TAMPEREEN TEKNILLINEN YLIOPISTO
TAMPERE UNIVERSITY OF TECHNOLOGY

ANSSE SAARIMÄKI

KAMERAPOHJAINEN PAIKANNUS

Kandidaatintyö

Tarkastaja: Yliopistonlehtori Heikki Huttunen
Jätetty tarkastettavaksi 21.5.2015

TIIVISTELMÄ

ANSSE SAARIMÄKI: Kamerapohjainen paikannus
Tampereen teknillinen yliopisto
Kandidaatintyö, 26 sivua, 0 liitesivua
Toukokuu 2015
Signaalinkäsittelyn ja tietoliikennetekniikan koulutusohjelma
Pääaine: Signaalinkäsittely ja multimedia
Tarkastaja: Yliopistonlehtori Heikki Huttunen
Avainsanat: konenäkö, paikannus

Työssä käydään läpi videokuvaan pohjautuvan paikannuksen taustalla oleva teoria ja esitellään toteutettu paikannusalgoritmi, joka määrittää kameran sijainnin ja asennon ympäristöönsä nähden. Algoritmin toiminta perustuu määrätyn kuvion seurantaan ja etsintään videokuvan yksittäisistä kuvakehyksistä. Kuvio on jokin kameran ympäristöön sijoitettu taso, jonka sijainti ei muutu paikannuksen aikana. Kuviosta valitaan yksittäisiä piirteitä, joita etsitään videon yksittäisestä kuvakehyksestä. Kuvasta löytyneiden seurantakuvion piirteiden sijainnin pohjalta määritetään seurantakuvion sijainti kuvassa. Kun tiedetään kuinka kamera projisoi ympäristön kohteiden kolmiulotteiset koordinaatit kameran kuvatason kaksiulotteisiksi koordinaateiksi, voidaan määrittää seurattavan kohteen sijainti kameraan nähden. Kun kuvion tiedetään pysyvän paikallaan ympäristön koordinaatistossa, voidaan kameran asento ja sijainti kohteeseen nähden määrittää helposti.

Työssä toteutetun algoritmin tavoitteena oli reaaliaikainen toiminta 30Hz videokuvalla, joka saavutettiin käytetyllä testilaitteistolla. Algoritmin paikannuksen tarkkuus todettiin riittäväksi sille suotuisissa testitapauksissa, eikä sen tarkkaan mittaamiseen tai optimointiin perehdytty työssä. Algoritmi toimii vain hyvin rajoittuneilla liikealueella ja siten myös käyttökohteet ovat hyvin rajalliset. Algoritmin toimintaa on kuitenkin mahdollista parantaa sekä tarkkuuden että mahdollisen liikealueen suhteen yksinkertaisilla laajennuksilla.

ALKUSANAT

Tämä työ sai alkunsa työskennellessäni signaalinkäsittelyn laitoksella tutkimusapulaisena 3D-konenäköä tutkineessa ryhmässä. Haluaisin kiittää silloista esimiestäni Jari Yli-Hietasta mahdollisuudesta tehdä tätä työtä osana ryhmän tutkimusta ja hyödyntää ryhmän laitteistoa työn toteutuksessa. Lisäksi haluaisin kiittää Heikki Huttusta hyödyllisistä neuvoista ja ohjeista sekä erityisesti joustavuudesta ja kärsivällisyydestä työn aikataulujen suhteen.

21.5.2015

SISÄLLYS

LYHENTEET JA MERKINNÄT

SLAM	"Simultanenous localization and mapping", paikannusalgoritmi, joka kartoittaa tuntematonta ympäristöä paikannuksen aikana
FAST	"Features from Accelerated Segment Test", piirteenetsintäalgortimi
BRIEF	"Binary Robust Independent Elementary Features", piirteenkuvausalgoritmi
ORB	"Oriented FAST and Rotated BRIEF", yhdistelmä edellisiä
RANSAC	"Random Sample Consensus", algoritmi poikkeavien näytteiden sulkemiseen pois mallin sovituksesta

1. JOHDANTO

Liikkuvien laitteiden reaaliaikainen paikannus suhteessa ympäristöönsä on ollut jo pitkään tärkeä osa-alue erityisesti robotiikassa, mutta uusia käyttökohteita syntyy jatkuvasti sekä teollisuudessa että kuluttajatekniikassa. Robotiikassa tarve paikannukselle on ilmeinen, sillä autonomisesti liikkuvan laitteen on tiedettävä sijaintinsa navigointia varten. Paikannukseen on kehitetty monia erilaisia menetelmiä, mutta ne voidaan jakaa karkeasti kahteen ryhmään: absoluuttiseen ja suhteelliseen paikannukseen [23].

Suhteellisessa paikannuksessa määritetään laitteen sijainti suhteessa lähtöpisteeseen esimerkiksi odometrian (liikelaskenta) tai inertiasensoreiden (gyroskooppi ja kiihtyvyysanturit) avulla [23]. Odometriassa hyödynnetään esimerkiksi pyörillä liikkuvan robotin renkaisiin yhdistettyjä rotaatioenkoodereita, joilla voidaan mitata renkaiden pyöriminen tarkasti. Siitä voidaan helposti laskea mihin suuntaan robotti on liikkunut ja kuinka paljon. Menetelmä on kuitenkin altis virheille esim. renkaan pyöriessä ilman pitoa. Odometriassa virhe kertautuu rajatta, ja sijaintiarviota on korjattava muilla mittauksilla. Inertiasensoreilla ongelma on vielä suurempi, sillä liikettä arvioidaan integroimalla kiihtyvyyttä nopeudeksi ja nopeutta kuljetuksi matkaksi. Myös virhe kertautuu ja kasvaa nopeasti ajan kuluessa. Inertiasensoreiden käyttöä vaikeuttavat monet erilaiset virhelähteet, jotka on huomioitava laskennassa [10].

Absoluuttiseen paikannuksessa laite paikannetaan suhteessa maamerkkeihin, eli sijainniltaan tunnettuihin kohteisiin nähden, josta tunnetuin esimerkki lienee Global Positioning System (GPS) [23]. Siinä GPS-vastaanottimen maantieteelliset koordinaatit selvitetään mittaamalla etäisyys maan kiertoradalla oleviin paikannussatelliitteihin. Paikannussatelliittien sijainti on ennalta tiedetty, mutta absoluuttista paikannusta voi tehdä myös ilman ennakkotietoa ympäristöstä. Silloin samanaikaisesti etsitään uusia maamerkkejä, määritetään niiden sijainti ja paikannetaan laitetta suhteessa niihin. Se tunnetaan englanninkielisessä kirjallisuudessa lyhenteellä SLAM eli Simultaneous localization and mapping. Se on vapaasti suomennettuna yhtäaikainen paikannus ja kartoitus. SLAM-sovelluksista ei yhtä yleisesti tunnettua esimerkkiä löydy, sillä se on tähän asti rajoittunut lähinnä robotiikan tutkimukseen. Mahdolliset käyttökohteet sille ovat kuitenkin hyvin laajat, esim. ahtaissa sisäti-

loissa, joissa GPS:n tarkkuus tai kuuluvuus ei riitä ja oleellista on tietää sijainti rakennuksessa eikä maantieteellisesti.

SLAM-järjestelmän toiminta vaatii jonkin tavan havainnoida ja mitata ympäristöä, mutta se ei ole itsessään riippuvainen siitä millä tekniikalla se toteutetaan. Robotiikassa on aikaisemmin käytetty pääosin etäisyyssensoreita kuten laseretäisyysmittareita ja kaikuluotaimia, mutta viime vuosina tutkimuksessa on keskitytty yhä enemmän kamerapohjaisiin SLAM-järjestelmiin [7, 15, 8]. Kameroiden käytöstä tekee erityisen houkuttelevaa niiden halpa hinta ja hyvä saatavuus, jonka myötä niitä on integroitu valmiiksi moniin laitteisiin. Kamerapohjaiselle SLAM:ille on sovelluskohteita robotiikan lisäksi myös esim. lisätyssä todellisuudessa [15]. Kasvanut laskentateho ja algoritmien kehitys on mahdollistanut myös koko kamerapohjaisen SLAM-järjestelmän toteutuksen suoraan kamerapuhelimilla [16].

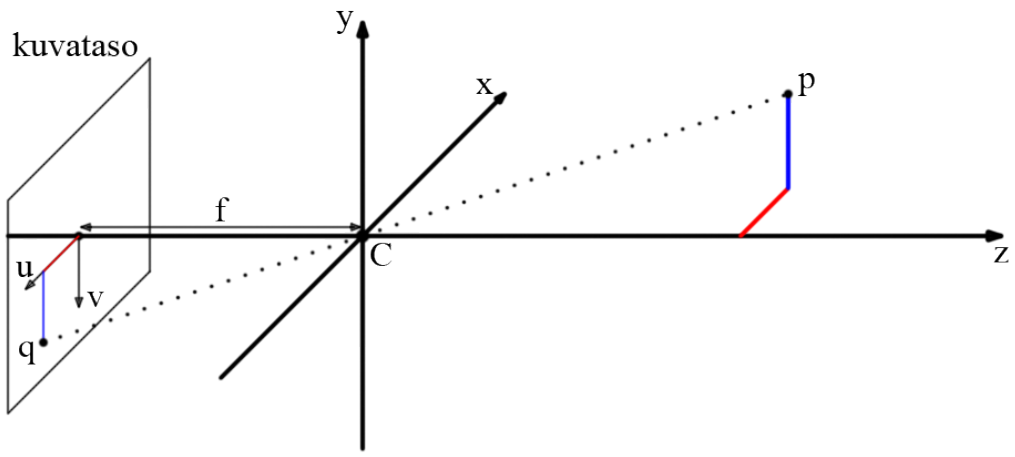
Tässä työssä toteutettava paikannus ei ole määritelmän mukainen SLAM-järjestelmä, sillä paikannukseen käytettävät maamerkit valitaan alustusvaiheessa ja niiden sijainnista tehdään muutamia oletuksia. Varsinaisen SLAM-järjestelmän toteutus olisi kandidaatintyöaiheeksi liian laaja, mutta poistamalla vaatimus tuntemattomasta ympäristöstä toteutus yksinkertaistuu huomattavasti. Työssä paikannusprosessi jakautuu alustukseen ja kolmivaiheiseen paikannukseen. Alustusvaiheessa kamera käännetään kuvaamaan haluttua, seurannassa käytettävää kuviota, josta ohjelma valitsee seurattavat maamerkit. Sen jälkeen aloitetaan paikannus, etsimällä ensin kameran kuvasta piirteitä, joita käytetään etsittävän kuvion paikannuksessa. Sen jälkeen löydettyjä piirteitä verrataan paikannuskuviosta laskettuihin piirteisiin ja yritetään löytää yhteneväisyydet. Lopuksi löydetyistä yhteneväisyyksistä arvioidaan kuvion tarkka sijainti kameran kuvassa ja määritetään kameran kolmiulotteinen sijainti kuvioon nähden. Paikannuksessa siis oletetaan kuvion pysyvän paikallaan todellisen maailman koordinaatistossa. Jotta järjestelmästä saisi täysiverisen SLAM-järjestelmän, pitäisi siihen lisätä vielä uusien maamerkkien automaattinen valinta ja niiden sijainnin määritys.

