,y* -avaruudessa eli rivin ja sarakkeen määrittämisessä. Suurella skaalauksella havaitaan pienetkin muutokset kuvan intensiteetissä ja pienellä skaalauksella nähdään suurempia kokonaisuuksia. Näin kuvan eri osia voidaan tarkastella eri skaalauksilla aallokemuunnoksen avulla. Aallokemuunnoksen avulla voidaan keskittyä paikallisiin muutoksiin kuten reunoihin. [Oskoei and Hu, 2010]

Käydään seuraavaksi lyhyesti läpi Hericin ja Zazulan [2007] algoritmi. Heric ja Zazula [2007] muodostivat reunamallin (eng. edge model) yhdelle riville tai sarakkeelle, jonka intensiteetit muodostavat yksiulotteisen signaalin. Tälle signaalille he tekivät aallokemuunnoksen käyttäen Haarin muunnosta. Tulosfunktion positiivisia ja negatiivisia huippuja kutsutaan nimellä *modulus maxima*. Niiden arvot kuvaavat reunojen jyrkkyyttä ja leveyttä. Suuri muutos lyhyellä matkalla esiintyy suurena maksimiarvona. Reunat esiintyvät huippuina (modulus maxima) usealla eri skaalauksella ja tämän ominaisuuden käyttäminen on osa Hericin ja Zazulan [2007] algoritmia.

Heric ja Zazula [2007] esittävät algoritminsa koostuvan seuraavista kuudesta vaiheesta.

1. Haarin muunnos riveistä tai sarakkeista muodostetuille signaaleille. Skaalauksen arvoina käytetään arvoja väliltä [1, $\lfloor \log_2(N) \rfloor$], missä *N* on signaalin eli rivin tai sarakkeen pituus. Tulos on aika-skaalaustaso *P*.
2. Modulus maxima -etsintä tasolta *P*. Tulos on taso *MM*.
3. Yhdistetään modulus maxima -pisteet tasolta *MM* maxima-viivoiksi tasolle *P*. Tulos on modulus maxima -viivojen joukko *S*.
4. Reuna-maximan valinta. Lasketaan kynnys *T* jokaisella skaalauksella tasolta *P*. Kynnyksen ylittävät modulus maximat tasolta *P* merkitään reuna-maximaksi.
5. Reunaviivojen valinta. Valitaan reunaviivoiksi ne viivat joukosta *S*, joilla on reuna-maximoita.
6. Reunan sijainnin asettaminen. Pienimmällä skaalauksella havaitun modulus maxima sijainti määrittelee reunan sijainnin.

Reunapisteiden löytymisen jälkeen Hericin ja Zazulan [2007] algoritmi vielä yhdistää löytyneet reunapisteet käyttämällä algoritmin omaa reunanlinkitysrutiinia. En kuitenkaan tarkastele sitä tämän tarkemmin.

Aallokemuunnoksiin liittyen Zhai ja muut [2008] kävivät läpi joukon uusia algoritmeja, jotka he jakoivat neljään eri ryhmään: aallokemuunnokseen pohjautuvat, matemaattiseen morfologiaan pohjautuvat, sumeisiin joukkoihin pohjautuvat ja fraktaaligeometriaan pohjautuvat. Näillä kaikilla on omat vahvuutensa ja heikkoutensa, mutta parhaiten tästä joukosta suoriutuivat aallokemuunnokseen pohjautuvat algoritmit [Zhai et al., 2008].

Aallokemuunnosten käyttöä reunantunnistukseen on tutkittu paljon viime vuosina [Gonzales and Woods, 2008]. Aiheena aallokemuunnokseen pohjautuva reunantunnistus on niin laaja, että tämän tutkielman puitteissa en siihen tämän enempää paneudu.

4. Algoritmien vertailua

Verrattaessa algoritmeja yleisesti kiinnostaa niiden nopeus ja tuloksen oikeellisuus. Reunantunnistusalgoritmien tapauksessa nopeutta voidaan vertailla esimerkiksi mittaamalla suoritusaikaa kellolla, mutta tuloksen oikeellisuuden mittaaminen on vaikeaa. Usein uuden algoritmin keksinyt henkilö vertailee tuloksiaan silmämääräisesti muiden algoritmien tuloksiin. Jos testikuvat ovat kovin kohinaisia, niin silloin jopa ihmisillä on vaikeuksia sanoa, missä reunan pitäisi olla ja eri ihmiset saattavat sijoittaa reunat eri kohtiin. Miten silloin voidaan vertailla eri algoritmien tuloksia, jos oikeaa vastausta ei ole? Vakiintunutta menetelmää vertailla eri reunantunnistusalgoritmien tuloksia ei ole, ellei sellaiseksi sitten lasketa silmämääräistä vertailua. Joitakin menetelmiä ongelman ratkaisuksi on esitetty ja tässä luvussa käyn niitä läpi. Aluksi kuitenkin vertailen hieman algoritmien suoritusaikoja.

4.1. Suoritusajoista

Park ja Murphey [2008] mittasivat eri algoritmien suoritusaikoja erikokoisilla kuvilla. Kuten oletettavaa on, yksinkertaiset Robertsin, Prewittin ja Sobelin algoritmit kuluttivat vähiten aikaa. Cannyn algoritmi kulutti noin kymmenkertaisesti aikaa Robertsiin, Prewittiin ja Sobeliin verrattuna, mutta LoG-algoritmi oli vain nimellisesti Cannyn algoritmia nopeampi. Kun katsotaan kuinka algoritmit on esitelty luvussa 3, voisi ajatella Cannyn algoritmin olevan selvästi hitaampi kuin LoG. Pitää kuitenkin ottaa huomioon, että Cannyn algoritmin kuvaus on lainattu Gonzalesin ja Woodsin oppikirjasta, jossa pääpaino on esityksen selkeydessä. Gonzalesin ja Woodsin [2008] mukaan algoritmi voidaan toteuttaa nopeammallakin tavalla.

