Juha Vallinen

ETÄISYYDEN MITTAAMINEN STEREOKAMERAN JA OPENCV KIRJASTON AVULLA

Automaatiotekniikan koulutusohjelma

2017

samk
Satakunnan ammattikorkeakoulu
Satakunta University of Applied Sciences

ETÄISYYDEN MITTAAMINEN STEREOKAMERAN JA OPENCV KIRJASTON AVULLA

Vallinen, Juha
Satakunnan ammattikorkeakoulu
Automaatiotekniikan koulutusohjelma
Helmikuu 2017
Sivumäärä: 49
Liitteitä: 5

Asiasanat: konenäkö, stereokuvaus, stereokamera, C++, OpenCV

Opinnäytetyössä lähdettiin etsimään keinoja mitata etäisyyttä stereokonenäön avulla, tavoitteena saada Satakunnan ammattikorkeakoulun robotille ”näkökyky”. Tavoite oli, että stereokameralla tutkitaan robotin liikesuunnassa (vain eteenpäin) olevaa maisemaa ja pyritään havaitsemaan kohteet, joita robotin on väistettävä.

Toteutusta lähdettiin miettimään ECOM e-CAM_9V024_STEREO -kameralla, mutta sen sulautettu Linux järjestelmä osoittautui hankalaksi ja laskentakapasiteetti rajalliseksi. Kun tällä kameralla ei päästy toivottuun lopputulokseen, lähdettiin hakemaan toista ratkaisua edullisten USB kameroiden ja minitietokoneen yhdistelmällä. Toisaalta samalla oli tarkoitus tutustua myös Kinect-ohjaimen käyttömahdollisuuksiin.

Tutkimus osoittautui aika haasteelliseksi ja aikaa vieväksi, lopulta oli pakko todeta, että lopullinen ratkaisu jäi tavoitteista. Etäisyyttä onnistuttiin mittaamaan sekä ECOM kameralla, että USB-kameroilla, mutta lopullinen sovellus ja kytkeminen robottiin jäi tekemättä. Tutkimuksen löydökset viittaavat, että ECOM kameralla saisi tehtyä tarkemman mittauksen, mikäli keksisi vähemmän laskentatehoa kuluttavan tavan etsiä kuvasta kohteita. UDOO:ssa laskentateho ei ollut ongelma, kuten ECOM kamerassa, USB-kamerat + UDOO yhdistelmällä saataisiin etäisyys mitattua laadullisesti paremmilla kameroilla, Kinect-ohjain osoitti kykynsä etäisyyden mittauksessa, mutta miten etäisyystieto saataisiin robotin ymmärtämään muotoon ja siirretyksi robottia ohjaavalle ohjelmistolle vaatisi toisen opinnäytetyön. Seuraava opinnäytetyön tekijä toivottavasti saa tästä työstä pohjan omalle työlleen ja voi keskittyä varsinaiseen ongelmaan, eikä tarvitse painia perusongelmien kanssa.

THE DISTANCE MEASUREMENT THROUGH STEREO CAMERA AND THE OPENCV LIBRARY

Vallinen, Juha
Satakunnan ammattikorkeakoulu, Satakunta University of Applied Sciences
Degree Programme in Automation Technology
February 2017
Number of pages: 49
Appendices: 5

Keywords: Machine vision, stereo vision, stereo camera, C++, OpenCV

The purpose of this thesis was to look for ways to measure the distance with the stereo machine vision. The target was to provide “vision” for the robot at Satakunta University of Applied Sciences. The aim was to observe the robot's direction of motion (forward only) on the landscape with the stereo camera and detect objects which the robot should give way.

At first, it was studied how this could be implemented with the ECOM e-CAM_9V024_STEREO camera, but its embedded Linux system turned out to be quite cumbersome and its computing capacity was limited. When the desired result was not achieved with this camera, it was started to look for another solution. A combination of low-cost USB cameras and a minicomputer was selected. It was also decided to explore the possibilities of the Kinect control. The study turned out to be quite challenging and time consuming and eventually I was forced to conclude that the final settlement fell short of targets. The measurement of distance succeeded, but the final application and connecting to the robot were not done.

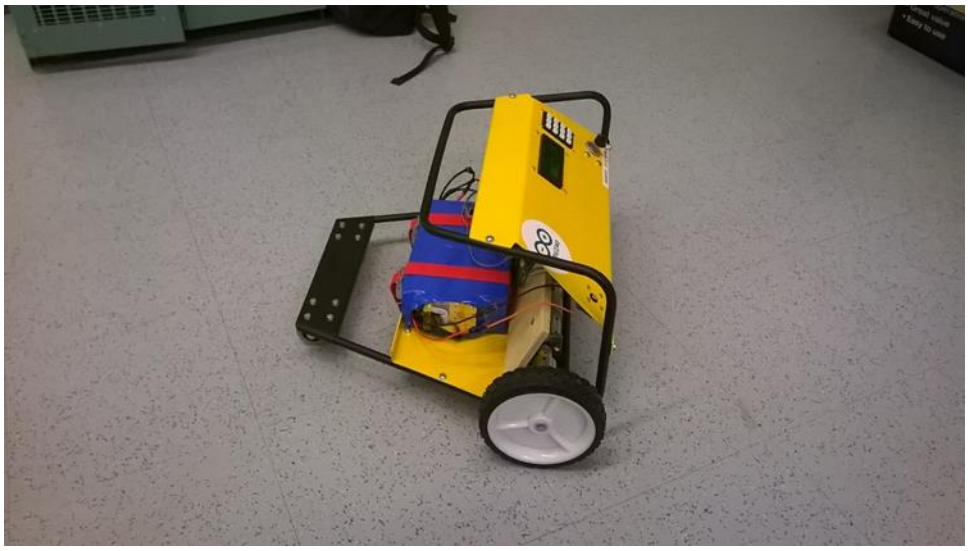
Each of the studied solutions had its weakness. It would be required another thesis to find a solution with enough computing capacity, accurate distance measurement and capability to transmit measurement data to the robot. This thesis provides good basis for further studies and includes information about identified basic problems.

.

SISÄLLYS

1 JOHDANTO

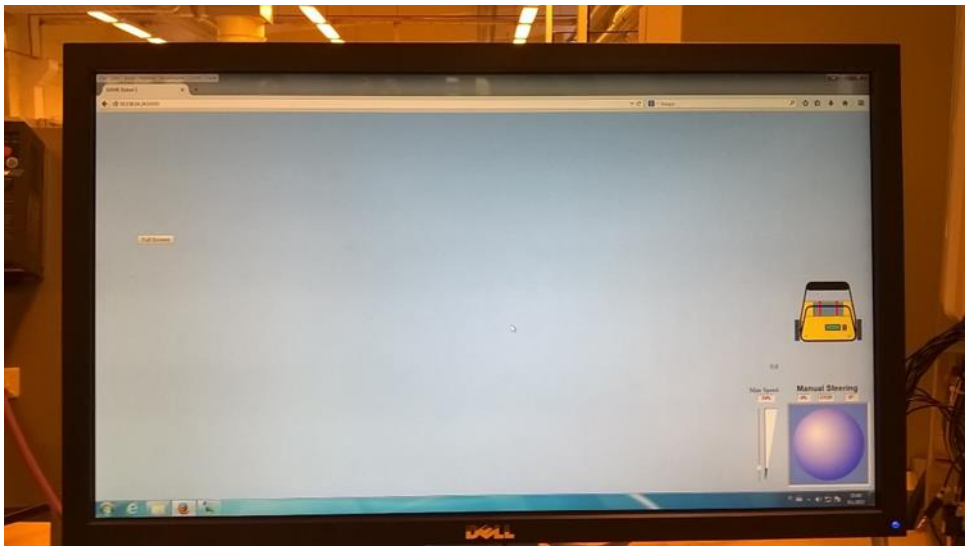
Tavoitteena oli ohjata ammattikorkeakoulun robottia (kuva 1) stereokamerasta saatavalla etäisyystiedolla. Koululla oli valmiina E-COM:n e-CAM_9V024_STEREO kamerajärjestelmä, jonka avulla ongelmaa lähdettiin ratkomaan. Tarkoitus oli kiinnittää kamera robotin päälle ja sen antaman etäisyystiedon avulla pysäyttää robotti ennen törmäystä ja jopa ohittaa este omatoimisesti.



Kuva 1 Kuva koulun robotista

Robottiin oli jo tehty ohjausjärjestelmä käyttäen Arduino-sulautettua tietokonetta. Arduino otti ohjauskäskyt vastaan WLAN:n välityksellä esim. tablettitietokoneelta ja muunsi käskyt moottoriohjauskortille sopiviksi komennoiksi.

Ohjausjärjestelmän käyttöliittymä kuvassa Kuva 2 tavallisessa pöytätietokoneessa. Pyörittämällä hiirellä alakulmassa olevaa palloa robotti kulkee ja kääntyy haluttuun suuntaan. Tavoite oli, että samassa käyttöliittymässä olisi näkynyt etäisyys esteeseen.



Kuva 2 Kuva ohjaus käyttöliittymästä

Etäisyyttä on perinteisesti tämänkaltaisissa sovelluksissa mitattu ultraäänianturilla, optisella anturilla (infrapuna), tai tässä tapauksessa jopa kosketusanturilla olisi saavutettu perustavoite eli estää estettäisiin törmääminen. Nyt lähdettiin hakemaan nykyaikaisempaa menetelmää ja yritettiin ratkaista ongelma stereokameralla.

2 STEREOKUVAUS

Stereokuvaus ei ole uusi asia, jo filmiaikakaudella on otettu stereokuvia, katselulaitteetkin poikkeavat nykyisestä, Kuva 3 esittää katselulaitetta 1800-luvulta.

Kuva 3 Stereoskoopi 1800-luvulta (Aalto, yliopisto, 2004)

Stereokuvauksella pyritään saamaan kolmiulotteinen vaikutelma kolmiulotteisesta kohteesta. Tässä käytetään hyväksi samaa menetelmää, jolla ihminen havainnoi etäisyyksiä eli molemmat silmät muodostavat oman kuvansa kohteesta, kuvien perspektiivi on erilainen. Näistä kuvista aivot muodostavat yhden stereokuvan.

Sama on mahdollista kameroilla, mutta kuvattaessa on huomioitava, että kuvat otetaan oikein, että ne muodostavat stereokuvan ja että tarkasteluhetkellä kuvat asemoidaan niin, että katsojan silmien on mahdollista muodostaa kuvista stereokuva.