Työn luvussa 2 käydään läpi työn taustalla oleva teoria. Osiossa 2.1 kerrotaan, kuinka kamera ja sen linssistö piirtävät kolmiulotteisessa maailmassa näkyvät kohteet kaksiulotteiseksi kuvaksi, ja kuinka kuinka kuvattavan kohteen kolmiulotteiset koordinaatit voidaan laskea kuvan pohjalta. Osiossa 2.2 käsitellään yksittäisten visuaalisten piirteiden etsimistä kuvasta, ja niiden kuvausta eli yksilöintiä vertailua varten. Osiossa 2.3 kerrotaan kuinka piirteiden yhteneväisyys lasketaan ja kuinka siitä arvioidaan kameran kolmiulotteinen sijainti suhteessa etsittävään kuvioon. Luvussa 3 käsitellään toteutusta ja siinä hyödynnettyjä ohjelmistotyökaluja ja laitteistoa sekä saatuja tuloksia. Luvussa 4 esitellään tulosten pohjalta tehdyt päätelmät.

2. TEORIA

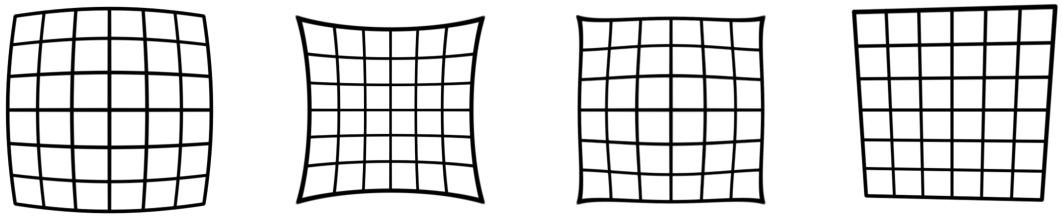
2.1 Kameramalli ja perspektiivinen projektio

Kamerapohjaisen paikannuksen ytimessä on malli, jonka mukaan kamera projisoi kolmiulotteisen maailman koordinaatit kaksiulotteiseksi pisteiksi kameran kuvatasolle, esimerkiksi digitaalisen kameran kuvakennolle. Tässä työssä käytetään mallia, jossa kaikkien valonsäteiden oletetaan kulkevan yhden pisteen (keskipiste) kautta kuvatasolle. Yksi kuvatason piste voi siis todellisessa maailmassa sijaita millä tahansa tämän pisteen ja keskipisteen kautta kulkevalla suoralla. Mallia kutsutaan myös neulansilmäkameramalliksi, sillä se kuvaa ideaalisen neulansilmäkameran toimintaa, jossa kameran valotusaukko (neulansilmä) on äärettömän pieni [13, s. 153–156]. Mallin geometriaa on havainnollistettu kuvassa 2.1.



Kuva 2.1 *Neulansilmäkameran geometria. Kaikki valonsäteet kulkevat kameran keskipisteen C kautta kuvatasolle, ja piste $\mathbf{p} \in \mathbb{R}^3$ projisoituu kameran kuvatasolle pisteeksi $\mathbf{q} \in \mathbb{R}^2$. Koska kuva piirtyy tasolle ylösalaisin, on akselit u ja v käännetty akseleihin x ja y nähden.*

Kuvaan 2.1 on sijoitettu kameran keskipiste C kolmiulotteisen koordinaatiston origoon. Kuvattava pisteestä $\mathbf{p} = (x_0, y_0, z_0)^T \in \mathbb{R}^3$ heijastuva valonsäde kulkee pisteen



(a) Tynnyrivääristymä (b) Tyynyvääristymä (c) Yhdistelmä tyyny- ja tynnyrivääristymiä (d) Tangentiaalinen vääristymä

Kuva 2.2 *Linssistön aiheuttamat vääristymät.*

C kautta kuvatasolle pisteeseen $\mathbf{q} = (u_0, v_0)^T \in \mathbb{R}^2$. Kuvataso $z = -f$ on x- ja y-akselien suuntainen ja etäisyydellä f origosta. Silloin kuvauksen $\mathbf{p} \mapsto \mathbf{q}$ voi laskea kaavalla [13, s. 154]

$$\begin{pmatrix} u \\ v \end{pmatrix} = \begin{pmatrix} f \cdot x/z \\ f \cdot y/z \end{pmatrix}, \tag{2.1}$$

josta havaitaan että kauempana kamerasta sijaitsevat kohteet (suuri z) piirtyvät kennolle pienempänä. Ilmiötä kutsutaan perspektiiviksi ja kuvausta $\mathbf{p} \mapsto \mathbf{q}$ kutsutaan myös perspektiiviseksi projektioksi.

Tämän työn kamerassa kuitenkin käytetään linssistöä, jonka vaikutuksia neulansilmäkameramalli ei huomioi. Tämän sovelluksen kannalta ainoat merkittävät vaikutukset ovat epätäydellisestä linssistöstä aiheutuvat radiaaliset ja tangentiaaliset vääristymät. Radiaalinen vääristymä syntyy valon taittuessa eri määrän linssin reunoilla keskialueeseen nähden. Se saa kuvan reuna-alueet piirtymään kuvatasolle väärään paikkaan ja näkyy kuvassa joko ns. tynnyrivääristymänä, tyynyvääristymänä tai niiden yhdistelmänä (kuva 2.2). Tangentiaalista vääristymää syntyy kun linssit eivät ole suorassa kuvakennoon nähden. Sen vaikutuksia on havainnollistettu kuvassa 2.2(d). [5, s. 375 – 376]

Radiaalinen vääristymä voidaan kompensoida kaavan

$$\mathbf{q}_k = \mathbf{q}(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) \tag{2.2}$$

mukaisesti, jossa $\mathbf{q} = (u, v)^T \in \mathbb{R}^2$ on kuvatasolle virheellisesti projisoituneen pisteen koordinaatit, $\mathbf{q}_k = (u_k, v_k)^T \in \mathbb{R}^2$ on pisteen korjatut koordinaatit ja r on pisteen $\mathbf{q}$ etäisyys vääristymän keskipisteestä, jonka oletetaan olevan koordinaateisa $u = v = 0$. Vakiot k_1, k_2 ja k_3 ovat radiaalisen vääristymän määrittävät parametrit.

Tangentiaalisen vääristymän korjaamiseen vastaava kaava on

$$\mathbf{q}_k = \mathbf{q} + \begin{pmatrix} 2p_1uv + p_2(r^2 + 2u^2) \\ 2p_2uv + p_1(r^2 + 2v^2) \end{pmatrix}, \tag{2.3}$$

jossa parametrit p_1 ja p_2 määrittävät tandentiaalisen vääristymän. Parametrit k_1, k_2, k_3, p_1 ja p_2 on kalibroitava jokaiselle kameran ja linssistön yhdistelmälle erikseen, mutta siihen ei perehdytä tässä työssä syvemmin. [5, s. 375 – 376]

Muunnos kolmiulotteisesta koordinaatistosta toiseen voidaan kuvata matriisilaskennan keinoin. Muunnos koostuu kolmesta komponentista: skaalauksesta $s \in \mathbb{R}$, rotaatiosta $\mathbf{R} \in \mathbb{R}^3$, ja translaatiosta $\mathbf{t} \in \mathbb{R}^3$. Näiden avulla piste $(x, y, z) \in \mathbb{R}^3$ kuvautuu pisteeksi $(u, v, w) \in \mathbb{R}^3$ kaavalla

$$\begin{pmatrix} u \\ v \\ w \end{pmatrix} = s\mathbf{R} \begin{pmatrix} x \\ y \\ z \end{pmatrix} + \mathbf{t}. \tag{2.4}$$

Tämä on erikoistapaus yleisemmästä muunnoksesta, jossa tulon $s\mathbf{R}$ tilalla voi olla mikä tahansa 3×3 matriisi. Rotaatiomatriisilla voidaan esittää mikä tahansa kolmiulotteinen rotaatio ja peräkkäisiä rotaatioita voidaan yhdistellä vapaasti matriisitulon avulla. Jokainen rotaatiomatriisi kuvaa kiertoa yhden akselin ympäri, ja mikä tahansa kolmiulotteinen rotaatio voidaan esittää kolmen ortogonaalisen akselin ympäri tapahtuvien kiertojen yhdistelmänä. Siten rotaatiot voidaan esittää esimerkiksi x-, y- ja z-akselien ympäri kiertävien matriisien tulona. Esimerkiksi kierto x-akselin ympäri kulmalla θ voidaan esittää matriisilla

$$\mathbf{R}_x(\theta) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{bmatrix}, \tag{2.5}$$

ja vastaavat matriisit voidaan muodostaa myös akseleille y ja z [13, s. 579]. Esimerkki kaavan 2.4 mukaisesta muunnoksesta on

$$\begin{pmatrix} u \\ v \\ w \end{pmatrix} = 1,5 \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\frac{\pi}{2} & -\sin\frac{\pi}{2} \\ 0 & \sin\frac{\pi}{2} & \cos\frac{\pi}{2} \end{bmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} + \begin{pmatrix} 1 \\ 2 \\ 0 \end{pmatrix}, \tag{2.6}$$

joka kiertää pistettä $(x, y, z)^T$ neljänneskierroksen x-akselin ympäri, kasvattaa sen etäisyyden 1,5-kertaiseksi ja siirtää sitä x-akselin suuntaisesti yhden askeleen ja y-akselin suuntaisesti kaksi askelta.