Valitettavasti Park ja Murphey [2008] eivät mainitse mitään siitä, kuinka he toteuttivat algoritmit tai mitä parametreja he käyttivät. Kuvistakin on tiedossa vain niiden koot. Taulukkoon 4.1.1 on merkitty Parkin ja Murpheyn tulokset sekä laskettu suoritusajat per pikseli. Kun suoritusajat lasketaan pikselikohtaisesti, voidaan todeta erityisesti LoG ja Cannyn algoritmien kohdalla, että suoritusaika kasvaa lineaarisesti kuvan koon suhteen. Toisin sanoen niiden asymptoottinen suoritusaika olisi luokkaa $O(n)$. Hieman erikoisesti Robertsin, Prewittin ja Sobelin algoritmeilla menee suhteessa enemmän aikaa keskikokoisten kuvien laskemiseen kuin muiden, mutta toisaalta kyseessä ovat myös erittäin pienet ajan yksiköt. Jos tuloksista otettaisiin yksi desimaali pois, niin erot katoaisivat ja myös näiden algoritmien asymptoottisiksi suoritusajoiksi tulisi $O(n)$.

Kuvan koko	680 x 510	1152 x 864	1760 x 1168
Pikseleiden lukumäärä	346 800	995 328	2 055 680
Roberts-suoritusaika	32	125	219
per pikseli	0.00009	0.00013	0.00011
Prewitt-suoritusaika	62	234	422
per pikseli	0.00018	0.00024	0.00021
Sobel-suoritusaika	47	187	343
per pikseli	0.00014	0.00019	0.00017
LoG-suoritusaika	500	1453	2984
per pikseli	0.00144	0.00146	0.00145
Canny-suoritusaika	469	1531	3172
per pikseli	0.00135	0.00154	0.00154

Taulukko 4.1.1: Algoritmien suoritusajat millisekunteina Parkin ja Murpheyn [2008] mukaan. Valitettavasti lähteessä ei mainita millaisella koneella testit on tehty.

4.2. Rajan tunnistamiseen perustuva arviointimenetelmä

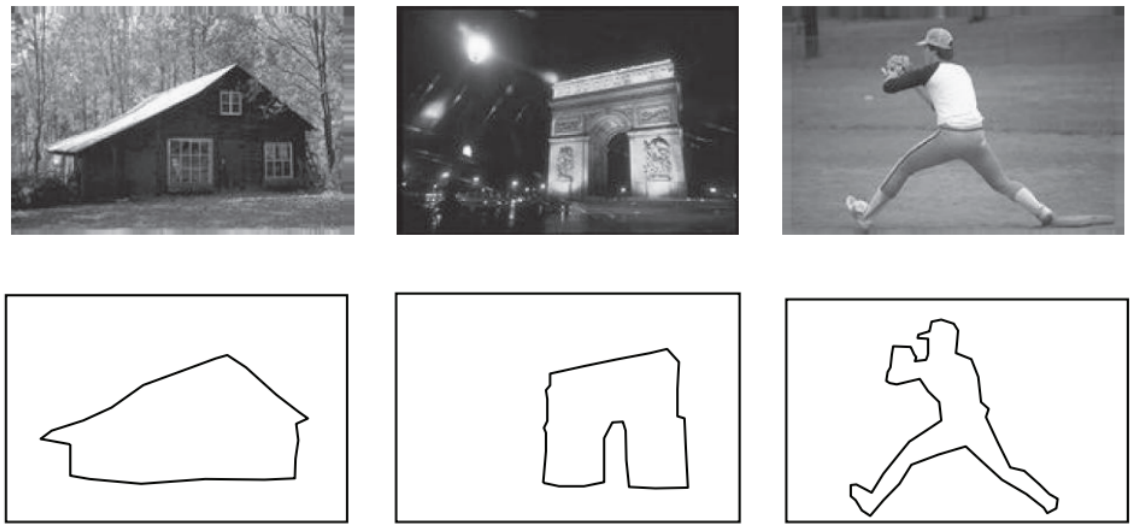
Wang ja muut [2006] luokittelevat reunantunnistusalgoritmien arviointimenetelmät subjektiivisiin ja objektiivisiin menetelmiin. Subjektiiviset menetelmät perustuvat ihmisen havaintoihin ja päätelmiin ja saattavat johtaa eri lopputuloksiin riippuen vertailijasta. Objektiivisissa menetelmissä mitataan kuvista ja reunantunnistuksen tuloksista testiarvoja, joiden perusteella algoritmit voidaan laittaa paremmuusjärjestykseen.

Wangin ja muiden [2006] mukaan on haastavaa löytää mittari, jolla mitata reunantunnistuksen tuloksia objektiivisesti. Yleistä mittaria on vaikea määritellä ja johonkin tiettyyn sovellukseen perustuvaa mittaria ei voi yleistää. Tämä on reunantunnistusalgoritmien vertailun dilemma [Wang et al., 2006]. Ratkaisuksi he esittävät neljää ominaisuutta, jotka hyvän vertailumenetelmän tulisi täyttää:

1. Yleistettävyys. Mittarin tulee olla yleisesti saatavilla ja soveltuva.
2. Objektiivinen vertailu. Vertailussa tulee käyttää kuvasta *yksiselitteisesti mitattavia ominaisuuksia* (eng. ground truth).
3. Aidot kuvat. Testiaineiston tulee koostua aidoista kohinaisista kuvista, koska aikaisemmat tutkimukset ovat osoittaneet [Wang et al., 2006], että synteettisillä kuvilla saadut tulokset eivät päde aidoilla kuvilla.
4. Suuri testiaineisto. Tarvitaan suuri testiaineisto, jotta tuloksille saadaan tilastollista merkittävyyttä.

Wang ja muut [2006] esittelivät uuden vertailumenetelmän, joka pohjautuu rajantunnistukseen (eng. boundary detection). Heidän menetelmässään reunantunnistuksen tulokset yhdistetään yhtenäiseksi rajaksi käyttämällä Wangin ja muiden [2005] kehittämää algoritmia, mutta mikä tahansa rajantunnistusalgoritmi käy. He päättelivät, että rajan muodostus on eräs reunantunnistuksen päätavoitteista monissa sovelluksissa, joten on sinänsä perusteltua käyttää rajoja vertailuun. Rajojen tunnistus tekee menetelmästä yleistettävän ja näin ensimmäinen ehto tulee täytettyä.