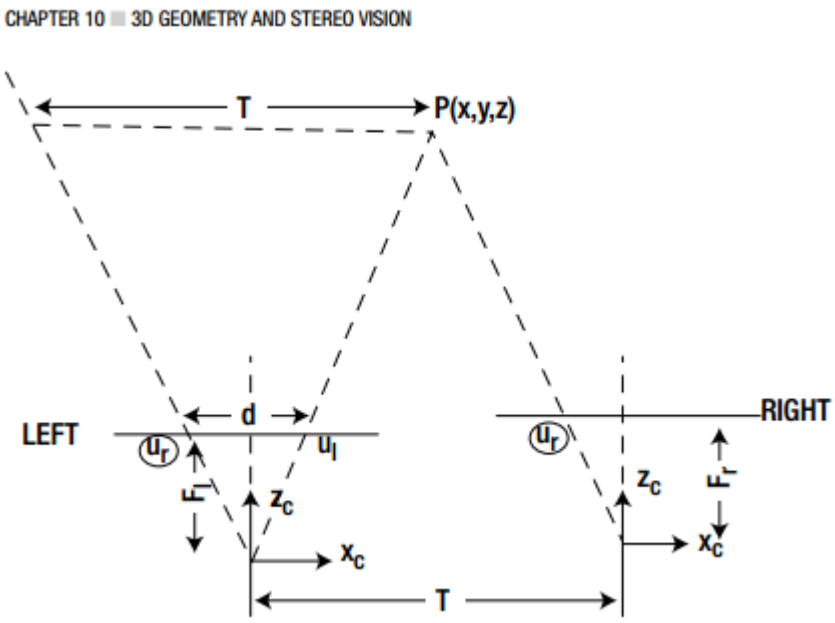
Konenäkösovelluksissa stereokuva voidaan muodostaa yhdellä kameralla, jos kohde liikkuu sopivasti, tai kahdella kameralla, jolloin saadaan kaksi kuvaa samanaikaisesti. On mahdollista myös ottaa useita kuvia (jos kohde on liikkumaton) tai käyttää useampaa kameraan, jos kohteesta halutaan kattavampi 3D malli.

Tässä työssä stereokuvauksella haluttiin ratkaista kuvassa näkyvien kohteiden etäisyys, tähän tarkoitukseen tarvitaan kaksi kameraa ja niiden pitäisi ottaa kuva samanaikaisesti, jotta vältetään liikkeestä johtuva virhe mittaustuloksessa.

2.1 Etäisyyden mittaamisen teoriaa

Etäisyyden mittaaminen stereokameralla perustuu siihen, että kahdesta samanaikaisesti otetusta kuvasta tunnistetaan sama reaalimaailman piste molemmista kuvista. Lisäksi kun tiedetään kameroiden polttoväli sekä kameroiden välinen etäisyys voidaan trigonometrian avulla laskea pisteen etäisyys kuvapinnasta.

Kuva 4 esittää teoreettisen trigonometria mallin siitä miten OpenCV kirjasto laskee pisteen etäisyyden, kuva on kirjasta Practical OpenCV sivulta 180 (Brahmbhatt, 2013).



Kuva 4 OpenCV kirjaston trigonometria malli

Kahdesta eri kuvasta voidaan mitata kuvapisteen ero, jota kuvassa merkitään d kirjaimella, se saadaan seuraavasti d = ul –ur , jossa ul ja ur ovat pisteen P koordinaatit vasemmassa ja oikeassa osakuvassa (kuvassa oikea kameran kuva on siirretty vasemman kameran kuvan vasemmalle puolelle havainnollistamaan kuvaa) . Kun huomioidaan trigonometrian säännöt yhtenevistä kolmiosta, niin voidaan laskea pisteen P Z-koordinaatti seuraavasti.

$$\frac{d}{T} = \frac{f}{Z} \quad eli \quad Z = T * \frac{f}{d}$$

Yhtälöstä tiedetään T kameralinssien etäisyys toisistaan, f kameroiden polttoväli. Pisteen paikkojen erotus kuvissa eli d saadaan laskettua kuvista.

2.2 Kuvien käsittely

Kuvien käsittelyyn on saatavilla valmiita ohjelmistoja varsinkin, jos käytetään konenäkökameroita. Kuvien käsittelyä voidaan helpottaa käyttämällä OpenCV (open source computer visio) kirjastoa (opencv.org, 2017) joka julkaistaan BSD lisenssin alla. BSD lisenssi antaa mahdollisuuden käyttää ja muokata kirjaston tarjoamia funktioita omiin tarkoituksiinsa opiskelu- ja kaupallisissa tarkoituksissa, eikä vaatimuksena ole koodin julkistamista. Eli OpenCV kirjastoa voi käyttää osana omaa kaupallista tuotetta. Toinen mahdollisuus on käyttää MathLab (MathWorks, Inc., 1994-2017) ohjelmistoa kuvin käsittelyyn. MathLab ohjelmisto on kuitenkin maksullinen.

Tässä työssä tutustuttiin erilaisiin kameroihin kuvien ottamiseksi, mutta kuvaa analysoitiin kaikissa tapauksissa käyttäen OpenCV kirjastoa. OpenCV kirjastosta kerrotaan lisää kappaleesta 5 sivulla 21.

3 TYÖN ETENEMINEN

3.1 Lähtökohdat

Jo koulutukseen hakeutuessani yhtenä ratkaisevana tekijänä oli SAMK:ssa annettava konenäkökoulutus. Olen harrastanut valokuvausta ja halusin jonkin erikoisosaamisalueen, jolla voisi erottua työttömien insinöörien massasta, jos irtisanominen osuu omalle kohdalle. Olen sitä mieltä, että konenäkö on liian vähän käytetty mahdollisuus.

Työtä aloittaessani en kuitenkaan ollut kovin hyvin perillä mitä itse konenäkö on. Luentosarja ja harjoitukset olivat kyllä hyviä, mutta aikaa oli vaan liian vähän koko konenäköalueen läpikäymiseen ja tutustuminen jäi aika pintapuoliseksi.

Haasteita tuli myös minulle oudommista käyttöjärjestelmistä, koodaustavoista kuten embedded Linux, Qt, kuin myös uudesta ohjelmointikirjastosta OpenCV. Kaikista näistä olin kyllä jotain lukenut ja kuullut, mutta en ollut niiden kanssa aiemmin työskennellyt.

Vielä kun samanaikaisesti jouduin päivätyössäni vaihtamaan toimenkuvaa radikaalisti, joka vaati paljon aikaa ja energiaa uuden oppimiseen, niin tämän opinnäytetyön tekeminen joutui taka-alalle ja venyi. Työssäni jouduin myös opiskelemaan Linux järjestelmien ylläpitoa, niin sen kautta tämäkin projekti sai uusia ideoita.

3.2 Työn kulku

Työtä aloiteltiin syksyllä 2014, ensimmäinen puoli vuotta meni openCV kirjallisuutta keräämällä ja tutustumalla ECOM e-CAM_9V024_STEREO stereokameraan (e-con systems, n.d.). E-com kamera osoittautui haasteelliseksi saada toimimaan vähäisellä Linux tietämyksellä. Kaikki kameran mukana tulleet esimerkkiohjelmat olivat toteutettu graafisella käyttöliittymällä ja vaativat toimiakseen hiiren ja sen kytkeminen kameraan tuotti ongelmia, vielä kun kameran mukana tullutta miniUSB – normiUSB adapteria ei löytynyt koululta. Hankin uuden adapterin, mutta sen sopimattomuus tä-

hän projektiin paljastui vasta paljon myöhemmin. Adapterilta vaadittiin OTG kyvykkyys, ilman sitä hiiri ei saanut sähkösyöttöä ja ei toiminut. Myynti-ilmoituksen mukaan tämän adapterin piti tukea OTG ominaisuutta, mutta näin ei ollut ja siitä syystä hiiri ei toiminut. Toista testilaitetta minulla ei ollut käytettävissä, johon adapteri olisi sopinut, niin adapterin toimimattomuus jäi toteamatta. Epäilin edelleen ongelman olevan kameran konfiguraatiossa ja kamera jäi odottamaan pöydälle.

Alkuvuodesta 2015 kun opettajien kanssa keskustelin tuosta ongelmasta, niin keksittiin ajatus, että saataisiinko vastaava toiminnallisuus käyttäen kahta edullista USB kameraa. Koska robotissa oleva Arduino mikro-kontrolleri ei ole kovinkaan tehokas, päätettiin samalla ottaa käyttöön järeämpi sulautettu tietokone ja hankittiin UDOO QUAD. UDOO koneessa on yhdistettynä mini Linux kone ja Arduino yhteensopiva mikro-kontrolleri (UDOO.ORG, 2016). UDOO:sta tarkemmin kappaleessa 4.2.1 UDOO.

Kun noudin koululta näitä laitteita, keskustelin konenäkölaboratoriossa lehtori Mirka Leinon ja muiden paikalla olleiden kanssa. Keskustelussa päädyttiin siihen, että otan rinnalle vielä Microsoft:n Kinetic liikeohjaimen, jolla varmasti etäisyyden mittaaminen onnistuisi. Kinect-ohjain on rakennettu liikkeentunnustusta silmällä pitäen. Siinä yhdistetään RGB kameran värikuva sekä laserprojektorin + mustavalkokameran tuottama ”etäisyys” informaatio. Näiden avulla ohjain tuottaa kuvan, jossa eri etäisyydet ilmaistaan värien avulla. Ohjain on tarkoitettu xbox360 pelikonsolille ja sillä korvataan kädessä pidettävä peliohjain (wikipedia, Kinect, 2015).

Näiden laitteiden kanssa projekti lähti etenemään, UDOO:n Linux puolelle sain kamerat kiinni ja kuvien ottokin onnistui. OpenCV ja Qt kirjastojen virittäminen UDOO:lle tuotti kuitenkin haasteita, mutta senkin sain toimimaan. Qt kirjaston mukaan ottaminen johtui siitä, että oli ajatus hyödyntää e-con kameran mukana tulleita esimerkkiohjelmistoja, joiden graafinen liityntä oli toteutettu hyödyntäen Qt-kirjastoa.