Kaavassa 2.4 translaatio on ainoa jota ei voi esittää matriisitulon avulla, mikä vaikeuttaa laskentaa joissain tapauksissa. Ongelma voidaan ratkaista muuttamalla koordinaatit homogeenisiksi, jolloin jokaiseen vektoriin lisätään neljäs komponentti. Piste $(x, y, z)^T$ on homogeenisessa muodossa $(x', y', z', w)^T$, kun $w \neq 0$. Muunnos koordinaattien välillä tapahtuu kaavalla

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} x'/w \\ y'/w \\ z'/w \end{pmatrix}. \tag{2.7}$$

Yleensä w:n arvoksi valitaan yksi, jolloin piste $(x, y, z)^T$ on homogeenisessa muodossa yksinkertaisesti $(x, y, z, 1)^T$. Tästä havaitaan, että koordinaatit $(x, y, z, w)^T$ ja $(kx, ky, kz, kw)^T$ kuvaavat samaa kolmiulotteisen avaruuden pistettä, kun $k \in \mathbb{R}$ ja $k \neq 0$ [13, s. 65]. Tätä homogeenisten koordinaattien ominaisuutta kuvataan tässä työssä merkinnällä

$$\begin{pmatrix} x \\ y \\ z \\ w \end{pmatrix} \sim \begin{pmatrix} kx \\ ky \\ kz \\ kw \end{pmatrix}. \tag{2.8}$$

Homogeenisilla koordinaateilla kaavan 2.4 muunnos $(x, y, z)^T \mapsto (u, v, w)^T$ voidaan esittää muodossa

$$\begin{pmatrix} u \\ v \\ w \\ 1 \end{pmatrix} = s \begin{bmatrix} \mathbf{R} & \mathbf{t} \\ \mathbf{0} & 1 \end{bmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}, \tag{2.9}$$

jota käytetään myös myöhemmin tässä työssä kuvaamaan kameran liikettä. Esimerkki 2.6 voidaan esittää homogeenisella koordinaatistolla muodossa

$$\begin{pmatrix} u \\ v \\ w \\ 1 \end{pmatrix} = 1,5 \begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & \cos\frac{\pi}{2} & -\sin\frac{\pi}{2} & 2 \\ 0 & \sin\frac{\pi}{2} & \cos\frac{\pi}{2} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}. \tag{2.10}$$

Kaava 2.9 kuvaa yleistä muunnosta kahden eri kolmiulotteisen koordinaatiston välillä, mutta kameran projektion kuvaamiseen tarvitaan muunnos kolmiulotteisesta kaksiulotteiseen. Homogeenisiä koordinaatteja käyttäen muunnos voidaan ilmaista

kaavalla

$$\begin{pmatrix} u \\ v \\ 1 \end{pmatrix} \sim \mathbf{K} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}, \tag{2.11}$$

jossa u ja v ovat kuvatason pisteen koordinaatit, x, y ja z ovat kolmiulotteisen avaruuden koordinaatit ja $\mathbf{K}$ on 4×3 matriisi. Matriisi $\mathbf{K}$ ei välttämättä ole muodoltaan sellainen, että tulon kolmas muuttuja saa suoraan arvon 1, jonka takia kaavassa on käytetty kaavan 2.8 mukaista merkintää. Kaavan 2.1 perspektiivisen projektion tapauksessa matriisi saa arvon

$$\mathbf{K} = \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}, \tag{2.12}$$

jolloin se on ideaalisen neulansilmäkameran *projektiomatriisi*. Kun matriisi sijoitetaan kaavaan 2.11 saadaan

$$\begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} f \cdot x \\ f \cdot y \\ z \end{pmatrix} \sim \begin{pmatrix} f \cdot x/z \\ f \cdot y/z \\ 1 \end{pmatrix} = \begin{pmatrix} u \\ v \\ 1 \end{pmatrix}, \tag{2.13}$$

joka on yhtenevä kaavan 2.1 kanssa. [13, s. 154]

Esitelty projektiomatriisi on ideaalinen, joten todellisten kameroiden kanssa matriisia on muokattava hieman. Kuvassa 2.1 on merkitty f etäisyydeksi kuvatasolta kameran keskipisteeseen, ja siten se vastaa kameran linssistön polttoväliä joka on sama x- ja y-suunnissa. Käytännön tilanteissa se kuitenkin sisältää myös muunnoskertoimen metreistä digitaalisen kuvakennon kuvapisteiksi, eikä sitä voi pelkän polttovälin perusteella määrittää. Esimerkiksi joissain tilanteissa kuvapisteet saattavat olla eri etäisyyksillä toisistaan x- ja y-suunnissa, jolloin myös f saa eri arvon suunnan mukaan. Lisäksi todellisissa kameroissa linssistö on harvoin keskitetty täydellisesti kuvakennoon nähden, eikä pisteet suoralla $x = y = 0$ kuvaudu kuvatason keskelle. Nämä muutokset huomioiden projektiomatriisi saa muodon

$$\mathbf{K} = \begin{bmatrix} f_x & 0 & c_x & 0 \\ 0 & f_y & c_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}, \tag{2.14}$$

jossa f_x ja f_y ovat x- ja y-akseleille erotetut f:n arvot ja muuttujat c_x ja c_y ovat kes-

kipisteen koordinaatit kuvapisteissä ilmoitettuna [13, s. 157]. Tätä kameran täydennettyä projektiomatriisia kutsutaan nimellä *kalibraatiomatriisi*. Tyypillisesti digitaalisessa kuvankäsittelyssä kuvan koordinaatiston origo sijoitettu kulmaan, ja muuttujien avulla c_x ja c_y saadaan myös se huomioitua.

Kuvassa 2.1 esitelty malli olettaa kameran keskipisteen C sijaitsevan koordinaatiston origossa, mutta tässä työssä kamera liikkuu suhteessa ympäristöönsä ja kameran sijainti ja asento halutaan määrittää ympäristön koordinaatistossa. Jos pisteen sijainti ympäristön koordinaatistossa on $\mathbf{r} = (x_0', y_0', z_0', 1)^T \in \mathbb{R}^4$, on sen sijainti kameran koordinaatistossa $\mathbf{p} = (x_0, y_0, z_0, 1)^T \in \mathbb{R}^4$ kaavan 2.9 mukaisesti

$$\mathbf{p} = \begin{bmatrix} \mathbf{R} & \mathbf{t} \\ \mathbf{0} & 1 \end{bmatrix} \mathbf{r}, \tag{2.15}$$

jossa $\mathbf{R}$ on kameran koordinaatiston rotaatio ympäristön koordinaatistoon nähden ja $\mathbf{t} = -\mathbf{Rc}$ on ympäristön koordinaatiston origon sijainti kameran koordinaatistossa. Vektori $\mathbf{c} \in \mathbb{R}^3$ on kameran keskipisteen sijainti ympäristön koordinaatistossa. Yhdistämällä tähän kaavaan kameran kalibraatiomatriisi 2.14, voidaan ympäristön pisteen $\mathbf{r} = (x_0', y_0', z_0', 1)^T \in \mathbb{R}^4$ projektion koordinaatit kuvatasolla laskea kaavalla

$$\mathbf{q} \sim \mathbf{K} \begin{bmatrix} \mathbf{R} & \mathbf{t} \\ \mathbf{0} & 1 \end{bmatrix} \mathbf{r}, \tag{2.16}$$

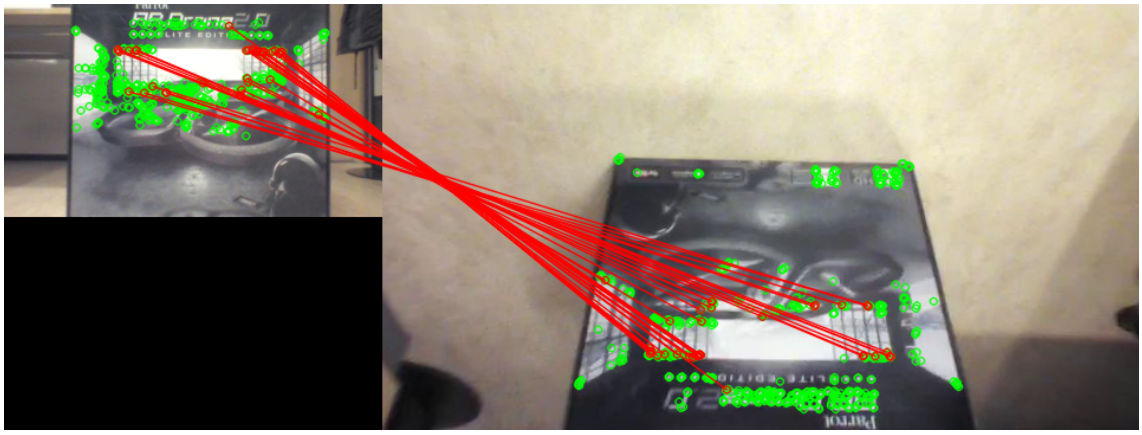
jossa $\mathbf{q} = (u, v, 1)^T \in \mathbb{R}^3$ on projisoituneen pisteen koordinaatit kuvatasnolla. [13, s. 155–157]

2.2 Piirteiden etsintä ja kuvaus ORB-algoritmilla

Tyypillisesti SLAM-algoritmeissa paikannus perustuu yksittäisten piirteiden sijainnin seuraamiseen, jotka valitaan dynaamisesti algoritmin suorituksen aikana (mapping). Tässä työssä ei perehdytä uusien piirteiden valintaan, vaan ainoastaan niiden seuraamiseen. Tämän työn yhteydessä toteutetussa algoritmissa seurattavat piirteet valitaan alustusvaiheessa. Käyttäjä kääntää kameran kohti haluamaansa näkymää, ja käskee ohjelmistoa alustamaan algoritmin seuraamaan näkymän piirteitä. Seurattavat piirteet etsitään koko kuvan alueelta ja niiden oletetaan olevan tasossa laskennan yksinkertaistamiseksi, joten seurattavaksi sopii esimerkiksi seinään kiinnitetty paperille tulostettu kuvio. Kuviolle ei ole asetettu muita vaatimuksia, ja käyttäjä voi valita vapaasti minkä tahansa tason mihin kameran voi osoittaa.

Kuvassa 2.3 on havainnollistettu kuvion seurantaa. Pienempi kuva vasemmalla on

seurattavaksi valittu kuvio, eli testilaitteistona toimivan helikopterin myyntipakkauksen kansi. Oikean puoleisessa kuvassa kamera on käännetty ylösalaisin ja siirretty kuvaamaan laatikkoa yläviistosta. Molemmista kuvista on etsitty vertailtaviksi sopivat piirteet, jotka on merkitty kuvaan vihreillä ympyröillä. Piirteet joille algoritmi on löytänyt vastaavuuden toisesta kuvasta, on merkitty punaisilla ympyröillä ja niiden välille on piirretty punainen viiva.