Objektiivista vertailua varten Wang ja muut [2006] valitsivat testiaineistoonsa kuvia, joissa oli yksi selvästi erottuva kohde kohinaisella taustalla. Kohteen rajat he etsivät itse jokaisesta kuvasta erikseen. Nämä rajat muodostavat ne yksiselitteiset mitattavat ominaisuudet, jotka täyttävät toisen ehdon. Reunantunnistusalgoritmien tuloksista muodostettuja rajoja verrataan näihin ennalta etsittyihin rajoihin. Wang ja muut [2006] valitsivat testiaineistoon yli tuhat aitoa kuvaa, jolloin loputkin heidän asettamistaan ehdoista tuli täytettyä. Kuvassa 4.2.1 on esimerkki kolmesta alkuperäisestä kuvasta ja niistä muodostetuista rajoista.



Kuva 4.2.1: Esimerkkikuva testiaineistosta. Lähde: Wang ja muut [2006].

Wang ja muut [2006] esittelivät menetelmän, miten reunapisteet yhdistetään rajaksi perustuen pitkälti Wangin ja muiden [2005] aikaisempaan työhön. Sen tarkastelu on tämän työn aiheen ulkopuolella, mutta sen sijaan tulosten vertailu ei ole. Jos merkitään ennalta etsittyjen rajojen sisäpuolelle jäävää aluetta A:lla ja reunantunnistuksen tuloksena saatujen rajojen sisäpuolle jäävää aluetta B:llä, reunantunnistuksen onnistumisaste saadaan seuraavalla kaavalla.

$$P(I, d, p) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A| + |B| - |A \cap B|} \quad (4.2.1)$$

Kaavassa I on alkuperäinen kuva, d on käytettävä reunantunnistusalgoritmi ja p on algoritmin parametrit. Merkintä $|A|$ tarkoittaa alueen A pinta-alaa. Toisin sanoen lasketaan A:n ja B:n yhteinen alue eli leikkaus ja jaetaan se A:n ja B:n yhdistetyllä alueella eli unionilla. Tässä on huomattavaa, että koska nimittäjässä käytetään unionia, niin vääristä positiivisista havainnoista rangaistaan. Siis jos reunantunnistus ja sitä kautta rajantunnistus on löytänyt pisteitä oikean alueen ulkopuolelta, niin siitä seuraa pienempi tulos, koska nimittäjä kasvaa. Kaavan tulos vaihtelee välillä [0,1], missä 0 tarkoittaa täydellistä epäonnistumista, jolloin yhtään reunapistettä ei löydetty, ja 1 puolestaan tarkoittaa täydellistä onnistumista, jolloin kaikki reunapisteet löytyivät eikä tullut yhtään vääriä havaintoja.

Wang ja muut [2006] ottivat vertailtavakseen Sobelin, Marrin ja Hildrethin, Cannyn, Rothwellin [Rothwell et al., 1995] ja Edison [Meer and Georgescu, 2001] algoritmit. Wang ja muut [2006] havaitsivat testeissään, että algoritmien parametreilla on suuri vaikutus reunantunnistuksen lopputulokseen. He kokeilivat kullekin algoritmille viittä eri parametrijoukkoa ja havaitsivat, että millään algoritmilla ei ollut yhtä tiettyä parametrijoukkoa, joka olisi antanut parhaan tuloksen kaikille kuville. Algoritmien väliseen vertailuun he valitsivat kullekin algoritmille sen parametrijoukon, joka antoi keskimääräisesti parhaan lopputuloksen.

Wang ja muut [2006] suorittivat varsinaisen vertailun siten, että jokaisesta kuvasta etsittiin reunapisteet jokaisella algoritmilla käyttäen kullekin algoritmille keskimääräisesti parhaita parametreja. Löydetyistä reunapisteistä muodostettiin raja Wangin ja muiden [2005] kehittämällä menetelmällä. Löydettyä rajaa verrattiin ennalta etsittyihin rajoihin käyttäen kaavaa 4.2.1 ja se algoritmi, joka sai parhaan tuloksen, valittiin voittajaksi kyseiselle kuvalle. Näin käytiin koko aineisto läpi. Wang ja muut [2006] suorittivat saman testin myös asettamalla kaavan 4.2.1 tulokselle alarajoja, jolloin he pääsivät vertailemaan algoritmeja myös tapauksissa, joissa reunantunnistusaste oli hyvä. Wangin ja muiden [2006] tulokset ovat esitettynä taulukossa 4.2.1.

Alaraja	Sobel	LoG	Canny	Rothwell	Edison	Kuvien lkm
> 0	230	184	213	219	184	1030
> 0.35	189	145	177	180	161	852
> 0.55	140	108	128	131	129	636
> 0.75	70	45	73	70	77	335
> 0.90	24	9	27	33	40	133

Taulukko 4.2.1: Wangin ja muiden [2006] testitulokset. Voittojen lukumäärä.

Wang ja muut [2006] päätyivät tulokseen, että kaikki tutkimuksessa mukana olleet reunantunnistusalgoritmit eli Sobel, LoG, Canny, Rothwell ja Edison tunnistivat reunoja suunnilleen samalla tarkkuudella, mutta kunkin algoritmin oletusarvoiset parametrit olivat kaukana optimaalisesta. Oletusarvoilla he tarkoittivat silloisen MATLABin oletusparametreja. Heidän mukaansa yksi jatkokehityksen suunta voisikin olla algoritmien parametrien dynaaminen asetus perustuen kuvien ominaisuuksiin.

Wangin ja muiden [2006] tutkimuksen tulos on sinänsä hämmästyttävä, koska se on ristiriidassa sen yleisen käsityksen kanssa, jonka mukaan esimerkiksi Cannyn algoritmi olisi huomattavasti parempi kuin alkeellinen Sobel. Esimerkiksi Park ja Murphey [2008] sanovat, että Cannyn algoritmi tuottaa yksityiskohtaisemman tuloksen kuin LoG tai Sobel. Wang ja muut [2006] sanovat itsekin, että esimerkiksi Heath ja muut [1996] todistivat Cannyn algoritmin olevan Sobelia parempi subjektiivisesti arvioituna. Miksi Wang ja muut [2006] sitten päätyivät toisenlaiseen tulokseen?

Yksi mahdollisuus voisi olla, että heidän [Wang et al., 2006] käyttämänsä algoritmien pisteytys on liian raaka. Siinä valitaan kutakin kuvaa kohti vain yksi algoritmi, joka sai parhaan tuloksen, ja muut algoritmit jäävät ilman pisteitä, vaikka ne olisivat suoriutuneet reunantunnistuksesta lähes yhtä hyvin kuin paras. Olisi ollut yksinkertaista laskea kaikille algoritmeille kaavan 4.2.1 tulokset ja vaikkapa laskea ne yhteen tai tehdä tilastollista vertailua esimerkiksi keskiarvojen ja -hajontojen avulla. Tämä olisi ollut reilumpi menetelmä kuin Wangin ja muiden [2006] käyttämä.