Kun oikeita stereokuvia vihdoin saatiin kuvattua, alkoi nousta esiin uusia ongelmia. Edullisten USB kameroiden laatupoikkeavuudet aiheuttivat kuviin liikaa kohinaa ja muotovirheitä, joka taas vaikeutti kameroiden kalibrointia ja tätä kautta etäisyyden mittaamista, mutta tästä lisää kappaleessa 4.2.2 USB-kamera. Samaan aikaan kokeilin

Kinect ohjainta, valmiilla ohjelmilla etäisyyskaavio tuli selvästi näkyviin, mutta kun yritin saada toimivaa käännösympäristöä rakennettua Linux-koneeseen, niin törmäsin seinään. En saanut aikaiseksi sellaista kokoonpanoa, jolla olisin saanut itse käännetyn koodin toimimaan. Kinect vaatii Linux-ympäristössä toimiakseen OpenNI ja Nite2 kirjastot (kdab group, 2012).

Tässä kohtaa iski totaalinen aikapula ja tämä työ siirtyi hyllylle, jatkuakseen vasta keväällä 2016. Keväällä 2016 siirsin koodauksen ja kameroiden testauksen Linux läppärille. Työ nopeutui ja helpottui jonkin verran, pöydällä oli johtoja ja laitteita huomattavasti vähemmän ja netistä sai helpoimmin kopioitua tavaraa, koodien kääntäminen ja kalibrointiajotkin nopeutuivat. Lisäksi työ oli helpompi ottaa mukaan. Tästä huolimatta aikapula venytti projektia. Loppusyksystä 2016 pistettiin sitten isovaihde päälle ja ennen vuodenvaihdetta sain mielestäni etäisyyden tunnistuksen tehtyä niin hyvin, kuin se edullisilla USB kameroilla on mahdollista. Samat ohjelmat eivät vaan tuottaneet e-com kameran kanssa toivottua lopputulosta, sillä kameran sisäänrakennetusta embedded linux koneesta loppuvat tehot.

Vuodenvaihteen jälkeen vielä kokeilin miten läppärillä pyöritetyt ohjelman toimivat UDOO laudalla, ja ensimmäinen ongelma oli saada openCV 3.x päivitettyä UDOO:lle. Lopulta UDOO vaati koko käyttöjärjestelmän uudelleen asentamisen, eli microSD kortille asennettiin uusi versio UDOObuntu 2.1 (Ubuntu 14.04), ja sen päälle tarvittavat ohjelmat.

4 TUTKITTAVAT KAMERAJÄRJESTELMÄT

4.1 E-com Systems e-CAM_9V024_STEREO E-COM

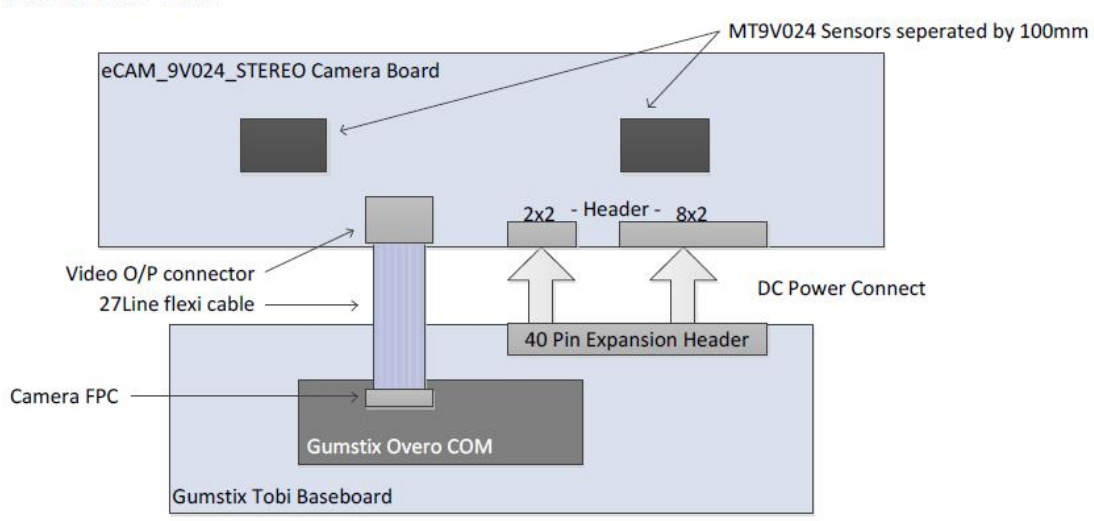
E-CAM _9V024 on E-com Systems:n referenssikamera stereokuvaukseen. Siinä on kaksi pixel synkronoitua OMAP35x kameraa, jotka valmistaja on valmiiksi kalibroinut ja kiinnittänyt. Lisäksi kamera pitää sisällään embedded Linux moduulin, jonka ohjelmistoon valmistaja on tehnyt muutoksia niin, että stereokuvaksesta saadaan paras hyöty. Ohjelmistoon on tehty standardi V4L2-ajuri ja OpenCV SDK. Lisäksi kameran mukana tulevat valmiit esimerkkiohjelmat kameran kalibrointiin ja etäisyyden mittaamiseen. Etäisyyden mittaaminen tapahtuu ainoastaan ympyrän muotoiseen kappaleeseen. Eli ohjelma etsii kuvista ympyrän muotoisen kappaleen ja antaa X,Y ja Z koordinaatit kappaleen keskelle.

Teknisiä tietoja:

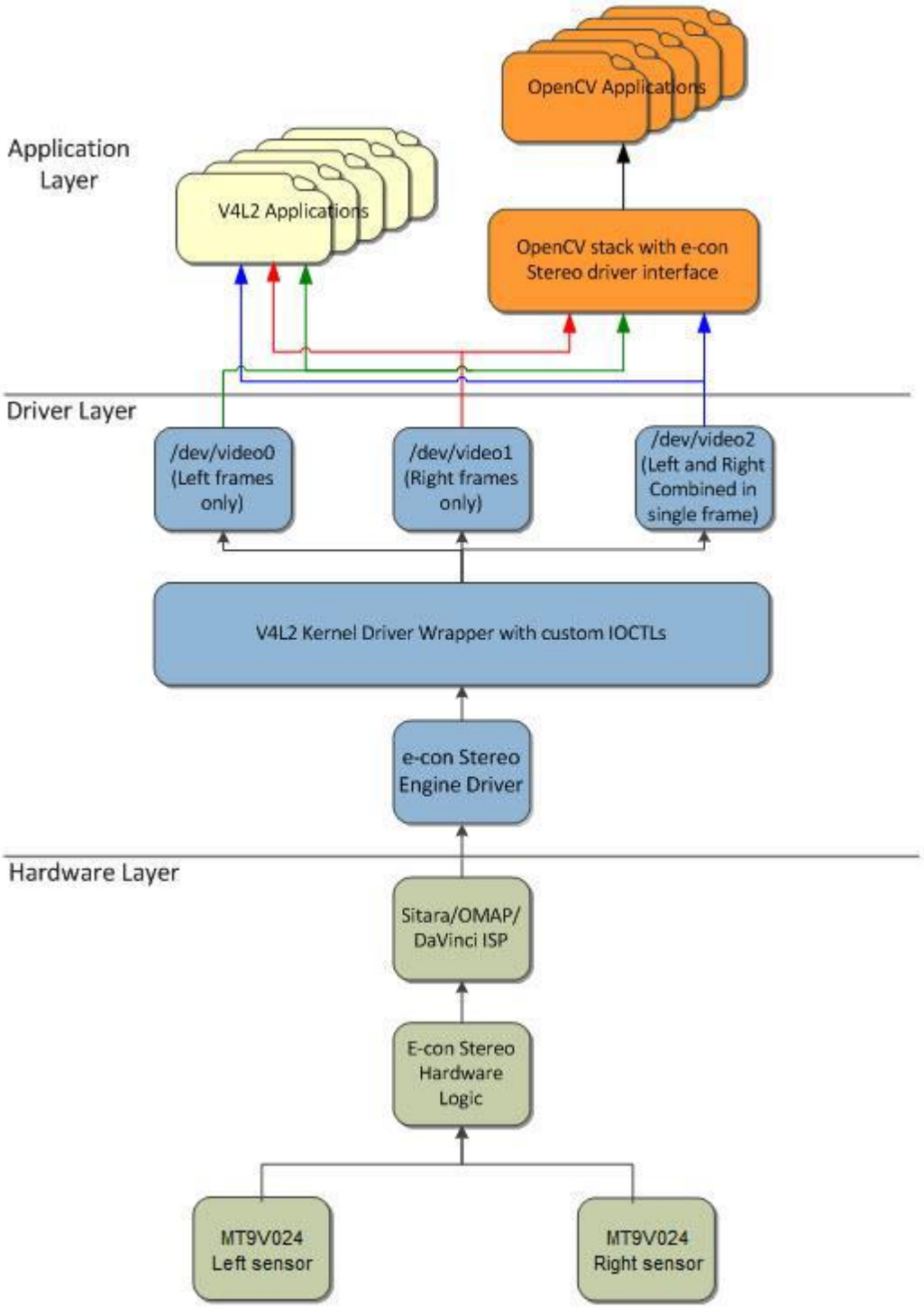
- Kaksi 1/3” m/v MT9V024 CMOS kuva kennoa
- Kuva kennojen tarkkuus WVGA 752 x 480, kuvien tarkkuus 736x480 8-bit harmaasävy.
- Video 30 kuvaa/sekunti /kenno, synkronoituna (video0, vasen ja video1, oikea)
- Tai yhdistettynä yhdeksi video ulostuloksi (video2) 1472 x 480 pistettä, 30 kuvaa/sekunti
- Linssien välinen etäisyys (baseline distance) 100mm
- S-mount linssin pidin (M12 P0.5) jotka on lukittu ja esi-kalibroitu
- Standardi V4L2 ja yhteensopiva ajuri ohjelmisto
- Linux 2.6.35 kernel
- OpenCV SDK (e-con Systems India Pvt Ltd, 2012)

Kuva 5 esittää kameran lohkokaavion. Camera Board pitää sisällään kamerat ja niiden ohjaukseen vaadittavan elektroniikan. Gumstix moduli taas sisältää embedded Linux osuuden. Kuva 6 esittää miten kameran eri toiminnallisuudet on jaettu eri kerroksiin,

fyysinen kerros (Hardware Layer) sisältää kamerat ja ohjauselektroniikan, ohjainkerros (Driver Layer) tarjoaa standardi V4L2 rajapinnan ohjelmistoille, tähän kerrokseen valmistaja on lisännyt kolmannen videoliitynnän, jonka kautta kamera pystyy tarjoamaan oikean ja vasemman kameran kuvista muodostetun yhdistelmäkuvan. Ohjelmistokerroksella (Application Layer) voi olla V4L2 ohjelmistoja, mutta tämän työn kannalta mielenkiintoisempi oli valmistajan oma OpenCV kirjasto ja sen päälle tehdyt OpenCV ohjelmat.