Kuva 2.3 *Löydetyt piirteet ja niiden vastaavuudet kuvien välillä. Vihreät ympyrät kuvaavat löydettyjen piirteiden keskipisteitä, ja punaiset viivat kuvien väliltä löytyneitä vastaavuuksia.*

Siihen, kuinka tarkasti valittua kuviota pystyy seuraamaan, vaikuttaa kuvion lisäksi sen piirteiden määritykseen käytetty algoritmi. Koska seurattavaa kuviota ei ole tarkemmin rajattu, sen voisi luoda mahdollisimman helposti seurattavaksi ja käyttää siihen optimoitua algoritmia kuten [14], jossa kuvio koostuu määrättyyn järjestykseen asetetuista harmaasävyltään eroavista neliöistä. Työn tavoitteena on kuitenkin pitää mahdollisimman paljon yhtymäkohtia piirrepohjaisiin SLAM-algoritmeihin ja siksi valittiin yleiskäyttöisempi lähestymistapa.

Tähän työhön valittu algoritmi on ORB, eli Oriented FAST and Rotated BRIEF [22]. Se yhdistää kaksi eri algoritmia: piirteiden etsintään tarkoitetun FAST:n [20] ja niiden kuvaukseen tarkoitetun BRIEF:n [6] sekä laajentaa ne toimimaan riippumatta rotaatiosta. Algoritmi on suunniteltu mahdollisimman nopeaksi reaaliaikakäyttöä varten, joka on suurin syy sen valintaan. Lisäksi sille löytyy valmis toteutus OpenCV-kirjastosta, eivätkä patentit rajoita sen käyttöä, kuten sen tunnetuimpia kilpailijoita SIFT (Scale-Invariant Feature Transform) [17] ja SURF (Speeded Up Robust Features) [4].

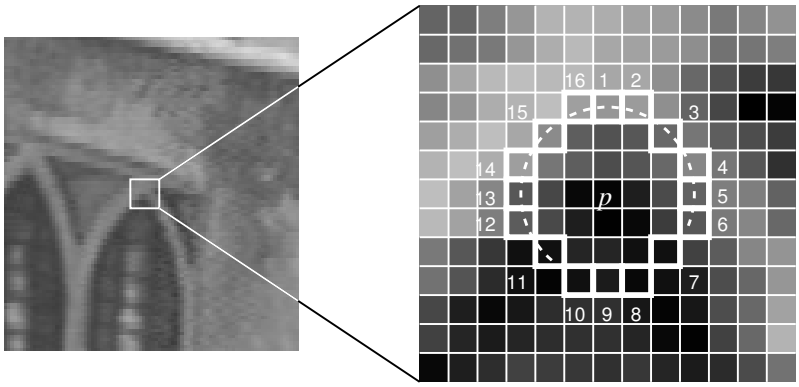
ORB-algoritmin käyttö jakaantuu kahteen erilliseen vaiheeseen: piirteiden etsimiseen ja piirteiden kuvaamiseen. Piirteen kuvaaminen (feature description) tarkoittaa nk. piirrevektorin laskemista, joka kuvaa numeerisesti piirteen yksilöiviä ominaisuuksia.

Piirrevektoreita vertaamalla voidaan määrittää kuvaavatko ne samaa kohdetta.

2.2.1 Piirteiden etsintä

ORB:ssa piirteiden etsintä pohjautuu FAST-algoritmiin (Features from Accelerated Segment Test), jota käytetään erityisesti sen laskennallisen tehokkuuden takia. Se on huomattavasti nopeampi [21] kuin muut suositut piirteenetsintämenetelmät kuten Harris [12] ja Difference of Gaussians, jota käytetään mm. SIFT-menetelmässä [17]. FAST soveltuu nopeutensa takia erityisen hyvin SLAM-algoritmeihin, kuten Kleinin ja Murrayn esittelemä PTAM eli Parallel Tracking and Mapping [15].

FAST käy kuvan jokaisen pisteen läpi ja määrittää onko kyseinen piste kulman keskikohta. Se tekee sen vertailemalla 16 ympäröivän pisteen kirkkauksia keskipisteen $\mathbf{p}$ kirkkauteen $I(\mathbf{p})$ kuvassa 2.4 havainnollistetulla tavalla. Piste määritetään kulmaksi, jos ympäröivistä pisteistä vähintään n vierekkäistä pistettä ovat kaikki kirkkaampia kuin $I(\mathbf{p}) + t$ tai kaikki tummempia kuin $I(\mathbf{p}) - t$. Parametri t on algoritmin käyttäjän määrittämä kynnysarvo kirkkauserolle. FAST-algoritmin alkuperäisessä versiossa Rosten ja Drummond [20] valitsivat pisteiden lukumääräksi $n = 12$, mutta osoittivat myöhemmin [21] tulosten paranevan arvolla $n = 9$, jota käytetään myös ORB-algoritmin tapauksessa [22].



Kuva 2.4 *FAST-algoritmi määrittää onko piste p kulman keskipiste vertaamalla sen kirkkautta ympäröivien pisteiden 1–16 kirkkauksiin. Kuvassa p on määritetty kulmaksi, sillä vähintään 12 vierekkäistä pistettä (11–16 ja 1–6) ovat kirkkaampia kuin p. Kuvan lähde on [20].*

Koska FAST ei kerro pisteestä muuta tietoa kuin onko kyseessä kulma vai ei, löydettyjä kulmia ei voi laittaa minkäänlaiseen paremmuusjärjestykseen. Rublee *et al.* havaitsivat [22], että FAST tulkitsee kulmiksi usein myös reunoja, joka sopivat huonommin seurantaan. Suodattaakseen pois reunat he käyttävät ORB:ssa Harrisin menetelmää [12] laskeakseen jokaiselle löydetylle FAST-piirteelle kulmapistearvon,

jonka avulla voidaan erottaa kulmat reunoista. Löydetyt FAST-piirteet asetetaan järjestykseen kulmapistearvon perusteella, ja niistä valitaan seurantaan k eniten kulmaa muistuttavaa piirrettä. ORB:ssa asetetaan FAST:n kynnysarvo t riittävän matalaksi, jotta kuvasta löydetään yli k piirrettä karsintaa varten.

Toinen FAST:n soveltamista vaikeuttava ominaisuus on voimakas riippuvuus kuvan skaalasta, sillä kulman tunnistaminen sidottu kiinteästi kuvapisteisiin. Esimerkiksi hieman pyöristetty kulma voi läheltä kuvattuna vaikuttaa algoritmin kannalta reunalta. Tyypillinen ratkaisu on ns. pyramidimenetelmä, jossa kuvaa pienennetään asteittain n kertaa käyttäen skaalauskerrointa s peräkkäisten tasojen välillä, ja jokaiselta tasolta etsitään piirteet erikseen. Tätä menetelmää käytetään FAST:n apuna ORB:n lisäksi myös PTAM:ssa [15]. ORB:ssa myös eri pyramiditasoilla löydetyt FAST-piirteet pisteytetään ja karsitaan Harrisin menetelmällä [22].

ORB:n kolmas lisäys puhtaaseen FAST-algoritmiin on kulman orientaation, eli sen osoittaman suunnan määritys. Se ei kuitenkaan vaikuta itse piirteiden etsimiseen millään tavalla, vaan sitä hyödynnetään vasta piirrevektorin laskennassa. ORB käyttää Rosinin [19] esittelemää ”intensity centroid” -menetelmää orientaation laskentaan, johon ei perehdytä tässä työssä syvemmin. Määritetty orientaatio ilmaistaan kulmalla θ, joka on kierto suhteessa kuvan koordinaatistoon.

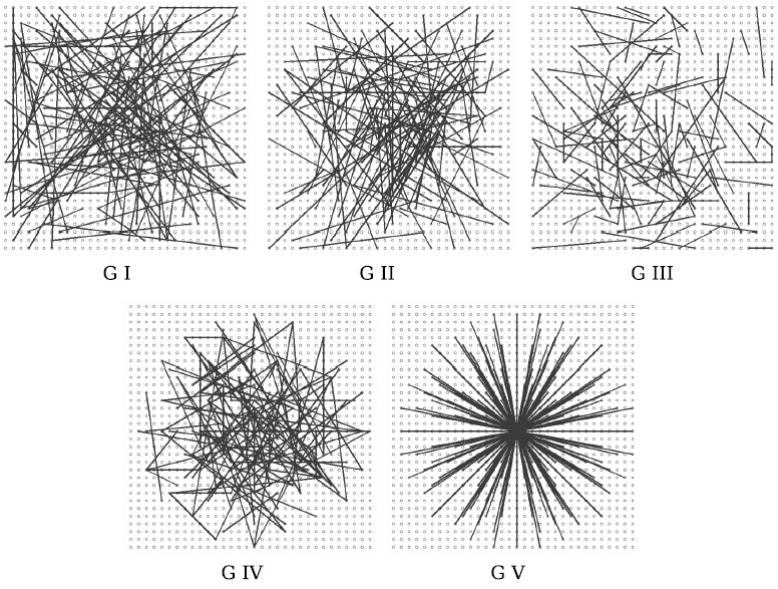
2.2.2 Piirteiden kuvaus

Kun FAST-algoritmilla on löydetty kuvasta seurattavaksi sopivat kohdat, lasketaan jokaiselle niistä yksilöllinen piirrevektori. Piirrevektorin tulee kuvata ympäröivää aluetta riittävällä tarkkuudella, että vertailukuvasta voidaan löytää vastaava kohta. ORB-algoritmissa piirrevektorien laskenta perustuu BRIEF-algoritmiin [6], jossa vertaillaan ympäristön kuvapisteiden harmaasävyjä pareittain. Oletetaan, että on n kappaletta pistepareja $\{(\mathbf{p}_1, \mathbf{q}_1), (\mathbf{p}_2, \mathbf{q}_2), \ldots (\mathbf{p}_n, \mathbf{q}_n)\}$, jossa $\mathbf{p}_i, \mathbf{q}_i \in \mathbb{R}^2$ ovat ympäristön kuvapisteiden koordinaatteja. Tällöin BRIEF-piirre on binäärivektori $\mathbf{b} = (b_1, b_2, \ldots b_n)$, missä

$$b_i = \begin{cases} 1, & \text{jos } I(\mathbf{p}_i) > I(\mathbf{q}_i) \\ 0, & \text{muulloin} \end{cases}, \tag{2.17}$$

kun $I(\mathbf{p})$ on kuvan harmaasävyarvo pisteessä $\mathbf{p}$. Pisteparit $(\mathbf{p}_i, \mathbf{q}_i)$ voidaan valita mielivaltaisesti ja kuvassa 2.5 on havainnollistettu muutamia vaihtoehtoja. Kuvassa viivojen päät kuvaavat vertailtavia pisteitä.