Toinen mahdollisuus voisi olla, että Wangin ja muiden [2005] kehittämä rajanmuodostusalgoritmi pystyy muodostamaan rajan myös pienemmällä määrällä pikseleitä. Näin kehittyneempien algoritmien tunnistamat lisäpikselit eivät oleellisesti parantaisi tulosta. Jos näin on, niin voidaan kyseenalaistaa rajantunnistuksen soveltuvuus reunantunnistuksen mittariksi. Näin siksi, että reunantunnistuksen tehtävänä on etsiä mahdollisimman monta reunapikseliä. Se käytetäänkö kaikkia löytyneitä pikseleitä vai ei, on prosessin myöhempien vaiheiden päätettävissä, mutta reunantunnistus on luonteeltaan esiprosessointia ja sen tulee toimia mahdollisimman hyvin.

4.3. Ihmisten tekemiin arviointeihin perustuva menetelmä

Heathin ja muiden [1996] mukaan reunantunnistusalgoritmien kvantitatiivinen vertailu edellyttää jonkin yksiselitteisesti mitattavan ominaisuuden olemassaoloa. Esimerkiksi Wang ja muut [2006] etsivät itse kuvasta oikeat rajat ja sitten vertailivat reunantunnistusalgoritmien avulla löydettyjä rajoja niihin. Heathin ja muiden [1996] mukaan tällaisten ominaisuuksien asettaminen luonnollisille kuville ei ole realistista. Jos ajatellaan esimerkiksi ramppireunaa, jossa intensiteetin muutos tapahtuu usean pikselin levyisellä alueella, mikä kyseisistä pikseleistä edustaa oikeaa reunaa? Myös Wang ja muut [2006] joutuivat asettamaan rajoituksia käyttämilleen testikuville, jotta niistä olisi löytynyt jokin kappale, jonka sai selvästi erotettua taustasta. Synteettisistä kuvista voitaisiin helposti määritellä oikea reuna, mutta Heathin ja muiden [1996] mukaan useimmat tutkijat haluavat silti tehdä vertailua myös luonnollisilla kuvilla. Lisäksi Wangin ja muiden [2006] mukaan synteettisillä kuvilla saadut tulokset eivät päde luonnollisilla kuvilla.

Heathin ja muiden [1996] mukaan muun muassa edellisistä syistä johtuen useimmat tutkijat tekevätkin algoritmien vertailua silmämääräisesti asettamalla kuvia vierekkäin, ja on hyväksytty käytäntö esittää uusien algoritmien tuloksia esittelemällä kuvia uuden algoritmin löytämistä reunoista verrattuna muiden algoritmien tuloksiin. Heath ja muut [1996 ja 1997] lähtivät tutkimaan, miten hyvin ihmisiä voisi käyttää reunantunnistusalgoritmien tuloksien vertailuun. He julkaisivat alustavan tutkimuksen vuonna 1996 ja sille jatkoa vuonna 1997. Tässä luvussa keskitytään lähinnä tuohon jälkimmäiseen tutkimukseen.

Heath ja muut [1997] päättivät käyttää joukkoa ihmisiä algoritmien evaluontifunktiona, käyttää testeissä vain luonnollisia kuvia ja asettaa algoritmien parametrit siten, että mitään algoritmia ei suosita. Testattaviksi algoritmeiksi he valitsivat Cannyn [1986], Nalwan [Nalwa and Binford, 1986], Iversonin [Iverson and Zucker, 1995], Bergholmin [1987] ja Rothwellin [Rothwell et al., 1995] algoritmit. Nämä algoritmit he valitsivat, koska ne olivat riittävän erilaisia, ne olivat kaikki julkaistu alan tieteellisissä julkaisuissa ja niihin kaikkiin oli saatavilla lähdekoodi.

Heath ja muut [1997] valitsivat vertailtavat kuvat siten, että niissä oli vallitsevana ihmiselle helposti tunnistettava esine kuten esimerkiksi eväskori. Lisäksi jotta kuvat olisivat olleet mahdollisimman monipuolisia, ne jaettiin neljään eri kategoriaan: ihmisen tekemä esine tekstuurilla, ihmisen tekemä esine ilman tekstuuria, luonnollinen kohde tekstuurilla ja luonnollinen kohde ilman tekstuuria. Kaiken kaikkiaan kuvia oli yhteensä 28.

Algoritmien parametrien valinta on kriittinen vaihe algoritmien suorituskyvyn vertailussa, koska ne vaikuttavat erittäin paljon lopputulokseen [Heath et al., 1997]. Aluksi Heath ja muut [1997] valitsivat 64 eri parametrijoukkoa kullekin algoritmille perustuen algoritmien kehittäjien suosituksiin. Tämän jälkeen he ajoivat algoritmit kaikilla parametrijoukoilla kaikille kuville. Joku tutkimusryhmän ulkopuolinen henkilö toimi esikarsijana ja valitsi kullekin kuva-algoritmi-parille viisi parasta kuvaa. Lopuksi Heath ja muut [1997] käyttivät ahnetta algoritmia valitsemaan 12 parametrijoukkoa, jotka olivat esiintyneet eniten edellisessä kohdassa valituissa kuvissa. Näin he saivat valittua kullekin algoritmille edustavan joukon parametreja.

Heath ja muut [1997] ajoivat kaikki 28 kuvaa kaikkien algoritmien läpi kaikilla 12:lla parametrijoukolla ja pyysivät sitten yhdeksää vapaaehtoista opiskelijaa arvioimaan tuloksia. Opiskelijat olivat kaikki opiskelleet jonkin verran konenäköä, joten reunantunnistus oli heille tuttua. Opiskelijoita pyydettiin arvioimaan kustakin tuloskuvasta, miten hyvin he pystyivät tunnistamaan kuvan esineen. Asteikko oli yhdestä seitsemään, missä yksi tarkoitti, että esine ei ole ollenkaan tunnistettavissa ja seitsemän, että esine on helposti ja nopeasti tunnistettavissa. Mitään aika-rajaa ei asetettu arviointien tekemiseen.