Kuva 5 Kameran lohkokaaavio (e-con Systems India Pvt Ltd, 2012).



Kuva 6 Kameran toiminnallisuudet jaettuna kerroksiin (e-con Systems India Pvt Ltd, 2012).

Kameran mukana tullut kalibrointiohjelma pääsi parhaimmillaan todella hyvään 0,17 RMS arvoon, kun käytin kuvakokoa 736/480. Mutta tällä kuvakoolla etäisyyden mittaus omalla ohjelmallani oli tuskastuttavan hidasta. Etäisyydenmittausohjelmassa oleva parametrien haarukointi liukusäätimillä oli mahdoton tehtävä, kun hiirikursori päivittyi näytölle noin joka kymmenes sekunti.

4.2 UDOO + USB kamerat

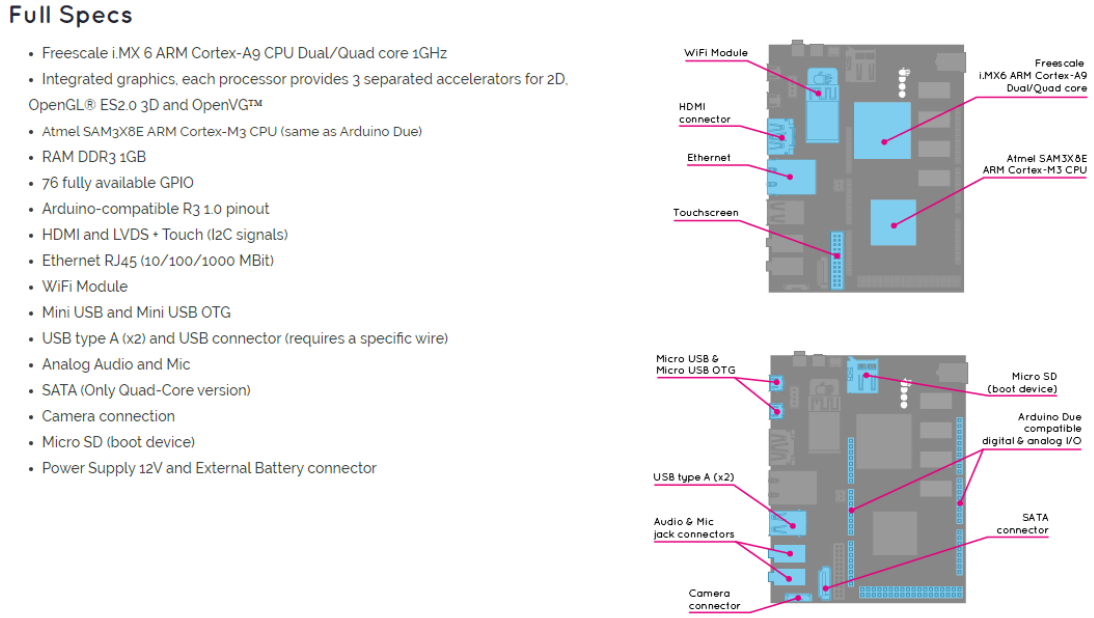
Tällä yhdistelmällä lähdettiin tavoittelemaan samaa toiminnallisuutta kuin e-COM:n laitteella toimiessaan olisi ollut, mutta tee-se-itse periaatteella. USB kameroiden lähtötarkkuus oli jo huonompi kuin e-COM kamerana eli 640x480 ja siitäkin jouduttiin vielä tinkimään kuva resoluutioon 320x240, koska USB2.0 porttien kaistan leveys ei riittänyt täyden tarkkuuden siirtämiseen kuin ajoittain. Toisaalta tämä yhdistelmä sopisi robottiin paremmin, koska sillä voitaisiin korvata robotin nykyinen Arduino tietokone. UDOO laudassa on Arduino ”sisään” rakennettuna ja ulkoiset liitännät ovat yhteensopivia nykyien kanssa.

4.2.1 UDOO

UDOO-minitietokone hankittiin virallisesta UDOO verkkokaupasta. Malliksi valikoitiin tehokkain malli, koska pelkona oli prosessoritehon riittämättömyys kuvien käsittelyyn openCV kirjastolla.

UDOO valikoitiin alustaksi siksi, että siinä on yhdistettynä sekä Linux-tietokone (vastaava kuin RaspberryPI), että Arduino sulautettu tietokone. Robotissa on jo käytössä Arduino ohjaamassa moottoreita. Kun kamerasovellus saadaan hiottua toimivaksi, voidaan nykyisen Arduinon toiminnallisuudet siirtää UDOO kortille, tiedon siirtoon voidaan käyttää UDOO:n sisäistä sarjaliikenne väylää Linux ja Arduino yksiköiden välillä.

UDOO:n tärkeimmät liitännät ja spesifikaatiot:



Kuva 7 UDOO tekniset tiedot (UDOO.ORG, 2016)

4.2.2 USB-kamerat

USB-kamerat hankittiin Clas Ohlson:lta, hintaan 15,99 euroa kappale. Hintaluokka e-CON:iin verrattuna oli aivan eri, E-CON hinta on 1299 dollaria.

Kuva 8 USB-kamera (Clas Ohlson, 2016)

Kuten hinnasta voi päätellä, kyseessä ei ole mikään huipputuote. Toki kamerat ajavat asiansa sovellutuksissa, joihin ne on tarkoitettu eli skype puheluissa ja vastaavissa. Stereokuvaukseen niiden mekaaninen laatutoleranssi ei riitä. Kameroiden linssin ja kennon välinen ero aiheutti ongelmia, koska kameroiden polttoväli ei ollut täysin sama. Tästä aiheutui se, että kameroiden kuva-ala ei ole täysin sama, kun kameroiden

linssit asemoitiin samalle linjalle. Kameroiden syväterävyys oli myös kohtalaisen kapea, eli jos kamera säädettiin tarkaksi esimerkiksi viiden metrin päässä oleville kohteille, niin 50 senttimetrin päässä olevat kohteet olivat epäselviä ja päinvastoin. Samoin linssien ja kennojen laadussa oli eroa, toinen kameroista tuotti selvästi kohinaisemman kuvan ja terävyyskin oli astetta heikompi. Kuva 9 antaa kuvan siitä, minkä laatuiset linssit kameroissa on.

Kuva 9 USB kameran linssin laatu

En osannut tuollaista edes etsiä ja kuva tulikin vähän sattumalta, kun yritin säätää kameroiden tarkkuutta ja ruuvasin linssin kokonaan auki ja olin laittamassa sitä takaisin paikoilleen. Eli kuva ei ole suurennus vaan linssi on niin etäällä kennosta, että kenno ”tarkentui” linssin kohdalle.

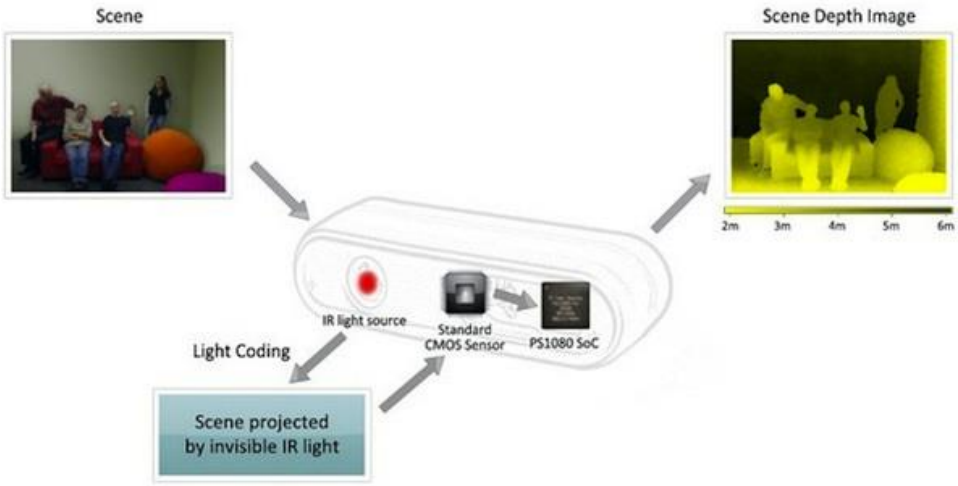
Stereokuvaussovelluksessa olisi myös ollut etua, että kamerat olisi saanut kiinnitettyä vielä paremmin paikalleen. Kun kamerat oli tarkoitus siirtää robottiin, niin en halunnut lähteä kameroita liimaamaan mihinkään tukevaan levyyn, vaan kiinnitin kamerat paksuun pahviin mahdollisimman tukevasti.

Kameroiden mukana ei tullut Linux yhteensopivia ajureita, mutta Linux yleis-USB kamera-ajurit toimivat hyvin ja guvcview-ohjelmalla, josta enemmän kappaleessa 7.2 Guvcview, sai varmistettua missä liitännässä kamerat milloinkin olivat. Jos kamerat irrotti välillä Linux koneessa, niin kameroiden portti vaihtui ja se piti huomata myös vaihtaa ohjelmiin. Samalla ohjelmalla sai kameroiden parametreja säädettyä.

4.3 Kinect-ohjain

Kinect-ohjain on alun perin tehty xbox360-pelikoneelle ja sen tehtävä on tunnistaa ihmishahmo ja sen liikkeet ja niiden perusteella siirtää hahmoa pelissä. Ohjaimen kamera tuottaa 640x320 pikselin kuvan 30 kuvaa/s nopeudella. (wikipedia, Kinect, 2015)

Kinect-ohjainta en lopulta juurikaan ehtinyt tutkimaan, mutta ne muutamat kokeilut mitä tein, vakuuttivat ohjaimen toimivan. Varsinaista numeerista etäisyyttä en saanut mitattua, koska ohjelmistojen muokkaukseen ei aika riittänyt. Kameran tarjoaman etäisyystiedon pohjalta tehty kuva kuitenkin vastasi hyvin todellisuutta. Kuva 10 havainnollistaa ohjaimen toimintaa.