Kuvassa 2.5 kaikissa paitsi viimeisessä vaihtoehdossa on pisteparit arvottu satun-



Kuva 2.5 *Esimerkkejä mahdollisista BRIEF-piirteen laskentakuvioista [6].*

naisesti käyttäen erilaisia jakaumia. Calonder *et al.* havaitsivat [6], että kokeillut satunnaisjakaumat tuottivat lähes yhtä hyviä tuloksia, mutta viimeinen järjestyksellinen vaihtoehto suoriutui selvästi muita huonommin.

Koska piirrevektorin laskentaan käytetään yksittäisten pikselien kirkkauksia, voi niiden satunnaisvaihtelu vaikuttaa suuresti lopputulokseen. Ongelma on ratkaistu BRIEF:ssä sumentamalla kuvaa ennen piirrevektorin laskentaa. Se auttaa myös tilanteissa joissa piirteessä on teräviä reunoja, parantaen piirrevektorin vakautta ja toistettavuutta vaikeissa vertailutilanteissa [6].

BRIEF-piirre on voimakkaasti riippuvainen kuvan rotaatiosta [22], ja sellaisenaan se soveltuisi huonosti tähän työhön. ORB:ssa ongelma on ratkaistu määrittämällä piirteen orientaatio ennen piirrevektorin laskentaa osion 2.2.1 lopussa kuvatulla tavalla. Pisteparien koordinaatteja käännetään orientaatiota kuvaavan kulman θ verran, ja saatuja uusia koordinaatteja käytetään piirrevektorin laskentaan. Mikäli orientaatiot ovat konsistentteja eri kuvien välillä, on laskentakuvio aina samassa asennossa piirteeseen nähden. Laskennan nopeuttamiseksi ORB:ssa kulma θ on jaettu 12 asteen välein ja kaikille kulmille lasketaan valmiiksi pisteparien koordinaatit ja tallennetaan ne taulukkoon.

Rublee *et al.* havaitsivat [22], että BRIEF:n kyky erottaa piirteet toisistaan on riippuvainen piirteiden satunnaisesta orientaatiosta. Koska ORB:ssa piirteiksi valitaan yleensä kulmia ja niiden orientaatiot normalisoidaan ennen piirrevektorin laskentaa, eri piirteiden vektorit muistuttavat huomattavasti enemmän toisiaan ja varians-

si piirteiden välillä laskee verrattuna muokkaamattomaan BRIEF:iin. Tämä tekee piirteiden luokittelusta ja vastaavuuksien löytämisestä huomattavasti vaikeampaa. Rublee *et al.* esittelivät [22] menetelmän laskentakuvion määrittämiseen, joka toimii paremmin orientaatioltaan normalisoidun BRIEF:n kanssa.

Kuten piirteiden etsiminen FAST:llä, myös piirrevektori on voimakkaasti riippuvainen kuvan skaalauksesta. Pelkkä BRIEF ei tarjoa siihen mitään ratkaisua, joten ORB hyödyntää samaa osiossa 2.2.1 esiteltyä pyramidi-menetelmää kuin piirteiden etsinnässä [22]. Eri skaalaustasoilta löydetyille piirteille lasketaan piirrevektorit erikseen, ja mahdollistetaan siten piirteiden tunnistus esim. kameran liikkuessa kauemmaksi kuvattavasta kohteesta.

2.3 Piirteiden sovitus

Jotta voitaisiin määrittää kuvaavatko kaksi eri piirrevektoria samaa kohdetta, on oltava jokin mitta joka kertoo niiden samankaltaisuuden. ORB:n käyttämät BRIEF-piirteet ovat binäärilukuja, joiden samankaltaisuus voidaan laskea Hamming-etäisyydellä [6]. Hamming-etäisyys määritetään vertaamalla kahta binäärilukua toisiinsa ja laskemalla kaikkien eroavien bittien yhteismäärä. Laskenta onnistuu tietokoneella erittäin tehokkaasti suorittamalla luvuille XOR-operaatio bittitasolla, ja laskemalla lopputuloksesta 1-bittien lukumäärä [18].

Jotta voitaisiin määrittää piirteille vastaavuudet kahden eri kuvan välillä, on jokaiselle piirteelle löydettävä eniten sitä muistuttava pari verrokkikuvasta. Kyseessä on siis lähimmän naapurin haku (NNS, nearest neighbour search), josta tässä käytetään lyhennettä NN-haku. Yksinkertaisin NN-hakumenetelmä on niin kutsuttu brute-force -menetelmä. Siinä kuvan A piirrevektoria a_n verrataan jokaiseen kuvan B piirrevektoriin b_n ja valitaan pariksi pienimmän Hamming-etäisyyden tuottava piirrevektori.

Brute-force -menetelmä on kuitenkin hyvin hidas suurilla piirremäärillä. Sen vuoksi on kehitetty NN-hakumenetelmiä, jotka käyvät läpi vain osan mahdollisista vaihtoehdoista ja valitsevat approksimaation lähimmästä naapurista. Tällöin valittu naapuri ei välttämättä ole lähin mahdollinen, mutta haku voi olla useita kertaluokkia nopeampi kuin brute-force -menetelmä [18]. Sovelluksesta riippuen ero tarkkuudessa voi olla merkityksetön saatavaan nopeushyötyyn verrattuna. Approksimoivia NN-hakumenetelmiä on monia, mutta useat yleisesti käytetyt menetelmät eivät toimi sellaisenaan binääripiirteiden kanssa [18]. Binääripiirteiden kanssa toimivia menetelmiä ovat esim. LSH (Locality Sensitivity Hashing) [11], jota Rublee *et al.* käyttivät [22] vertaillessaan ORB:ta aiempiin menetelmiin, sekä Mujan ja Lowen esittelemä

Hierarchical Clustering [18].

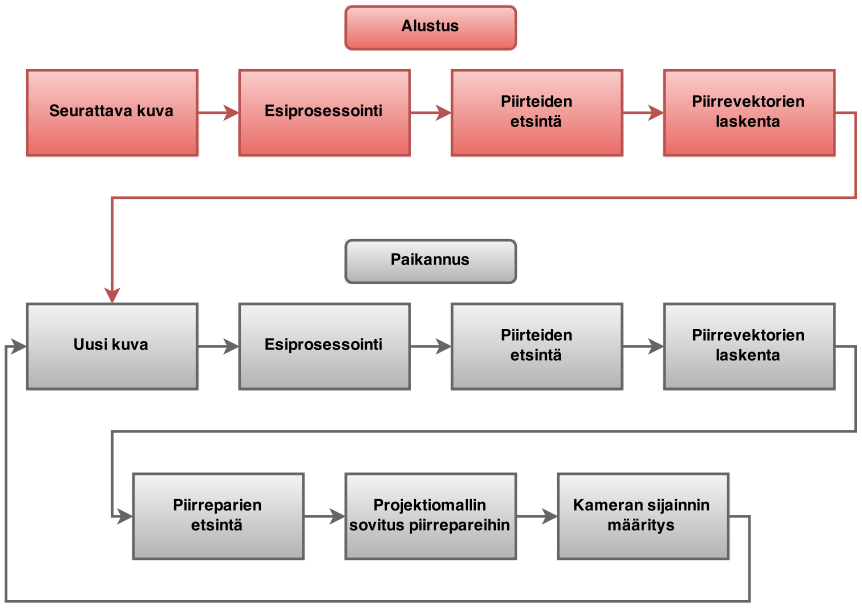
Vastaavuudet piirteiden välillä eivät vielä riitä ratkaisemaan kameran sijainnin muutosta kuvien välillä, vaan siihen on niihin on sovitettava matemaattinen malli, joka kuvaa miten piirteen sijainti kameran ympäristön koordinaatistossa projisoituu kuvatason koordinaatistoon. Malli on määritetty kaavassa 2.16, josta pitää ratkaista kameran rotaatiomatriisi $\mathbf{R}$ ja ympäristön koordinaatiston origon sijainti kameran koordinaatistossa $\mathbf{t} = -\mathbf{Rc}$. Piirteen sijainti ympäristön koordinaatistossa $\mathbf{r}$ on määritetty alustusvaiheessa, ja piirteen projektion sijainti $\mathbf{q}$ kuvatasolla on piirteelle löydetyn vastaavuuden koordinaatit. Kameran kalibraatiomatriisi $\mathbf{K}$ on määritetty samalla kalibrointiprosessilla kuin kameran linssivääristymien parametrit.

Kun kameran kalibrointiparametrit ovat tunnettuja, voidaan liikeparametrit ratkaista kolmen piirreparin perusteella [13, s. 116]. Käytännön tilanteissa piirteiden kuvasta määritetty sijainti sisältää liian paljon virhettä, jonka vaikutusta voidaan vähentää käyttämällä useampaa pisteparia. Sellaisessa tilanteessa parametrit voidaan selvittää epälineaarisella pienimmän neliösumman menetelmällä, joka pyrkii iteratiivisesti löytämään virheiden neliösumman minimoivan mallin pisteparien välille. Tässä työssä hyödynnetään Levenberg-Marquardt-menetelmää [13, s. 600], johon ei perehdytä tämän tarkemmin. Menetelmä toimii optimaalisesti vain, kun kaikki virhe piirreparien koordinaateissa on normaalijakautunutta, eli peräisin mittausvirheestä piirteen sijainnin määrityksessä [13, s. 116].