Kun arvioinnit oli tehty Heath ja muut [1997] suorittivat tuloksille muutamia tilastollisia testejä. Aluksi he tutkivat, miten hyvin eri arvioijien arviot vastasivat toisiaan. Heath ja muut [1997] havaitsivat, että vastauksien välillä oli vahva korrelaatio, mikä oli tärkeä tulos, jotta ihmisiä voitaisiin käyttää luotettavasti tulosten arvioinnissa. Heathin ja muiden [1997] mukaan arvioijilla vaikutti olevan yhteinen käsitys siitä, millainen on hyvä reuna. He havaitsivat myös, että ne viisi parasta kuvaa, jotka esikarsija valitsi aluksi kullekin algoritmille, saivat hyvät pisteet myös varsinaisilta arvioijilta.

Pisteiden perusteella Heath ja muut [1997] laskivat kullekin algoritmille keskimääräisesti parhaat parametrit ja kuvakohtaiset mukautetut parametrit. Näillä parametreilla he ajoivat jälleen kaikki algoritmit kaikille kuville saaden lopulliset testikuvat. Tällä kertaa 16 ihmistä pisteytti kaikki kuvat. Asteikko oli yhdestä seitsemään kuten edellisessäkin testissä.

Heath ja muut [1997] asettivat tutkimukselleen lähtökohdaksi löytää vastaukset seuraaviin kysymyksiin:

1. Paraneeko algoritmien tulos huomattavasti, kun käytetään mukautettuja parametreja vakioparametrien (keskimääräisesti parhaiden parametrien) sijaan?
2. Mikä on algoritmien paremmuusjärjestys?
3. Vaikuttavatko vertailussa käytettävät kuvat vertailun lopputulokseen?

Heath ja muut [1997] saivat mukautettuja parametreja käyttäen tuloksien keskiarvoksi 4.60 ja vakioparametreilla 4.15. Tilastollisia menetelmiä käyttäen he todistivat, että ero oli merkittävä. Mukautetut parametrit tuottivat siis merkittävästi parempia tuloksia kuin vakioparametrit. Wang ja muut [2006] päätyivät samaan tulokseen.

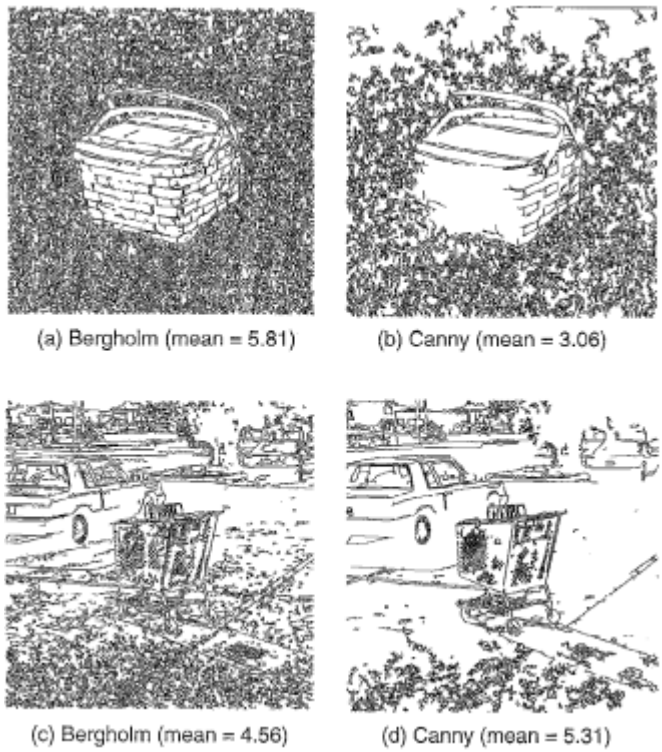
Heath ja muut [1997] tutkivat algoritmien paremmuusjärjestystä kahdessa eri osassa. Ensimmäisessä osassa he käyttivät mukautettuja parametreja ja toisessa osassa vakioparametreja. Testien tulokset ovat listattuna taulukossa 4.3.1. Mukautettujen parametrien testissä Heath ja muut [1997] päätyivät tulokseen, missä Cannyn, Bergholmin ja Rothwellin algoritmit suoriutuivat tilastollisesti merkittävästi paremmin kuin Nalwan ja Iversonin algoritmit. Vakioparametreilla puolestaan Cannyn algoritmi suoriutui huonoiten ollen yhdessä Nalwan algoritmin kanssa merkittävästi huonompia kuin Bergholmin algoritmi. Vakioparametritestissä Iversonin ja Rothwellin algoritmit eivät eronneet merkittävästi muista algoritmeista [Heath et al., 1997].

Mukautetut parametrit	Vakioparametrit
Canny (4.80)	Bergholm (4.38)
Bergholm (4.78)	Iverson (4.24)
Rothwell (4.76)	Rothwell (4.21)
Nalwa (4.43)	Nalwa (3.97)
Iverson (4.26)	Canny (3.96)

Taulukko 4.3.1: Heathin ja muiden [1997] testitulokset algoritmien paremmuudesta.

On mielenkiintoista havaita, kuinka paljon parametrit vaikuttavat Cannyn algoritmin tulokseen. Mukautetuilla parametreilla Cannyn algoritmi suoriutuu hyvin, mutta vakioparametreilla huonosti. Iversonin algoritmiin parametreilla ei tuntuisi olevan vaikutusta, koska mukautetut parametrit antoivat vain 0.02 pistettä paremman tuloksen. Muilla algoritmeilla tulokset heikkenevät, kun käytetään vakioparametreja, mutta muutos ei ole niin radikaali kuin Cannyn algoritmilla.

Heathin ja muiden [1997] mukaan sekä mukautettuja että vakioparametreja käyttäen tulokset riippuivat käytetyistä kuvista. Esimerkiksi kuvassa 4.3.1 on esimerkkipari Cannyn ja Bergholmin algoritmien tuloksista kahdelle eri kuvalle vakioparametreja käyttäen. Eväskorista Cannyn algoritmi on saanut pisteikseen 3.06 ja Bergholmin algoritmi 5.81. Ostoskärrykuvassa järjestys vaihtuu ja Cannyn algoritmi saa 5.31 pistettä, kun Bergholmin algoritmi saa 4.56.



Kuva 4.3.1: Kahden kuvan testitulokset vakioparametreja käyttäen [Heath et al., 1997].