Kuva 10 Kinect-ohjaimen toiminta (ifixit, 2016).

Ohjelmistojen muokkaukseen olisi tarvinnut asennella useita uusia kirjastoja Linux ympäristöön, ohjeita löytyy Open Kinect-sivustolta (open kinect, 2011).

Kinect & Python asennusohjeita:
https://www.kdab.com/setting-up-kinect-for-programming-in-linux-part-1/
http://www.cc.gatech.edu/~ahuaman3/docs/tutorials/software/source/perception.rst

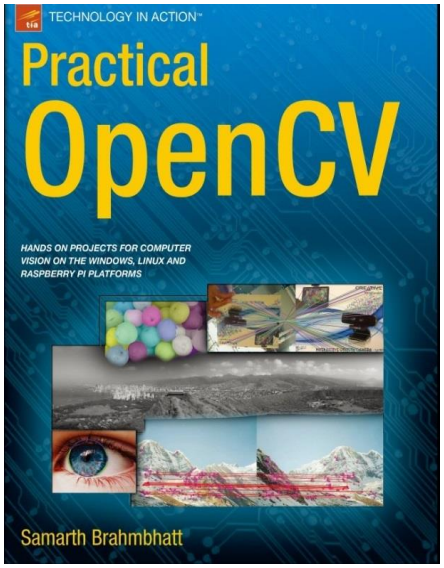
5 OPENCV KIRJASTO

OpenCV kirjasto on tehty BSD lisenssin alla. Kirjastosta löytyvät liitynnät yleisimmille ohjelmointikielille kuten C, C++, Java ja PYTHON. Kirjastoa on kehitetty vuoden 2000 kesäkuusta, sitä on koodattu C ja C++ kielillä, mutta uudet ominaisuudet tehdään vain C++:lla (opencv.org, 2016).

OpenCV kirjaston versiot 2.x käyttävät C-kielen syntaksia ja funktiokutsut on toteutettu samoin C-kielen tavoin. 3.x versioissa on siirrytty C++ käyttämiseen, joka näkyi erityisesti funktio kutsuissa, joita piti muokata, kun siirryin käyttämään kirjaston 3.x versiota. Kirjallisuudessa ja netissä esimerkit olivat lähes poikkeuksetta kirjoitettu kirjaston 2.x version mukaisella notaatiolla. Tässä työssä käytettiin vain hyvin pientä osaa openCV kirjaston tarjoamista funktioista.

5.1 Materiaalia opiskeluun

OpenCV.org sivustolla on jo paljon materiaalia opiskeluun. Samoin netistä löytyy lukuisia sivustoja, joissa OpenCV-kirjaston käyttöä on avattu. Stereokuvaukseen löysin hyvän step-by-step ohjeen kirjasta Practical OpenCV.



Kuva 11 Practical OpenCV kirjan kansikuva (Brahmbhatt, 2013)

Aluksi kirjassa käydään läpi OpenCV kirjaston asentaminen Linux koneeseen ja miten sen asennusta voi muokata, jos esimerkiksi tarvitaan joitakin lisäominaisuuksia tai ne halutaan pois (ccmaken käyttö). Sen jälkeen käydään OpenCV kirjaston käyttö läpi erilaisin esimerkein ja kappaleessa 10 käydään vaihe vaiheelta läpi, miten stereokameralla saadaan mitattua etäisyyttä. Kirjan koodiesimerkit löytyvät Github palvelusta (GitHub, practical opencv, 2016). Kirja löytyy myös Google DOCs listalta.

Huonoina tai hankalina asioina kirjasta voisi nostaa esille:

- Kirja on kirjoitettu aikana jolloin OpenCV:stä käytettiin versiota 2.x. Työssä käytin versiota 3.x ja näiden versioiden välillä on kaksi merkittävää muutosta: 2.x versiossa on käytössä C-pohjainen syntaksi, kun taas uudemmassa 3.x versiossa on käytössä C++ syntaksi.

  openCV 2.x tapa:

```
StereoSGBM stereo;
....
stereo.preFilterCap = 63;
stereo.SADWindowSize = 3;
stereo.P1 = 8 * 3 * stereo.SADWindowSize * stereo.SADWindowSize;
stereo.P2 = 32 * 3 * stereo.SADWindowSize * stereo.SADWindowSize;
stereo.uniquenessRatio = 10;
stereo.speckleWindowSize = 100;
stereo.speckleRange = 32;
stereo.disp12MaxDiff = 1;
stereo.fullDP = true;
```

  openCV 3.x tapa:

```
Ptr<StereoSGBM> stereo = StereoSGBM::create(0,16,3);
....
 stereo->setPreFilterCap(63);
 stereo->setBlockSize(3); //3

 stereo->setP1(8 * 3 * stereo->getBlockSize() * stereo->getBlockSize());
 stereo->setP2(32 * 3 * stereo->getBlockSize() * stereo->getBlockSize());

 stereo->setUniquenessRatio(10);
 stereo->setSpeckleWindowSize(100);
 stereo->setSpeckleRange(32);
 stereo->setDisp12MaxDiff(1);
 stereo->setMode(StereoSGBM::MODE_SGBM);
```

- Kirjassa käytetään boost nimistä kirjastoa. Boost kirjasto oli käytössä hakemistojen ja tiedostojen käsittelyssä. Boost kirjaston ja openCV version 3.x yhteistoiminnassa oli ongelmia saada kaikista lib tiedostoista oikeat versiot asennettua. Lopulta korvasin kohdat Qt:n vastaavilla toiminnollisuuksilla. Esimerkki tiedostojen hallinnasta:

Boost-kirjastolla:

```
// Read images
for(directory_iterator i(path), end_iter; i != end_iter; i++)
{
string filename = path + i->path().filename().string();
                        images.push_back(imread(filename));
}
```

Qt-kirjastolla:

```
// Read images

   QDirIterator it(path);
   while (it.hasNext())
   {
      QString filename = path + it.fileName();
      if (it.fileName().contains("jpg"))
      {
            images.push_back(imread(filename.toStdString()));
      }
      it.next();

   }
```

- Ohjelmakoodit eivät olleet viimeiseen asti testattuja ja kaatuivat joissakin tilanteissa. Alla esimerkki tilanteesta, missä ohjelma jäi jumiin tai kaatui kokonaan, kun num_disp (int tyyppinen) sai arvon 0.

```
void on_numDisp(int num_disp, void * _disp_obj)
{
disparity * disp_obj = (disparity *) _disp_obj;
num_disp = (num_disp / 16) * 16;
setTrackbarPos("numDisparity", "Disparity", num_disp);
disp_obj -> set_numDisp(num_disp);
}
```

Paranneltu versio:

```
void on_numDisp(int num_disp, void * _disp_obj)
{
   disparity * disp_obj = (disparity *) _disp_obj;

   if (num_disp < 16) num_disp=16;
   num_disp = (num_disp / 16) * 16;
   setTrackbarPos("numDisparity", "Disparity", num_disp);
   disp_obj -> set_numDisp(num_disp);
}
```

Toinen tutustumisen arvoinen kirja on Learning OpenCV (Kaehler, Learning OpenCV, 2008), tässä kirjassa ei ole kokonaisia esimerkkikoodeja, vaan pätkiä ohjelmista, mutta toisaalta teoriaa on selvitetty paljon syvällisemmin matematiikan avulla.

Kolmas kirja mistä perusteita tutkin oli OpenCV 2 Computer Vision Application Programming Cookbook, tässä kirjassa käydään asioita läpi kevyemmällä teorialla käytännön esimerkkien kautta. Tämänkin kirjan esimerkkikoodit löytyy valmiina github:sta (GitHub, OpenCV2Cookbook, 2012). Tästä kirjasta löysin idean etsiä kuvista kiinnostavia pisteitä ja niiden etäisyyksiä. Tämä tuntui lopulta antavan parhaan lopputuloksen USB-kameroilla, mutta se vaatii reilusti laskentatehoa (Laganière, 2011),. UDOO laskenta teho oli riittävä kuvien käsittelyyn, mutta E-COM:n kameralla samaa ohjelmaa ei voinut käyttää, ruudun päivitys nopeus putosi noin kuva kymmenessä sekunnissa nopeuteen.

5.2 Työn kannalta tärkeimmät OpenCV operaatiot/funktiot

5.2.1 Ikkunoiden käsittely

NamedWindow operaatiolla asetetaan esitettävälle ikkunalle nimi (TITLE) joka näkyy ikkunan yläpalkissa, esim. namedWindow("Left"), lisäksi voidaan antaa toinen parametri (flags) jolla kerrotaan, onko ikkunan tyyppi NORMAL vai AUTOSIZE. NORMAL tapauksessa käyttäjä voi säädellä ikkunan kokoa näytöllä, AUTOSIZE tapauksessa ikkuna säätyy itsenäisesti esitettävän kuvan koon mukaan.

Imshow operaatiolla ikkuna aktivoidaan eli ikkuna tulee näkyviin esim. `imshow("Left", framel)`, jossa framel cv::mat tyyppinen Array, kuvasta joka halutaan esittää ikkunassa. Mahdolliset kuvaan lisättävät tekstit ym. Pitää lisätä kuvaan ennen imshow-kutsua.

MoveWindow operaatiolla pystytään määräämään ikkunan paikka näytöllä.

```
moveWindow("Left",50,50);
```

Tätä tarvittiin erityisesti UDOO:n kanssa, sillä UDOO:ssa ikkunat piirtyivät aivan kuvan yläreunaan ja niitä ei pystynyt hiirellä siirtämään, koska käyttöjärjestelmän valikkopalkki piirtyi kuvien palkkien päälle

5.2.2 Kuvatiedostojen käsittely

Kuvatiedostoja on helppo käsitellä kahdella operaatiolla:

```
Mat imgL = imread (string(LEFT_FOLDER) + l_name.str(),flags );
```

```
imwrite(string(LEFT_FOLDER) + l_name.str(), framel, flags);
```

Imread lukee kuvan tiedostosta Mat tyyppiseen muuttujaan, flags-parametreillä voidaan kertoa, että luettava kuva talletetaan harmaasävy kuvana muuttujaan. Imwrite kirjoittaa kuvan levylle annettuun tiedostoon, tiedoston nimessä pitää olla mukana hakemistopolku. Tiedoston nimi on String tyyppinen, toinen funktion parametri on Mat tyyppinen array kuvasta. Flags-parametrilla pystytään määrittämään jpg kuvan laatu ja kompressointiaste.