Piirteet voidaan kuitenkin tulkita virheellisesti vastaaviksi vaikka niiden sijainti eroaisi merkittävästi, jolloin virhe ei noudata enää normaalijakaumaa. Tälläiset poikkeavat piirreparit on suodatettava pois paremman estimaatin saavuttamiseksi. Se onnistuu esimerkiksi Random Sample Consensus (RANSAC) menetelmällä. Se toimii yksinkertaistettuna seuraavasti: Kaikkien piirreparien joukosta S valitaan satunnaisesti otos, jossa on vähimmäismäärä piirrepareja sovitettavan mallin määrittämiseen. Sen jälkeen määritetään malli otoksen pohjalta ja lasketaan kaikkien piirreparien virhe määritettyyn malliin verrattuna. Mikäli virhe on alle kynnysarvon t, piirrepari kuuluu joukkoon S_i ja kaikki muut piirreparit tulkitaan poikkeamiksi. Sen jälkeen valitaan uusi satunnainen otos ja toistetaan aiemmat vaiheet. Tätä toistetaan N kertaa ja valitaan kooltaan suurin osajoukko S_i ja käytetään sitä mallin lopulliseen sovitukseen esim. Levenberg-Marquardt-menetelmällä. Suoritus voidaan lopettaa myös aimmin, kun löydetään joukko S_i jonka koko ylittää määritetyn kynnysarvon. [13, s. 117–118]

3. TOTEUTUS JA TULOKSET

Lopullisen toteutuksen toimintaa havainnollistaa kuva 3.1, jossa on jokainen vaihe eriteltynä ja niiden välinen suoritusjärjestys. Kuvaan on merkitty punaisella alustusvaihe, jossa valitaan tuorein kameralta saapuva kuva seurattavaksi kuvioksi ja määritetään kameran sen hetkinen sijainti ympäristön koordinaatiston origoksi. Alustus suoritetaan vain kerran käyttäjän määräämänä hetkenä, jonka jälkeen siirrytään varsinaiseen paikannukseen, joka merkitty kuvaan harmaalla. Paikannus suoritetaan jokaiselle kameralta saapuvalle kuvalle kunnes suoritus keskeytetään tai algoritmi alustetaan uudelleen. Paikannusvaiheessa käsiteltävää kuvaa verrataan alustuksessa valittuun kuvaan ja määritetään kameran liike kuvien välillä, ja siten kameran sen hetkinen sijainti ja asento. Vaiheiden tarkempi sisältö ja toteutus on kuvattu osiossa 3.2.



Kuva 3.1 *Algoritmin toiminta kuvattuna vaiheittain.*

3.1 Laitteisto ja ohjelmistoalusta

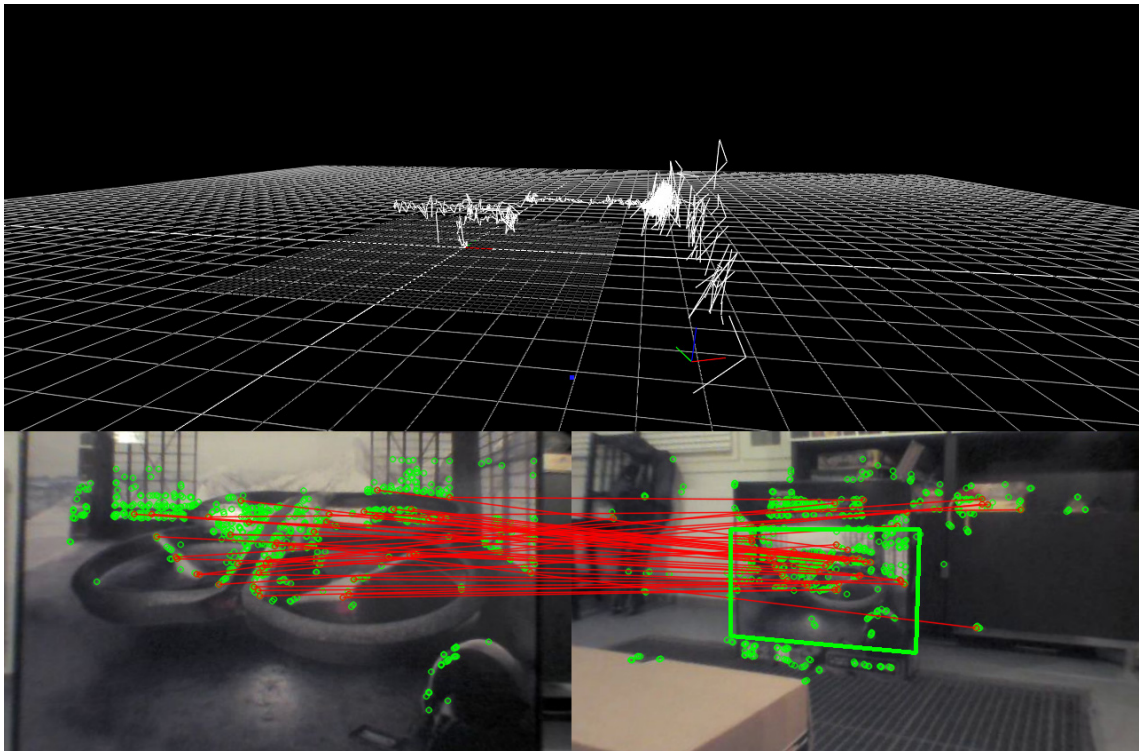
Toteutuksessa käytetty laitteisto koostuu kannettavasta tietokoneesta ja langattoman lähiverkon kautta siihen yhteydessä olevasta kamerallisesta Parrot AR.Drone 2.0 -helikopterista (kuva 3.2). Kannettavassa tietokoneessa käytetään Xubuntu 14.04 Linux-käyttöjärjestelmää ja siihen asennettua Robot Operating System (ROS) -ohjelmistoa [3]. ROS on avoimella lähdekoodilla tehty kokoelma kirjastoja, ohjelmamoduuleja ja työkaluja erilaisia robotiikan sovelluksia varten. ROS:n osana toimivia ohjelmamoduuleja voi tehdä Python- ja C++-ohjelmointikielillä ja se tarjoaa moduulien välille yhdenmukaisen rajapinnan viestinvälitysmekanismimeillaan. Toteutuksessa ROS toimii rajapintana paikannusalgoritmin ja paikannettavan helikopterin välillä.

Kuva 3.2 *Parrot AR.Drone 2.0 -helikopteri.*

Helikopteri on varustettu kahdella kameralla, joista toinen on suunnattu eteenpäin ja toinen alaspäin, sekä kiihtyvyysantureilla ja gyroskoopilla. Kopterin valmistaja on julkaissut ohjelmointirajapinnan, jonka kautta kopteria pystyy ohjaamaan ja sen lähettämää dataa voi lukea langattoman verkon yli. Tässä työssä hyödynnetään sen eteenpäin suunnattua kameraa, joka kykenee $1280 \cdot 720$ pikselin resoluutioon ja 30Hz kuvataajuuteen [1]. Valmistajan tarjoamaa ohjelmointirajapintaa ei hyödynnetä tässä työssä suoraan, vaan sitä käytetään ROS ohjelmistoon kuuluvassa *ardrone_autonomy*-moduulissa [3], joka muuttaa kopterin ohjauskomennot, sensoridatan ja videokuvan muiden ROS-moduulien kanssa yhteensopivaan muotoon.

Tämän työn algoritmi on toteutettu C++-kielellä osana ROS-moduulia[1], joka mah-

[1]Moduulin lähdekoodi on ladattavissa osoitteessa `https://github.com/ansse/flosdrone`.



Kuva 3.3 *Ohjelmiston käyttöliittymä: ylhäällä kuvassa on reitin kolmiulotteinen visualisaatio, alareunassa vasemmalla on seurattava kuva ja oikealla alhaalla on reaaliaikainen kuva kopterin kameralta.*

dollistaa paikannuksen lisäksi myös kopterin ohjaamisen tietokoneeseen kytketyllä peliohjaimella. Moduulin käyttöliittymä on toteutettu OpenGL-grafiikkarajapintaa hyödyntäen, jonka avulla voidaan visualisoida arvioitu reitti kolmiulotteisesti reaaliajassa (kuva 3.3). Varsinainen algoritmi on toteutettu hydyntäen OpenCV-konenäkökirjastoa.

Kopterin kameran kalibroinnissa on hyödynnetty valmista ROS-ohjelmiston moduulia `camera_calibration`, joka käyttää kalibrointiin OpenCV-kirjaston funktioita. Sen avulla on saatu selville tässä käytetyn kopterin kameran linssivääristymien ja kalibraatiomatriisin parametrit.

3.2 Algoritmin toteutus OpenCV-konenäkökirjastolla

OpenCV (Open Source Computer Vision Library) [2] on avoimen lähdekoodin kirjasto konenäön ja koneoppimisen sovellutuksiin. Se tarjoaa valmiit toteutukset kappaleessa 2 kuvatuille algoritmeille ja muita työkaluja helpottamaan kuvien käsittelyä. OpenCV on painottunut reaaliaikaisiin konenäkösovelluksiin ja sille on tarjolla rajapinnat C++-, C-, Python- ja MATLAB-ohjelmointikielille.

ROS-ohjelmistossa videokuvan välitys tapahtuu yhden kuvan sisältävinä viesteinä moduulilta toiselle, ja uuden viestin saapuminen aiheuttaa kuvaa vastaanottavassa moduulissa funktiokutsun. Algoritmin toiminta on synkronoitu näihin viesteihin. Ohjelman käynnistyksen yhteydessä vastaanotetaan ensimmäisenä viesti joka sisältää kuvan lisäksi myös kameran kalibraatioparametrit. Niiden pohjalta luodaan OpenCV:n funktiolla `cv::initUndistortRectifyMap` matriisi, jolla kuvista saadaan poistettua linssivääristymät funktiolla `cv::remap` [2]. Sen jälkeen siirrytään odottamaan kunnes käyttäjä käynnistää algoritmin alustuksen.

Alustuksessa uusin kameralta saapunut kuva valitaan seurattavaksi ja siirrytään sen esiprosessointiin. Esiprossointivaiheessa kuva muutetaan harmaasävyiseksi, sillä ORB-algoritmi perustuu kuvapisteiden kirkkauksien vertailuun. Lisäksi kuvasta poistetaan linssinvääristymät, jotta ne eivät aiheuttaisi virhettä löydettyjen piirteiden sijainnin määrityksessä.

Esiprosessoinnin jälkeen aloitetaan piirteiden etsintä ja piirrevektorien laskenta. OpenCV:ssä ORB-algoritmin käyttö tapahtuu `cv::ORB`-tyyppisen olion kautta [2]. Oliolle annetaan rakentajassa parametrit, jotka määrittävät sekä piirteiden etsinnän että piirrevektorien laskennan toiminnan. Parametrit on kuvattu taulukossa 3.1.