Heath ja muut [1997] tarkastelivat myös algoritmien tuloksia eri kuvaluokkien sisällä. Hehän jakoivat kuvat neljään eri luokkaan sen perusteella, oliko niissä kuvatuissa esineissä tekstuuria vai ei, ja oliko kuvassa jokin ihmisen valmistama esine vai luonnossa esiintyvä kappale. Näihin luokkiin jäi kuhunkin kuitenkin liian vähän kuvia, jotta testeillä olisi ollut tilastollista merkittävyyttä [Heath et al., 1997].

Yhteenvetona Heathin ja muiden [1997] mukaan algoritmien tulos paranee, jos parametrit mukautetaan kuvakohtaisesti. Lisäksi vertailussa käytettävät kuvat vaikuttavat vertailun lopputulokseen. Testatuista algoritmeista ei löytynyt selvää voittajaa, mutta vakioparametreilla Bergholmin algoritmi oli merkittävästi parempi kuin Cannyn tai Nalwan algoritmit. Mukautetuilla parametreilla puolestaan Rothwellin, Bergholmin ja Cannyn algoritmit olivat merkittävästi parempia kuin Iversonin ja Nalwan algoritmit.

Heath ja muut [1997] antavat myös neuvoja, miten heidän menetelmäänsä käyttäen voi testata uusia algoritmeja. He suosittelevat, että testissä käytettäisiin vähintään kahta samaa algoritmia kuin heidän testissään uuden algoritmin lisäksi. Luonnollisesti tarvitaan myös riittävän monta ihmistä arvioimaan kuvia. Heath ja muut [1997] arvioivat, että testien tekemiseen menisi kokonaisuudessaan kolmesta neljään viikkoa.

Menetelmän suurin ongelma lienee suuri resurssien tarve. Se vaatii monta ihmistä arvioimaan kuvia. Toisaalta tänä päivänä testikuvat voisi laittaa verkkoon, jossa vapaaehtoiset voisivat käydä niitä arvioimassa. Siten voitaisiin saada myös suurempi määrä testaajia, jolloin tilastollinen merkittävyys paranisi. Toinen mahdollinen ongelma liittyy testaajiin. Jotkut ovat esittäneet kritiikkiä ihmisten käyttämisestä arvioijina [Lopez-Molina et al., 2013] ja ehkä se on yksi syy, miksi tämän menetelmän käyttö ei ole yleistynyt.

4.4. Virheiden mittaamisesta

Lopez-Molina ja muut [2013] tutkivat millaisia mittareita on kehitetty reunantunnistusalgoritmien tekemien virheiden mittaamiseen. Heidän mukaansa reunantunnistuksen tuloksen arvioimiseen on kvalitatiivisia, kvantitatiivisia ja hybridimenetelmiä. Kvalitatiiviset menetelmät perustuvat ihmisten tekemiin arviointeihin ja kuvauksiin. Kvantitatiivisissa menetelmissä tulokselle lasketaan jokin numeerinen arvo, jota sitten käytetään vertailtaessa eri tuloksia. Hybridimenetelmä on kvalitatiivisen ja kvantitatiivisen menetelmän sekoitus, esimerkiksi Heathin ja muiden [1997] menetelmä.

Lopez-Molina ja muut [2013] tulivat tutkimuksessaan siihen johtopäätökseen, että ihmisiä ei tule käyttää tulosten vertailuun, vaan pitää olla jokin mitta, jolla voidaan mitata, mikä algoritmi suoriutui parhaiten. Näin ollen he keskittyivät tutkimaan kvantitatiivisia menetelmiä ja niissä käytettyjä mittoja. Tällaista mittaa voitaisiin käyttää myös yksittäisen algoritmin parametrien optimointiin, jolla on suuri merkitys, kuten myös Heath ja muut [1997] ja Wang ja muut [2006] totesivat.

Lopez-Molinan ja muiden [2013] mukaan yleisimmin hyväksytyt kriteerit reunantunnistuksen tulokselle ovat Cannyn [1986] asettamat kolme kriteeriä: hyvä havaitsemiskyky, hyvä paikallistamiskyky ja vain yhden pikselin vahvuinen reuna. Vaikkakin Cannyn kriteereitä on arvosteltu, niin kritiikki on kohdistunut lähinnä kriteereiden mallinnukseen eikä niinkään semantiikkaan [Lopez-Molina et al., 2013].

Lopez-Molinan ja muiden [2013] mukaan Cannyn kriteereistä kahta ensimmäistä käytetään reunantunnistuksen virheiden mittaamiseen. Kolmatta kriteeriä, joka edellyttää yhden pikselin vahvuista reunaa, ei mitata, koska tyypillisesti reunantunnistimet eivät palauta paksuja reunoja. Näin ollen reunantunnistusalgoritmien tekemät virheet voidaan jakaa seuraaviin kolmeen eri luokkaan:

1. *Väärä positiivinen*. Algoritmi tunnistaa reunapisteeksi pisteen, joka ei ole oikeasti reunapiste. Nämä ovat yleensä seurausta kohinasta tai kuvassa esiintyvistä tekstuureista.

2. *Väärä negatiivinen*. Algoritmi ei tunnista reunapistettä. Näitä ilmenee tyypillisesti matalan kontrastin alueilla, joissa intensiteetin muutokset ovat pieniä.
3. *Väärä sijainti*. Löydetyt reunapisteet saattavat mennä väärään kohtaan, jos alkuperäisestä kuvasta on suodatettu kohinaa ennen reunantunnistusta, kuten usein tehdään.

Jotta virhettä voitaisiin mitata, täytyy tietää *oikea vastaus* (eng. ground truth). Lopez-Molina ja muut [2013] määrittelivät oikeaksi vastaukseksi minkä tahansa reunan, jonka joku ihminen merkitsee reunaksi. Näin ollen oikeita vastauksia voi olla useita erilaisia. Lopez-Molinan ja muiden [2013] mukaan tämä menettely on kuitenkin parempi, kuin muut menettelyt, joissa vertailu tehdään jonkin toisen algoritmin tulokseen tai vertailu usean algoritmin keskiarvoon. Vertailu muiden algoritmien tuloksiin nimittäin rankaisee algoritmeja, jotka ovat parempia kuin oikean vastauksen tuottaneet algoritmit.