5.2.3 FindChessboardCorners

FindChessboardCorners-operaatiolla etsitään annetun kokoista shakkilautaa annetusta kuvasta, jos kuvio löytyy, palauttaa funktio TRUE arvon ja ruutujen risteyskohdat. Tosin tämä ei palauta kulmien tarkkaa paikka, tarkka paikka etsitään käyttäen komentoa CornerSubPix.

Esimerkki:

```
bool pattern_found = findChessboardCorners(
        im,                 // kuva josta shakkikuviota etsitään
        Size(width, height),// etsittävän shakkikuvion koko
        im_p,               // paluu muuttuja, ruudukon risteys kohdat
         CALIB_CB_ADAPTIVE_THRESH + //flags
         CALIB_CB_NORMALIZE_IMAGE +
          CALIB_CB_FILTER_QUADS);
          // + CALIB_CB_FAST_CHECK);
```

Flags:

- CALIB_CB_ADAPTIVE_THRESH, kun asetettu käytetään adaptiivista raja-arvoa, kun kuvaa muunnetaan mustavalkokuvaksi, toinen vaihtoehto olisi käyttää kiinteää kuvan histogrammista valittua keskiarvoa.
- CALIB_CB_NORMALIZE_IMAGE, kun asetettu kuvalle suoritetaan gamma normalisointi ennen threshold:n ajoa.
- CALIB_CB_FILTER_QUADS, kun asetettu käytetään lisäkriteereitä virheellisten löydösten erotteluun. Pyritään parempaan lopputulokseen, mutta hidastaa etsintää.
- CALIB_CB_FAST_CHECK, pyritään löytämään shakkilaudan kulmat, ja lopettamaan kutsu nopeasti, mikäli mitään ei löydy. Tällä voi olla iso vaikutus nopeuteen. USB kameroiden kanssa parametri aiheutti

sen, että shakkilautoja ei löytynyt kuvista ja kameroiden kalibrointi ei onnistunut.

Myös tilanne missä shakkilauta on isompi, siis ruutuja enemmän, kuin kutsuparametrissa annetaan, aiheuttaa paluuarvoksi FALSE arvon. Eli funktiolla etsitään juuri tietyn kokoista lautaa.

5.2.4 CornerSubPix

CornerSubPix, operaatiolla etsitään ruudukon kulmien paikka tarkemmin.

```
cornerSubPix(
            grayIMG,// harmaasävy kuva, josta shakki kuviota etsitään
            im_p,   // ruutujen kulmat in/out
            Size(7,5), //shakkikuvion sisäpuolisten pisteiden määrä
            Size(-1, -1), //tarkennus alueen koko, -1,-1 tarkoittaa,
                          //että kokoa ei ole määritelty
            TermCriteria(CV_TERMCRIT_EPS + CV_TERMCRIT_ITER, 30, 0.1))
```

TermCriteria määrittelee, miten pitkään tarkennusta jatketaan, esimerkin tapauksessa tarkennusta jatketaan 30 kierrosta (CV_TERMCRIT_ITER, 30) tai kun pisteen paikka siirtyy lähemmäksi kuin criteria.epsilon (CV_TERMCRIT_EPS, 0.1). Funktiolle viedään findChessboardCorners-funktion palauttama tieto shakkilaudan risteyksien paikoista ja tämä funktio pyrkii tarkentamaan jokaisen risteyksen paikka ja palauttaa pisteiden tarkennetut paikat.

5.2.5 CalibrateCamera

CalibrateCamera operaatiolla lasketaan kameran kalibrointiparametrit (intrinsic and extrinsic parameters) shakkilautakuvien perusteella.

```
float rms_error = calibrateCamera(
            object_points, //in, kuvan 3D esitys
            image_points, //in, kuvan 2D esitys
            images[0].size(), //in, kuvan koko
            cameraMatrix, // out, matriisi, kts. alla
            distCoeffs, // out, vääristymä matriisi
            rvecs, //out, rotation vectors kts. alla
            tvecs); //out, translation vectors kts. alla
```

rvecs, tvecs ovat kuvan kierto ja käännösvektorit, eli paljonko kuva on kiertyneenä pystyakselinsa suhteen (rvecs) ja paljonko kääntyneenä vaaka-akselinsa ympäri(tvecs).

Kameran ulostulo matriisi, $$\text{camera matrix } \boldsymbol{A} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}$$

fx, fy = polttoväli pikseleinä

cx,cy = kuvan optinen keskipiste pikseleinä

Tulokset yleensä talletetaan tiedostoon myöhempää käyttöä varten. Funktiolle voidaan myös viedä iso joukko parametreja, joilla voi vaikuttaa funktion toimintaan, mutta niillä ei tässä yhteydessä ole juurikaan merkitystä, koska tehdään ensimmäistä kalibrointia kameroille, eikä pohjatietoja ole.

5.2.6 StereoCalibrate

StereoCalibrate komennolla saadaan kalibroitua kamerat keskenään, funktio palauttaa kameroiden välisen kierto-ja kääntömatriisin, eli paljonko kameroiden pysty- ja vaaka-akselin välinen kulma poikkeaa toisistaan. Samoin funktio palauttaa RMS arvon, joka kertoo, kuinka hyvin kalibrointi onnistui, mitä pienempi arvo sitä parempi.

```
double rms = stereoCalibrate(object_points,
            l_image_points, //in, vasemmankuvan kulmat
            r_image_points, //in, oikeankuvan kulmat
            l_cameraMatrix, //in/out, vasemman kameran kamera matriisi
            l_distCoeffs,   //in/out, vasemman kameran vääristymä matriisi
            r_cameraMatrix, //in/out, oikean kameran kamera matriisi
            r_distCoeffs,   //in/out, oikean kameran vääristymä matriisi
            l_images[0].size(), //in, kuvan koko
            R, T, E, F      //out kts alla
            ,CALIB_FIX_ASPECT_RATIO +
             CALIB_ZERO_TANGENT_DIST +
            //CALIB_SAME_FOCAL_LENGTH +
             CALIB_RATIONAL_MODEL +
             CALIB_FIX_K3 + CALIB_FIX_K4 + CALIB_FIX_K5,
            TermCriteria(TermCriteria::COUNT+TermCriteria::EPS, 100, 1e-6));
```

Funktion palauttamat matriisit:

R = kameroiden välinen kierto matriisi

T = kameroiden välinen kääntö matriisi

E = sisältää tiedon kierosta ja käännöstä kameroiden välillä fyysisessä maailmassa

F = sama kuin edellinen, mutta huomioidaan kameroiden kamera matriisi ja tieto on pikseli muodossa

Funktion ohjausparametrit (flags)

- CV_CALIB_FIX_INTRINSIC: käytetään jo olemassa olevia kamera matriisia ja vääristymä matriisia, eikä tehdä niihin muutoksia.
- CV_CALIB_USE_INTRINSIC_GUESS: pohja arvoiksi annettuja kamera ja vääristymä matriiseja voidaan päivittää stereoCalib-funktion toimesta
- CV_CALIB_FIX_PRINCIPAL_POINT: kamera matriisissa olevaa optista keskipistettä voidaan muokata kamera funktion toimesta
- CV_CALIB_FIX_FOCAL_LENGTH: kamera matriisin polttoväli aroa voidaan muokata.
- CV_CALIB_FIX_ASPECT_RATIO: optimoidaan Fy ja Fx/Fy arvoja.
- CV_CALIB_SAME_FOCAL_LENGTH: pakotetaan kameroiden polttoväli kamera matriisissa samaksi, vaikka se pohjatietoina olevissa kamera matriiseissa olisi poikkeava
- CV_CALIB_ZERO_TANGENT_DIST asetetaan tangenttiaali vääristymä nollaksi ja korjataan se.
- CV_CALIB_FIX_K1,...,6: K1-K6 vääristymä matriisia ei muuteta, mutta jos CV_CALIB_USE_INTRINSIC_GUESS on asetettu käytetään annettua vääristymä matriisia, muussa tapauksessa, se asetetaan nollaksi.
- CV_CALIB_RATIONAL_MODEL: jos EI asetettu palautta ainoastaan 5 vääristymä kerrointa. Jos taas asetettu palautetaan kaikki 8 parametria. Parametrin avulla taataan taaksepäin yhteensopivuus.

5.2.7 Remap

Remap operaatiolla korjataan alkuperäinen kuva, huomioiden aikaisempien operaatioiden tulokset. Remap operaatio pitää myös suorittaa molemmille kuville erikseen.

```
remap(
framel, //in,alkuperäinen kuva
framel_rect, //out, korjattu kuva
map_l1, //in, InitUndistortRectifyMap operaatiosta saatu map1
map_l2, //in, InitUndistortRectifyMap operaatiosta saatu map1
INTER_LINEAR); // ohjaus parametrit
```

Ohjausparametrit:

- interpolation: interpolaatio metodi.
- borderMode: määrittää miten käsitellään lähdekuvan pisteitä, jotka jäävä lopullisen kuvan ulkopuolelle

- borderValue: käytetään, jos borderMode on CONSTANT, value määrittää millä arvolla border täytetään, oletusarvo on nolla eli musta.

5.2.8 StereoRectify

StereoRectify operaatiolla kuvapari kohdistetaan vaakasuunnassa samaan kohtaan. Eli jokin kuvapiste (ei kuvapikseli, vaan todellisen maailman piste esimerkiksi pöydän reuna, jalkalista) vasemmassa kuvassa on yhtä kaukana kuvan keskustasta vaakasuunnassa molemmissa kuvissa. Funktio huomioi aikaisempien ohjelmien tuottamat kamera ja vääristymämatriisit.

```
stereoRectify(l_cameraMatrix, //in, vasemmankameran matriisi
              l_distCoeffs, //in, vasemmankameran vääristymä matriisi
              r_cameraMatrix, //in, oikeankameran matriisi
              r_distCoeffs, //in, oikeankameran vääristymä matriisi
              image_size, //in, kuvankoko
              R, //in, kameroiden välinen kierto matriisi
              T, //in, kameroiden välinen kääntö matriisi
              Rl, //out, vasemman kameran kierto matriisi
              Rr, //out, oikean kameran kierto matriisi
              Pl, //out, vasemman kameran korjattu koordinaatio
              matriisi
              Pr, //out, oikean kameran korjattu koordinaatio
              Q); //out etäisyys matriisi
```

Funktiossa on lisäksi useita ohjausparametreja:

- Flags: 0 tai CV_CALIB_ZERO_DISPARITY, jos asetettu pyritään kuvat siirtämään, niin että niillä samat pikseli koordinaatit korjatussa kuvassa, jos ei asetettu kuvaa voidaan siitä huolimatta siirtää joko pysty-tai vaakasuunnassa riippuen epipolar viivojen suunnasta.
- Alpha: arvolla -1 on käytössä oletus skaalaus, arvolla 0 kuvat siirretään ja zoomataan, niin, että näkyvissä on vain kuvien validit pisteet eli kuvasta otetaan vain se osa mikä yhtenevä molemmissa kuvissa. Ja arvolla 1, kuvat siirretään niin, että kaikki pisteet ovat mukana kuvissa, jolloin kuvien reunoille jää mustaa kehystä.