Taulukko 3.1 *OpenCV:n ORB-luokan rakentajan parametrit.*

Parametri (Oletusarvo)	**Kuvaus**
`nfeatures` (500)	Kuvasta valittavien piirteiden maksimimäärä.
`scaleFactor` (1.2)	Pyramiditasojen välinen skaalauskerroin s.
`nlevels` (8)	Pyramiditasojen määrä määrä n.
`edgeThreshold` (31)	Kuvan reunalueen leveys kuvapisteina, josta piirteitä ei etsitä. Tulisi vastata suunnilleen `patchSize`-parametrin arvoa.
`firstLevel` (0)	Ensimmäinen pyramiditaso, josta piirteitä etsitään.
`WTA_K` (2)	Yhden BRIEF-piirrevektorin elementin laskennassa vertailtavien kuvapisteiden määrä. Oletusarvo 2 vastaa työssä kuvattua toimintaa. Muut vaihtoehdot (3 ja 4) tuottavat 2 bittiä yhtä elementtiä kohden.
`scoreType` (HARRIS_SCORE)	Läydettyjen piirteiden kulmapistearvon käytettävä menetelmä. Oletusarvoisen Harris-menetelmän vaihtoehtona nopeampi, mutta epävakaampi FAST_SCORE.
`patchSize` (31)	BRIEF-piirrevektorin laskentaan käytettävän kuva-alueen koko.

Parametrin `nfeatures` arvoksi valittiin 1000, `scaleFactor` asetettiin Calonder *et al.* [6] käyttämään arvoon 1.412 ja muut parametrit jätettiin oletusarvoihinsa. Arvot todettiin toimiviksi kokeellisesti, eikä niiden vaikutusta lopputulokseen tutkittu

tarkemmin. Kun olio on luotu ja alustettu, piirteiden etsintä suoritetaan sen metodilla `detect`, joka määrittää 1000 parhaiten seurattavaksi sopivan piirteen sijainnin ja asennon. Piirrevektorien laskenta löydetyille piirteille tapahtuu saman olion metodilla `compute`.

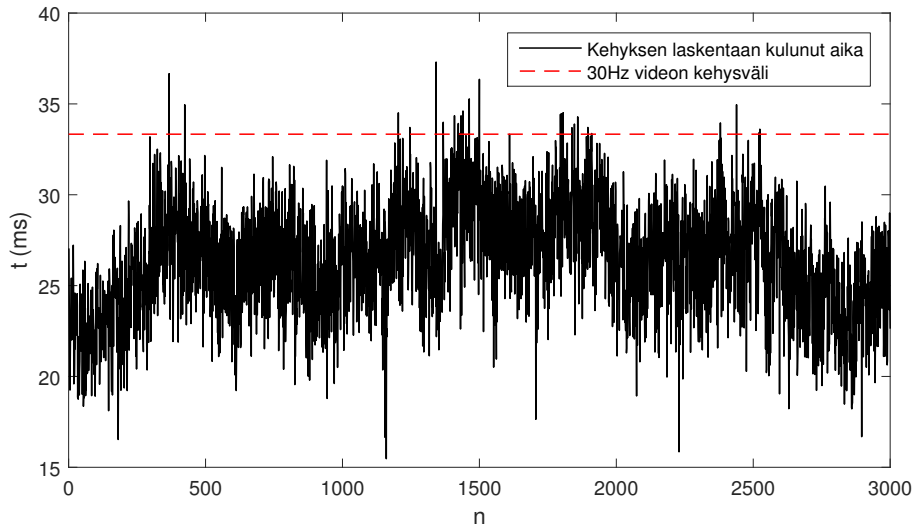
Alustusvaiheen jälkeen siirrytään odottamaan seuraavan kuvan saapumista ja varsinaisen paikannusprosessin alkua. Paikannuksessa suoritettavat vaiheet esiprosessoinnista piirrevektorien laskentaan ovat identtiset alustukseen verrattuna. Kun uusimmalle kuvakehykselle on saatu laskettua piirrevektorit, verrataan niitä seurattavan kuvan piirteisiin. Tässä työssä päädyttiin käyttämään brute-force -menetelmää, sillä se todettiin laskennallisesti riittävän nopeaksi eikä approksimoivia hakumenetelmiä siksi kokeiltu. Se toteutettiin OpenCV:n `BFMatcher`-luokalla [2], joka laskee piirteiden väliset Hamming-etäisyydet ja valitsee pareiksi toisiaan lähinnä olevat piirteet.

Löydetyt piirreparit välitetään funktiolle `solvePnPRansac` [2], joka ottaa parametrinaan myös kameran kalibrointiparametrit ja RANSAC:n toiminnan määrittelevät parametrit. Se laskee seurattavan kohteen sijainnin ja asennon suhteessa kameraan, josta voidaan helposti laskea kameran sijainti ja asento ympäristöön nähden.

3.3 Tulokset

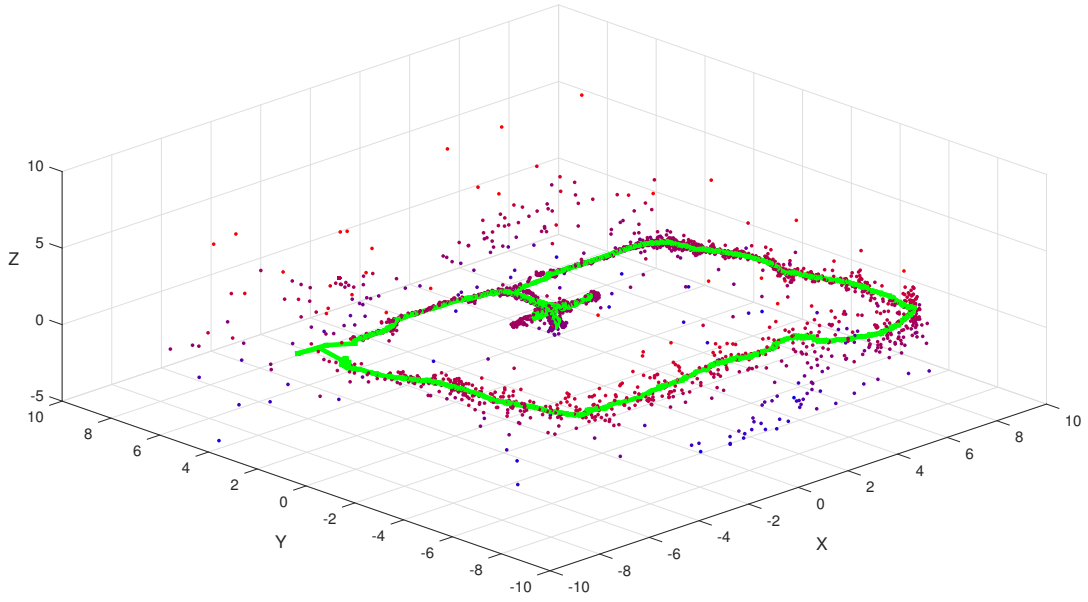
Toteutuksen suoritusnopeutta tutkittiin mittaamalla yhdelle kuvakehykselle suoritettavaan laskentaa kulunut aika ohjelmallisesti. Aika laskettiin 3000 kuvakehykselle ja tulokset on piirretty kuvaajaan 3.4, jossa y-akselilla on aika ja x-akselilla on kuvakehyksen järjestysnumero n. Mustalla viivalla on merkitty yksittäisen kuvakehyksen laskentaan kulunut aika. Punaisella katkoviivalla on merkitty 30Hz-videon kuvakehyksien välinen aika, joka on yläraja kun tavoitteena on reaaliaikainen suoritus. Kuvaajasta nähdään, että laskenta-aika pysyy pääosin selvästi rajan alapuolella. Ohjelmistoa suoritettiin tietokoneella, jossa oli 3.6GHz kellotaajudella toimiva Intel Core i7-3720QM neliydinprosessori.

Algoritmin toimintaa testattaessa ei ollu mahdollista mitata kamerakopterin tarkkaa asentoa ja sijaintia, eikä siten algoritmin absoluuttista tarkkuutta pystytty määrittämään. Jonkinlaisen vertailukohdan saavuttamiseksi algoritmia testattiin liikuttelemalla käsin kopteria lattiaan merkityn neliskanttisen radan mukaisesti ja pyrittiin pitämään se suunnilleen vakiokorkeudella koko matkan ajan. Seurattavana kohteena toimi kopterin myyntipakkaus. Tällä reitillä algoritmin määrittämät sijainnit yksittäisillä kuvakehyksilla on merkitty kuvaan 3.5 kolmiulotteiseen koordinaatistoon. Pisteen väri määräytyy sen sijainnista z-akselilla syvyysvaikutelman tehostamiseksi. Kirkkaan sininen väri tarkoittaa z-akselin arvoa -5 ja kirkkaan punainen väri arvoa



Kuva 3.4 *Yhden kuvakehyksen laskentaan kulunut aika mitattuna 3000 kuvakehykselle. Punaisella katkoviivalla on merkitty suurin sallittu aika reaaliaikaisessa toiminnassa 30Hz videolla.*

+5, ja niiden välillä skaalautuu lineaarisesti suhteessa sijaintiin. Vihreällä viivalla on merkitty pisteistä jälkikäteen MATLAB-ohjelmistolla laskettu mediaanisuodatukseen pohjautuva arvio kuljetusta reitistä. Työssä kuvattu algoritmi ei itse suodata sijaintiarviota millään tavalla, ja suodatettu arvio on lisätty kuvaajaan vain reitin havainnollistamiseksi.



Kuva 3.5 *Algoritmin arvio kopterin sijainnista, kun sitä liikultetiin suunnilleen vakiokorkeudella neliskanttista reittiä pitkin. Pisteillä on merkitty algoritmin laskemat arvot yksittäisille kuville ja pisteen väri riippuu sen sijainnista Z-akselilla. Vihreällä viivalla on merkitty pisteiden pohjalta mediaanisuodatettu arvio reitistä.*

Kamerakopteri lähti liikkeelle origosta kameran osoittaessa positiiviseen y-suuntaan, josta sitä nostettiin ylöspäin ja liikuteltiin ensin sivuttaissuunnassa (x-akselin suuntaisesti edestakaisin. Sen jälkeen sitä siirrettiin hieman eteenpäin ja aloitettiin liikuttelu neliskanttisella radalla ylhäältä katsoen myötäpäivään. Seurattava kohde sijaitsee x-akselilla positiivisessa Y-suunnassa, ja mitä kauempana kopteri on kohteesta, sitä suurempaa hajontaa on havaittavissa sijaintiarvioissa. Suurilla etäisyyksillä kameran rajallinen resoluutio tekee piirteiden sijainnin määrityksestä epätarkempaa ja lisäksi kuvasta löytyy suurempi määrä piirteitä jotka eivät kuulu seurattavaan kohteeseen. Vihreällä viivalla merkittyä suodatetusta tuloksesta havaitaan kuitenkin mittaustulosten seuraavan keskimäärin neliskanttista reittiä.