Mitat, joilla virhettä mitataan, voidaan jakaa kahteen luokkaan. Ensimmäinen luokka on sovelluskohtaiset mitat, joilla mitataan, kuinka hyvä tulos on sovelluksen tarkoituksiin. Näiden käyttö yleiseen vertailuun ei ole tietenkään mielekästä. Sen sijaan yleisiä mittoja voidaan käyttää ja ne voidaan jälleen jakaa kolmeen eri luokkaan: paikalliset, tilastolliset ja etäisyyteen perustuvat virhemitat. [Lopez-Molina et al., 2013]

Lopez-Molinan ja muiden [2013] mukaan paikallisilla virhemitoilla mitataan naapuruston ominaisuuksia kuten reunan jatkuvuutta ja paksuutta. Paikalliset virhemitat eivät tarvitse oikeaa vastausta, mikä ei täytä Lopez-Molinan ja muiden [2013] vaatimuksia. He kuitenkin toteavat, että paikallisia virhemittoja voidaan hyvin käyttää tilanteissa, missä oikeaa vastausta ei ole tiedossa ja näin voidaan lopputulosta parantaa huomattavasti. Lopez-Molinan ja muiden [2013] mukaan paikalliset virhemitat mittaavatkin kuinka hyvältä reuna näyttää, mutta eivät kuinka tarkasti reuna vastaa alkuperäisen kuvan reunaa.

Tilastollisilla mitoilla mitataan käytännössä väärien positiivisien ja väärien negatiivisien suhteita oikeisiin positiivisiin ja oikeisiin negatiivisiin. Lopez-Molinan ja muiden [2013] mukaan erilaisia menetelmiä on useita, mutta ne kaikki kärsivät samasta ongelmasta, missä esimerkiksi väärän positiivisen etäisyydellä oikeaan reunaan ei ole merkitystä.

Etäisyyteen perustuvat virhemitat mittaavat löydettyjen pisteiden etäisyyttä oikeasta reunasta. Lopez-Molina ja muut [2013] osoittavat, että nämäkään mittarit eivät ole täydellisiä, sillä kahdella erilaisella *oikealla vastauksella* saadaan aikaiseksi sama

mittaustulos. Lisäksi näiden mittareiden heikkoutena on herkkyys väärille positiivisille, joista saatetaan rangaista liikaa. Lopez-Molinan ja muiden [2013] mukaan täytyisi kehittää menetelmä, jolla esimerkiksi kohinan aiheuttamat väärät positiiviset voitaisiin erottaa väärään paikkaan menneistä reunapisteistä.

Pelkän teoreettisen tarkastelun lisäksi Lopez-Molina ja muut [2013] suorittivat myös empiirisiä testejä eri virhemitoilla. Testit osoittivat epäilyt todeksi ja he päätyivät tulokseen, että tällä hetkellä ei ole vakuuttavaa tapaa mitata reunantunnistuksen tuloksia kvantitatiivisesti. Tästä huolimatta mittoja voidaan käyttää, jos ottaa huomioon niiden heikkoudet tai niillä ei ole sovelluksen kannalta merkitystä. Algoritmien yleiseen vertailuun niistä ei kuitenkaan sellaisenaan ole. [Lopez-Molina et al., 2013]

5. Yhteenveto

Tässä tutkielmassa esittelin reunantunnistuksen ja kuvanprosessoinnin perusteita. Alalle on vuosien saatossa kertynyt hyviä tekniikoita, mutta alan termistö ei ole vielä täysin vakiintunut ja samasta asiasta saatetaan käyttää useaa eri termiä.

Reunantunnistus on tutkimusalue, jonka tuloksia on helppo kritisoida. Reunantunnistuksen ongelmia on yritetty ratkaista vuosikymmeniä, mutta vieläkään ei ole löytynyt täydellistä ratkaisua. Suurin syy tähän on varmasti se, että pelkästään reunan määrittäminen sopivan hankalasta kuvasta on jopa ihmiselle mahdotonta tai jos ei mahdotonta, niin vastaus ei missään tapauksessa ole yksiselitteinen.

Vuosien aikana ongelmaa on yritetty ratkaista lukuisin algoritmein. Esimerkiksi Google Scholar [2014] löytää haulla ”edge detection” 246 000 artikkelia. Tässä tutkielmassa esittelin tunnetuimmat algoritmit: Roberts, Sobel, Prewitt, LoG ja Canny. Kaikista algoritmeista Cannyn [1986] algoritmi on vakiinnuttanut asemansa parhaana pidettynä algoritmina, mutta tämä on pikemminkin alan perinne kuin minkään testin tulos.

Jain ja Binford [1991] herättelivät konenäön tutkijoita erittäin kriittisellä kirjoituksellaan. He peräänkuuluttivat mm. mittareiden luomista, jotta luotettavaa vertailua algoritmien välillä päästäisiin tekemään ja sitä kautta muuttamaan alaa tieteellisemmäksi. Sittemmin tällaisia mittareita on yritetty luoda. Tässä tutkielmassa esittelin Heathin ja muiden [1997] ja Wangin ja muiden [2006] ehdottamat mittarit. Molemmissa tutkimuksissa käytettiin hyviä menetelmiä, mutta ne ovat silti olleet alttiita kritiikille ja kumpikaan niistä ei ole päätynyt laajempaan käyttöön. Tällä hetkellä ei ole olemassa luotettavaa menetelmää mitata reunantunnistuksen onnistumisen astetta [Lopez-Molina et al., 2013].

Kaikesta kritiikistä huolimatta reunantunnistusta tehdään päivittäin onnistuneesti ympäri maailmaa. Vaikka täydellistä ratkaisua ei ehkä koskaan löydettäisikään, niin riittävän hyviä ratkaisuja kuitenkin. Kuten Heath ja muut [1997] osoittivat, algoritmien parametreja säätämällä tulokset paranevat huomattavasti. Reunantunnistusta voidaan myös helpottaa esimerkiksi teollisessa ympäristössä järjestämällä hyvät valaistus-olosuhteet, jolloin ylimääräiset häiriötekijät minimoituvat.

Tutkielmani painottui tahattomasti 1980- ja 1990-lukujen algoritmeihin. Tämä johtui ensinnäkin siitä, että alan klassisista algoritmeista LoG ja Canny ovat molemmat 1980-luvun tuotoksia, mutta ovat iästään huolimatta edelleen laajasti käytössä. Toisekseen kummatkin algoritmeja vertailevat tutkimukset, jotka kävin läpi, vertailivat tuon ikäluokan algoritmeja.