- validPixROI1, validPixROI2: suorakulmio, jonka sisässä on validit pisteet. Paluuarvo ilmoittaa kuvan koon ja mistä pisteestä se alkuperäisessä kuvassa lähtee eli esimerkiksi [321 x 183 from (0,25)] tarkoittaa, että 321 pistettä leveä ja 183 pistettä korkea suorakulmio piirretään alkamaan pisteestä 0,25. Kuvan vasen yläkulma on piste 0,0.

5.2.9 InitUndistorRectifyMap

InitUndistortRectifyMap-operaatiolla lasketaan oikaisu- ja korjausmatriiseja kuvalle. Paluumatriisit map1 ja map2 ovat suoria syötteitä remap() operaatiota varten. Sekä vasemmalle että oikealle kuvalle suoritetaan oma laskentansa.

```
initUndistortRectifyMap(
l_cameraMatrix, //in, kameramatriisi
l_distCoeffs, //in, vääristymä matriisi
Rl, //in, stereoRectify operaatiolla saatu kierto matriisi
Pl, //in, stereoRectify korjattu koordinaatio matriisi
image_size, //in, kuvan koko
CV_16SC2, //in, haluttu map_l1 tyyppi, CV_32FC1 or CV_16SC2
map_l1, //out, mapX jokaisen pisteen X koordinaatti
map_l2); //out, mapY jokaisen pisteen Y koordinaatti
```

Tällä laskennalla nopeutetaan remap-operaatiota.

5.2.10 Vconcat/hconcat

vconcat/hconcat operaatiolla voidaan yhdistää kuvia yhdeksi kuvaksi. Alla olevalla yhdistetään vasen ja oikea kuva uudeksi kuvaksi (combo).

```
hconcat(framel_rect,framer_rect,combo);
```

5.2.11 SURF

SURF on openCV luokka, jonka avulla voidaan kuvasta etsiä kohteita

Etsii mielenkiintoiset piirteet kuvasta:

```
Ptr<SURF> Surf = SURF::create (3000);
// Detection of the SURF features
Surf->detect(image1_blur,keyPoints1);
Surf->detect(image2_blur,keyPoints2);
```

Määrittää tarkemman sijainnin ja skaalauskertoimen edellä etsityille pisteille

```
Ptr<SURF> SurfDesc = SURF::create();
cv::Mat descrip1, descrip2;
SurfDesc->compute(image1_blur,keyPoints1,descrip1);
SurfDesc->compute(image2_blur,keyPoints2,descrip2);
```

Eli SURF operaation avulla saadaan kuvasta etsittyä kiinnostavia kohteita ja niiden paikka. Tämä tehdään molemmille kuville erikseen. Näillä ei vielä stereokuvauksessa juurikaan olisi käyttöä, mutta kun nämä tiedot välitetään seuraavalle BFMatcer luokalle, saadaan kuvista selville kiinnostavia kohteita, joiden etäisyys voidaan laskea.

SURF luokan kaltaisia jonkin laskenta-algoritmin toteuttavia luokkia on muitakin, kuten ORB, HARRIS, SIFT. Aikapulan vuoksi en ehtinyt vertaamaan olisiko jokin muu algoritmi tuottanut paremman lopputuloksen. Toiseksi SURF on siitä mukava, että se pystyy toteuttamaan molemmat tehtävät, kun taas esimerkiksi BRIEF voidaan käyttää ainoastaan jälkimmäiseen.

5.2.12 BFMatcher

BFMatcer luokan avulla yhdistetään edellisten (SURF) operaatioiden tuotoksia. Matcer käy läpi descrip1 listan pyrkien löytämään jokaiselle vastaavuuden descrip2 listasta. Kun vastaavuus löytyy, se lisätään Matches listaan.

```
cv::BFMatcher Matcher( cv::NORM_L2, false );
  std::vector< cv::DMatch > Matches;
  Matcher.match( descrip1, descrip2, Matches );
```

5.2.13 StereoSGBM

StereoSGBM laskee kahden kuvan välisen ero kartan, laskentaan vaikuttaa monta parametriä ja niiden vaikutus voi olla todella iso, jos parametrit ovat riittävästi pielessä, ero kartta on täyttä nolla tai ”ääretöntä”.

Algoritmin alustus, koodi:

```
Ptr<StereoSGBM> stereo = StereoSGBM::create(0,16,3);

stereo->setPreFilterCap(45);
stereo->setBlockSize(3);
stereo->setUniquenessRatio(10);
stereo->setSpeckleWindowSize(100);
stereo->setSpeckleRange(32);
stereo->setDisp12MaxDiff(1);
stereo->setMode(StereoSGBM::MODE_SGBM);
```

OpenCV:

```
static Ptr<StereoSGBM> cv::StereoSGBM::create    (
            int         minDisparity,
            int         numDisparities,
            int         blockSize,
            int         P1 = 0,
            int         P2 = 0,
            int         disp12MaxDiff = 0,
            int         preFilterCap = 0,
            int         uniquenessRatio = 0,
            int         speckleWindowSize = 0,
            int         speckleRange = 0,
            int         mode = StereoSGBM::MODE_SGBM
    )
```

Luokan ohjaukset:

- minDisparity, pienin mahdollinen eroarvo yleensä 0
- numDisparitities suurin ero arvo – pienin ero arvo, jaollinen 16 ja aina suurempi kuin nolla.
- blockSize, lohkon koko >= 1 ja pariton, funktio etsii lohkoja ei yksittäisiä pikseleitä, mutta jos blockSize arvoksi laitetaan 1 niin silloin lohko on yhden pikselin kokoinen.
- P1, ohjaa erojen tasaisuuta
- P2, ohjaa myös erojen tasaisuutta ja P2 > P1 aina

  Löysin useita eri lähteistä ohjeelliset arvot P1 ja P2:lle, jotka lasketaan seuraavasti:

  P1= 8 * number_of_image_channels * BlockSize*BlockSize

  P2= 32 * number_of_image_channels * BlockSize*BlockSize

- disp12MaxDiff suurin sallittu ero pikseleinä, asetetaan -1 jos halutta ettei tarkistusta tehdä

- preFilterCap, rajoitetaan x-derivaatta välille [-preFilterCap – preFilterCap]
- uniquenessRatio, arvo välillä 5-15 yleensä sopiva
- speckleWindowSize, kohinanpoistoa, 0=ei käytössä, jos halutaan käyttää aseta arvo välille 50-200
- speckleRange, suurin ero vaihtelu jokaisessa komponentissa, 0 ei käytössä, 1 tai 2 yleensä sopivia arvoja
- mode oletuksena SGBM, joka käyttää vain viittä kahdeksasta suunnasta, aseta MODE_HH, jos halutaan käyttää täydet 8 suuntaa.

5.2.14 ReprojectImageTo3D

ReprojectImageTo3D operaatiolla lasketaan todelliseen maailman koordinaatit kuvalle. Eli joka pisteellä on X, Y ja Z koordinaatti.

```
reprojectImageTo3D(disp_compute, //in, disparity matriisi
                   pointcloud,  //out, 3D kuva
                   Q,//in, perspektiivi matriisi saadaan
                   stereoRectify()- operaatio tuloksena.
                   true); //in, kun asetettu virheelliset/puuttuvat
                   arvot korvataan arvolla 10000.
```

6 ETÄISYYDEN MITTAAMINEN

Seuraavissa kappaleissa käydään vaihe vaiheelta läpi se, miten stereokamerat saadaan kalibroitua, jotta kameroilla saadaan etäisyys mitattua. Ohjelmakoodit ovat enemmän vähemmän muokattu Practical OpenCV (Brahmbhatt, 2013) kirjan esimerkkien pohjalta.

Etäisyyden selvittämiseksi kirjassa Learning OpenCV (Kaehler, Learning OpenCV, 2008) on esitetty seuraava kaava:.

$$Z = fT/(d - (cx\text{left} - cx\text{right})).$$

Kaavan mukaan etäisyys saadaan selville, kun tiedetään kameroiden välinen etäisyys (T), kameroiden polttoväli (f) sekä ero kuvien(d) välillä. Cx on kameroiden paino keskipiste (principal point), joka on eri asia kuin kuvan keskipiste.

Samassa kirjassa on havainnollinen kuva (Kuva 12) siitä miten ero kuvien (disparity) välillä kertoo kohteen etäisyyden.