4. JOHTOPÄÄTÖKSET

Työn tavoitteena oli perehtyä kamerapohjaisen paikannuksen taustalla olevaan teoriaan ja toteuttaa kameran sijainnin määrittävä algoritmi. Toteutettu algoritmi arvioi kameran sijaintia ja asentoa suhteessa ympäristöönsä etsimällä videokuvasta tunnettua kohdetta. Etsittävä kohde valitaan alustusvaiheessa, jossa kamera suunnataan seurattavaksi valittuun tasoon ja tallennetaan kameran sen hetkinen näkymä seurantaa varten. Videokuvan jokaiselle kuvakehykselle määritetään kameran sijainti ja asento alustusvaiheeseen verrattuna, etsimällä siitä seurattavaksi valittua kuvaa. Sekä seurattavasta kuvasta, että paikannuksessa olevasta kuvakehyksestä etsitään seurantaan sopivia piirteitä ja lasketaan niille niiden ulkonäköä kuvaava piirrevektori. Piirrevektoreita vertaamalla pyritään löytämään seurattava taso paikannettavasta kuvasta jonka sijainnista määritetään kameran liike kuvien välillä.

Toteutuksen tavoitteena oli reaaliaikainen suoritus 30Hz videolla, joka saavutettiin käytössä olleella laitteistolla. Algoritmin eri parametrien vaikutusta suoritusnopeuteen ei työssä tutkittu, ja niitä optimoimalla voisi saavuttaa lisänopeutusta ilman merkittävää heikennystä algoritmin tarkkuuteen. Myös algoritmissa käytetyn piirreparien etsintään käytetyn brute-force -menetelmän korvaaminen jollain approksimoivalla nn-hakumenetelmällä todennäköisesti nopeuttaisi algoritmin toimintaa.

Algoritmin paikannustarkkuutta ei pystytty absoluuttisesti mittaamaan, mutta se todettiin empiirisesti vastaavan alkuperäisiä oletuksia ja tavoitteita. Lähellä seurattavaa kohdetta algoritmi antoi suhteellisen tarkkoja ja toistettavissa olevia sijaintiarvioita, mutta erityisesti suurilla etäisyyksillä seurattavasta kohteesta sijaintiarviossa syntyi hyvin merkittäviä poikkeamia todellisesta sijainnista. Suurimmat virheet syntyivät tilanteissa jossa algoritmi määritti sijainnin virheellisesti valittujen piirreparien perusteella. Toteutuksessa käytetyn RANSAC-algoritmin parametrejä optimoimalla virhettä pystyisi todennäköisesti pienentämään.

Algoritmi määrittää jokaiselle kuvakehykselle kameran sijainnin ja asennon itsenäisesti, ilman että edellisille kuvakehyksille laskettuja arvioita huomioitiin miten-kään. Tuloksia analysoidessa huomattiin, että yksinkertaisella mediaanisuodatukseen perustuvalla menetelmällä saatiin sijaintiarvioiden satunnaisvaihtelua vähen-

nettyä merkittävästi ja vastaavan menetelmän lisääminen reaaliaikaiseen toteutukseen olisi yksinkertaista. Toinen vaihtoehto olisi Kalman-suotimen käyttö, jota Davison *et al.* hyödynsivät [7] MonoSLAM-algoritmissaan. Kalman-suodin tekee arvion järjestelmän tilasta aikaisemman tilan perusteella jota korjataan mittaustuloksien perusteella. Mikäli järjestelmä on mallinnettu oikein ja mittauksien tarkkuus on tunnettu, pystyy suodin tekemään optimaalisen arvion mittaustuloksien satunnaisvaihteluista huolimatta. Kalman-suotimen tuottama ennakkoarvio helpottaisi ja nopeuttaisi myös piirreparien etsintää, sillä piirteitä voisi etsiä vain paikoista joissa ne todennäköisesti sijaitsevat. Kalman-suodin mahdollistaa myös erilaisten paikannusmenetelmien yhdistämisen, jolloin työssä käytetyn kamerakopterin inertiasensoreita voisi hyödyntää kamerapaikannuksen tukena kuten Engel *et al.* [9].

Algoritmissa seurantaan käytettävät piirteet valitaan alustusvaiheessa ja niiden oletetaan sijaitsevan tasolla, mikä rajoittaa algoritmin käyttökohteita merkittävästi. Paikannuksen toiminta vaatii, että alustuksessa valittu kohde on koko ajan kameran näkökentässä ja tunnistettavissa siitä, mikä rajoittaa kameran asentoa ja etäisyyttä kohteesta. Lisäämällä algoritmiin uusien seurattavien piirteiden valinta ja niiden sijainnin määritys voitaisiin algoritmi laajentaa SLAM-algoritmiksi, joka pystyy kartoittamaan tuntematonta ympäristöä ja paikantamaan suhteessa siihen.

LÄHTEET

[1] AR.Drone 2.0. Parrot wi-fi quadricopter. Verkkosivu, viitattu 17.5.2015. Saatavissa: http://ardrone2.parrot.com/

[2] OpenCV API Reference – OpenCV 2.4.11.0 documentation. Verkkosivu, viitattu 17.5.2015. Saatavissa: http://docs.opencv.org/modules/refman.html

[3] Robot Operating System documentation. Verkkosivu, viitattu 17.5.2015. Saatavissa: http://wiki.ros.org/

[4] H. Bay, A. Ess, T. Tuytelaars, and L. Van Gool, "Speeded-up robust features (surf)," *Comput. Vis. Image Underst.*, vol. 110, no. 3, pp. 346–359, June 2008. Saatavissa: http://dx.doi.org/10.1016/j.cviu.2007.09.014

[5] G. Bradski and A. Kaehler, *Learning OpenCV: Computer Vision with the OpenCV Library*. O'Reilly Media, 2008.

[6] M. Calonder, V. Lepetit, C. Strecha, and P. Fua, "Brief: Binary robust independent elementary features," in *Proceedings of the 11th European Conference on Computer Vision: Part IV*, ser. ECCV'10. Berlin, Heidelberg: Springer-Verlag, 2010, pp. 778–792. Saatavissa: http://dl.acm.org/citation.cfm?id=1888089.1888148

[7] A. Davison, I. Reid, N. Molton, and O. Stasse, "Monoslam: Real-time single camera slam," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 29, no. 6, pp. 1052–1067, June 2007.

[8] J. Engel, T. Schöps, and D. Cremers, "LSD-SLAM: Large-scale direct monocular SLAM," in *European Conference on Computer Vision (ECCV)*, September 2014.

[9] J. Engel, J. Sturm, and D. Cremers, "Accurate figure flying with a quadrocopter using onboard visual and inertial sensing," in *Proc. of the Workshop on Visual Control of Mobile Robots (ViCoMoR) at the IEEE/RJS International Conference on Intelligent Robot Systems (IROS)*, Oct. 2012.

[10] H. Eren, *Inertial Navigation*. CRC Press, 01/29; 2014/10 2014, pp. 17–1–17–14, 09; M1: 2; doi:10.1201/b15474-20. Saatavissa: http://dx.doi.org/10.1201/b15474-20

[11] A. Gionis, P. Indyk, and R. Motwani, "Similarity search in high dimensions via hashing," in *Proceedings of the 25th International Conference on Very Large Data Bases*, ser. VLDB '99. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1999, pp. 518–529. Saatavissa: http://dl.acm.org/citation.cfm?id=645925.671516

[12] C. Harris and M. Stephens, "A combined corner and edge detector," in *Proceedings of the 4th Alvey Vision Conference*, 1988, pp. 147–151.

[13] R. I. Hartley and A. Zisserman, *Multiple View Geometry in Computer Vision*, 2nd ed. Cambridge University Press, 2004.

[14] A. Herout, I. Szentandrasi, M. Zacharia, M. Dubska, and R. Kajan, "Five shades of grey for fast and reliable camera pose estimation," in *Computer Vision and Pattern Recognition (CVPR), 2013 IEEE Conference on*, June 2013, pp. 1384–1390.

[15] G. Klein and D. Murray, "Parallel tracking and mapping for small ar workspaces," in *Mixed and Augmented Reality, 2007. ISMAR 2007. 6th IEEE and ACM International Symposium on*, Nov 2007, pp. 225–234.

[16] ——, "Parallel tracking and mapping on a camera phone," in *Mixed and Augmented Reality, 2009. ISMAR 2009. 8th IEEE International Symposium on*, Oct 2009, pp. 83–86.

[17] D. G. Lowe, "Object recognition from local scale-invariant features," in *Computer Vision, 1999. The Proceedings of the Seventh IEEE International Conference on*, vol. 2, 1999, pp. 1150–1157 vol.2.

[18] M. Muja and D. G. Lowe, "Fast matching of binary features," in *Proceedings of the 2012 Ninth Conference on Computer and Robot Vision*, ser. CRV '12. Washington, DC, USA: IEEE Computer Society, 2012, pp. 404–410. Saatavissa: http://dx.doi.org/10.1109/CRV.2012.60

[19] P. L. Rosin, "Measuring corner properties," *Comput. Vis. Image Underst.*, vol. 73, no. 2, pp. 291–307, Feb. 1999. Saatavissa: http://dx.doi.org/10.1006/cviu.1998.0719

[20] E. Rosten and T. Drummond, "Fusing points and lines for high performance tracking." in *IEEE International Conference on Computer Vision*, vol. 2, October 2005, pp. 1508–1511. Saatavissa: http://edwardrosten.com/work/rosten_2005_tracking.pdf

[21] ——, "Machine learning for high-speed corner detection," in *European Conference on Computer Vision*, vol. 1, May 2006, pp. 430–443. Saatavissa: http://edwardrosten.com/work/rosten_2006_machine.pdf

[22] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski, "Orb: An efficient alternative to sift or surf," in *Computer Vision (ICCV), 2011 IEEE International Conference on*, Nov 2011, pp. 2564–2571.

[23] J. Shen, J. Ryde, and H. Hu, *Landmarks and Triangulation in Navigation*. CRC Press, 05/04; 2014/10 2006, pp. 149–186, 09; M1: 0; doi:10.1201/9781420019445.ch4. Saatavissa: http://dx.doi.org/10.1201/9781420019445.ch4