Tänä päivänä aallokemuunnokseen perustuvat algoritmit tuntuvat olevan yksi tutkituimmista suuntauksista. Edellä mainittuihin yli 200 000:een artikkeliin sisältyy monia tässä tutkielmassa käsittelemättä jätettyjä mielenkiintoisia tapoja yrittää ratkaista reunantunnistuksen ongelmaa. Näin ollen tätä tutkielmaa ei pidä millään muotoa pitää kaiken kattavana katsauksena reunantunnistuksen maailmaan, mutta aiheen perusmenetelmät on tässä työssä käsitelty.

Viiteluettelo

[Badaud et al., 1986] Jean Badaud, Andrew P. Witkin, Michel Baudin and Richard O. Duda, Uniqueness of the Gaussian Kernel for Scale-Space Filtering, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **8**, 1, (Jan 1986), 26-33.

[Basu, 2002] Mitra Basu, Gaussian-based edge-detection methods - a survey, *IEEE Trans. Systems, Man, and Cybernetics*, **32**, 3, (Aug 2002), 252-260.

[Bergholm, 1987] Fredrik Bergholm, Edge focusing, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **6**, (1987), 726-741.

[Canny, 1986] John F. Canny, A computational approach to edge detection, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **8**, 6, (Nov 1986), 679-698.

[Deriche, 1987] Rachid Deriche, Using Canny's criteria to derive a recursively implemented optimal edge detector, *International Journal of Computer Vision*, **1**, 2, (1987), 167-187.

[Ding and Goshtasby, 2001] Lijun Ding and Ardeshir Goshtasby, On the Canny edge detector, *Pattern Recognition*, **34**, 3 (2001): 721-725.

[Gonzales and Woods, 2008] Rafael C. Gonzalez and Richard E. Woods, *Digital Image Processing (Third Edition)*, Pearson Education, Inc., 2008.

[Google Scholar, 2014] Google Scholar, *http://scholar.google.fi/*, (viitattu 17.6.2014)

[Heath et al., 1997] Michael Heath, Sudeep Sarkar, Thomas Sanocki and Kevin Bowyer, A robust visual method for assessing the relative performance of edge-detection algorithms, *IEEE Transactions on Pattern Analysis and Machine Intelligence,* **19**, 12 (1997), 1338-1359.

[Heath et al., 1996] Michael Heath, Sudeep Sarkar, Thomas Sanocki and Kevin Bowyer, Comparison of edge detectors: a methodology and initial study, *Computer Vision and Pattern Recognition,* 1996 IEEE Computer Society Conference on Proceedings CVPR'96, (1996), 143-148.

[Heric and Zazula, 2007] Dušan Heric and Damjan Zazula, Combined edge detection using wavelet transform and signal registration, *Image and Vision Computing*, **25**, 5, (May 2007), 652-662.

[Iverson and Zucker, 1995] Lee Iverson and Steven Zucker, Logical/linear operators for image curves, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **17**, 10, (1995), 982-996.

[Jain and Binford, 1991] Ramesh Jain and Thomas Binford, Ignorance, myopia, and naivete in computer vision systems, *CVGIP: Image Understanding*, **53**, 1, (1991), 112-117.

[Kirsch, 1971] Russell A. Kirsch, Computer determination of the constituent structure of biological images, *Computers and Biomedical Research,* **4**, (1971), 315-328.

[Lopez-Molina et al., 2013] Carlos Lopez-Molina, Bernard De Baets and Humberto Bustince, Quantitative error measures for edge detection, *Pattern Recognition*, **46**, 4, (2013) 1125-1139.

[Marr and Hildreth, 1980] David Marr and Ellen Hildreth, Theory of edge detection, *Proceedings of the Royal Society of London.*, Series B, Biological Sciences, **207**, 1167, (Feb 29, 1980), 187-217.

[MATLAB, 2014] MATLAB, *http://www.mathworks.se/help/images/ref/edge.html*, (viitattu 9.3.2014)

[Meer and Georgescu, 2001] Peter Meer and Bogdan Georgescu, Edge detection with embedded confidence, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **23**, 12 (2001), 1351-1365.

[Nalwa and Binford, 1986] Vishvjit Nalwa and Thomas Binford, On detecting edges, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **6**, (1986), 699-714.

[Octave, 2013] GNU Octave v3.6.4, *www.octave.org*, (viitattu 26.12.2013)

[Oskoei and Hu, 2010] Mohammadreza Asghari Oskoei and Huosheng Hu, A survey on edge detection methods, *Technical Report CES*,Vol. **506**, 2010.

[Park and Murphey, 2008] Jung Me Park and Yi Lu Murphey, *Wiley Encyclopedia of Computer Science and Engineering*, John Wiley & Sons, Inc., 2008.

[Robinson, 1977] Güner S. Robinson, Edge detection by compass gradient masks, *Computer Graphics and Image Processing*, **6**, 5, (1977), 492-501.

[Rothwell et al., 1995] Charlie Rothwell, Joe Mundy, Bill Hoffman and Van-Duc Nguyen, Driving vision by topology, *Computer Vision, 1995. Proceedings., International Symposium on. IEEE*, (1995), 395-400

[Shapiro and Stockman, 2001] Linda G. Shapiro and George C. Stockman, *Computer Vision*, Prentice-Hall, Inc., 2001.

[USC-SIPI, 2014] The USC-SIPI Image Database, *http://sipi.usc.edu/database/*, (viitattu 24.6.2014)

[Wang et al., 2005] Song Wang, Toshiro Kubota, Jeffrey Mark Siskind and Jun Wang, Salient closed boundary extraction with ratio contour, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **27**, 4, (2005), 546-561.

[Wang et al., 2006] Song Wang, Feng Ge and Tiecheng Liu, Evaluating Edge Detection through Boundary Detection, *EURASIP Journal on Advances in Signal Processing*, **2006**, 1, (2006), 1-15.

[Zhai et al., 2008] Lei Zhai, Shouping Dong and Honglian Ma, Recent methods and applications on image edge detection, *ETT and GRS 2008. International Workshop on Geoscience and Remote Sensing. IEEE*, **1**, (2008), 332-335.

[Wikipedia, 2014] Wikipedia, *http://fi.wikipedia.org/wiki/Aallokemuunnos*, (viitattu 8.6.2014)