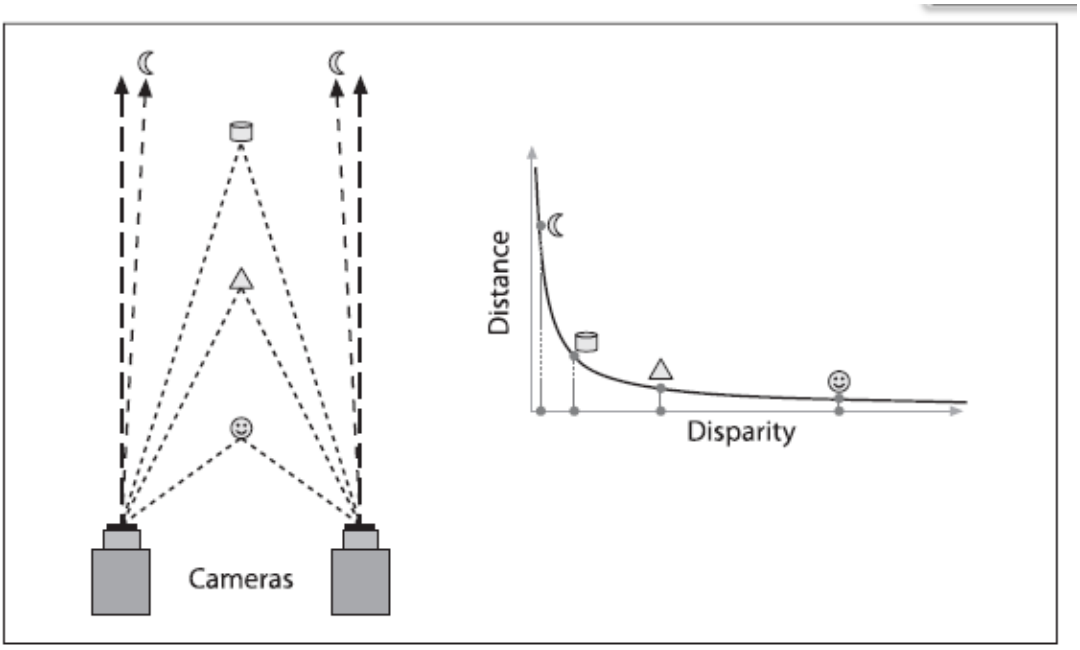


Figure 12-5. Depth and disparity are inversely related, so fine-grain depth measurement is restricted to nearby objects

Kuva 12 Kuvan kirjasta Learning OpenCV

Toinen esitys etäisyydelle on $d = \frac{f\ x\ B}{z}$, jossa d on ero (disparity) pikseleinä, f on polttoväli pikseleinä, B on baseline eli kameroiden välinen matka (esim. millimetreinä) ja z on etäisyys (millimetreinä). Näin esitettynä asia tuntuu kovin helpolta, mutta vaikeus onkin saada ratkaistua mikä on tuo d eli jonkin kohteen ero kuvien välillä pikseleinä. Siinä pitää huomioida vielä kameroiden aiheuttama virhe.

6.1 Huomioitavaa ennen kalibrointiin ryhtymistä

6.1.1 Valaistus

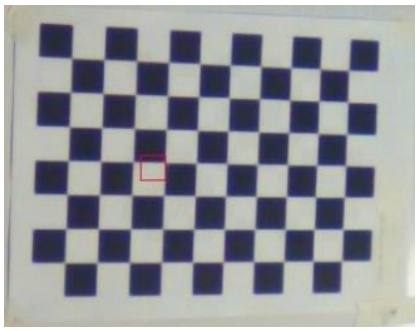
On tärkeää, että valaistus on riittävä, ja tasainen eikä pistemäisiä heijastuksia saa esiintyä. Minulla oli ongelmana erittäin kirkas kattovalo, joka aiheutti välillä tuon pistemäisen heijastuksen kuviin, jolloin kuva saattoi palaa puhki jostakin kohtaa shakkilautaa eli kuvaa mustavalkoiseksi muunnettaessa, jokin shakkilaudan osa muuttui kokonaan valkoiseksi, tällöin kulmien etsintä kuvasta ei onnistunut. USB-kameroiden kanssa oli valaistus vielä suuremmassa roolissa, kuin e-com kameran kanssa, jos valoa oli vähän (suurin osa kalibroinneista tehtiin vuoden pimeään aikaan ja iltaisin) kuviin tuli selvästi lisää kohinaa, joka taas huononsi lopputulosta. Samoin valon pitäisi olla tasaista kaikissa kalibrointiin käytetyissä kuvissa. Huomasin, että jos osa kuvista oli tummempia, kun toiset, se kasvatti RMS error arvoa (**reprojection error** (wikipedia, RMS, 2016)) verrattuna siihen jos kaikissa kuvissa oli tasainen valaistus. Tämä tuli esille kun osan kuvista aurinko valaisi ikkunan kautta ja osan aikaa aurinko oli pilvessä.

6.1.2 Kameran kiinnitys ja tarkennus

Koska kalibrointi on tarkoitus tehdä vai kerran, niin kameroiden tulisi olla kiinnitettynä, niin että eivät liiku kalibroinnin jälkeen, koska silloin kalibroinnin korjauskertoimet saattavat säätää kuvaa väärin. Esimerkiksi robottikäytössä ongelmaksi voi tulla se, että kamerat kolhaisevat johonkin esteeseen, niin sen jälkeen kalibrointi pitäisi suorittaa aina uudelleen. Samoin kameroiden tarkennusrengas tulisi olla lukittava, jos tarkennus vaihtuu niin taas olisi kalibrointi suoritettava uudelleen.

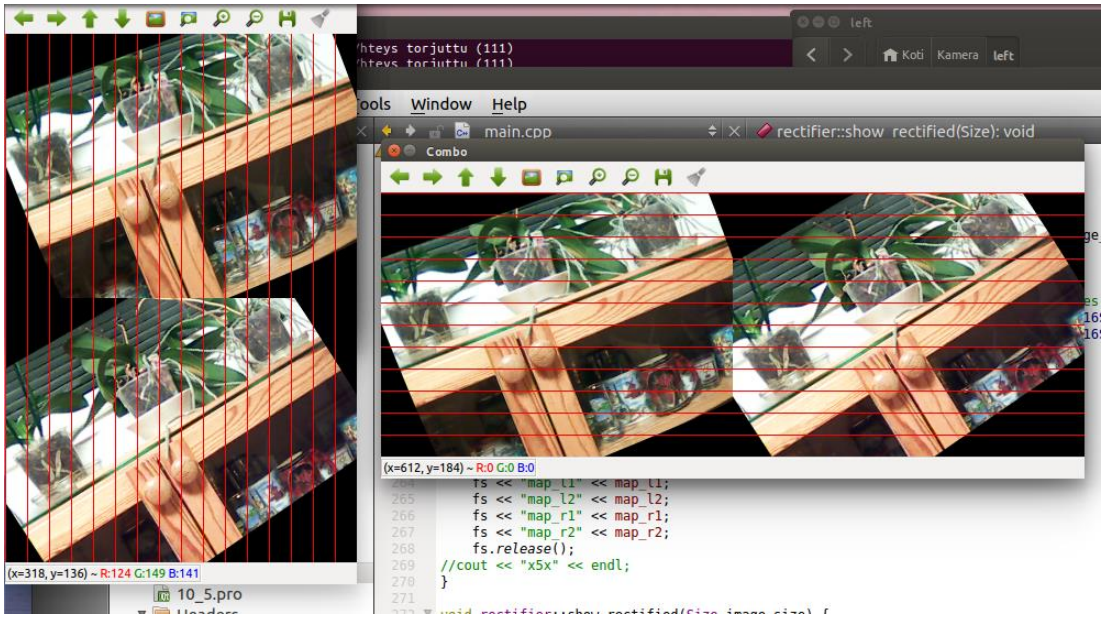
6.1.3 Shakkilauta

Todennäköisesti suurimmat virheet alussa minulla oli juuri shakkilaudan kanssa. Se oli väärän kokoinen ja väärin kiinnitetty. Shakkilaudan pitäisi olla mahdollisimman iso ja sisältää mahdollisimman paljon ruutuja sekä olla epäsymmetrinen. Tulostuksen ääriviivat tarkkarajaisia. Huonolaatuinen paperi ja mustesuihku, ei välttämättä ole paras mahdollinen ratkaisu shakkikuvion tulostukseen. Kuva 13 on kuva käyttämästäni shakkikuviosta, USB-kameralla kuvattuna, siinä on nähtävissä, että kameran piirto ei ole kovin tarkka ja ruutujen reunat ovat jo hivenen ”pehmentyneet”.



Kuva 13 Shakkikuvio

Alussa käytin symmetristä 5x5 shakkilautaa, joka oli kiinni ohuessa pahvin palassa, tämä yhdistelmä aiheutti välillä ongelmia kalibroinnin kanssa. Kalibroinnin jälkeen kuvat kääntyivät melko erikoiseen asentoon (Kuva 14).



Kuva 14 Epäonnistunut kalibrointi

Näiden kuvien jälkeen aloin metsästää tietoa, siitä mitä tein väärin. Lopulta netistä löysin keskustelun stackoverflow-keskustelu palstalta, missä käsiteltiin samaa ongelmaa, yhdeksi ratkaisuksi annettiin käyttää epäsymmetristä shakkilautaa. (stackoverflow, findChessboardCorners, 2013). Samalta keskustelupalstalta löytyi myös toinen keskustelu, jossa oli hyviä käytännön kokemuksia kalibroinnista. Tästä jälkimmäisestä keskustelusta löytyi myös vihje kiinnittää shakkilauta kuvio mahdollisimman hyvin jäykälle alustalle (stackoverflow, calibration of a webcam, 2016). Tämän vinkin pohjalta tulostin 10x7 shakkilaudan ja kiinnitin sen fläppitauluun teipillä mahdollisimman kireälle. Aikaisemmin minulla oli vain pahvinpalaan kiinnitetty A4 arkki, joka kupruili. Näillä vihjeillä RMS arvo putosi huomattavasti arvosta yli viisi alle yhden.

6.2 Kalibrointi

Kalibroinnin tarkoitus on korjata kameran virheitä ja sovittaa kamera 3D koordinaatistoon. Kun tiedetään shakkilaudan ruutujen määrä ja koko, niin pystytään laskemaan yksittäiselle kameralle ” intrinsic parameters” (kameran polttoväli, optinen keskipiste) ja ” extrinsic parameters” (kierto, kääntö) näillä sovitetaan koordinaatisto 3D koordinaatistoon. Stereokalibroinnissa taas sovitetaan kameroiden koordinaatistot yhteen siten, että niiden avulla on mahdollista laskea jonkin kohteen etäisyys.
Kalibroinnin työvaiheet:

- Otetaan kuvia shakkilaudasta, molemmilla kameroilla erikseen
- Kalibroidaan yksittäiset kamerat
- Samoja kuvia tai uusia käyttäen stereokalibrointi eli kalibroidaan molemmat kamerat yhdessä.

Kun nämä vaiheet on saatu onnistuneesti läpi, voidaan aloittaa varsinainen kuvaaminen ja etäisyyksien mittaaminen. Kalibroinnin onnistumista kuvataan RMS error arvolla, joka pitäisi saada mahdollisimman lähelle nollaa. Kalibrointiprosessin alussa arvot olivat USB-kameroilla luokkaa 10 ja yli, lopulta päästiin luokkaan 0,7 - 0,5. E-COM kameralla pääsin 0,17 lukemiin, kun käytettiin täyttä kuvatarkkuutta.

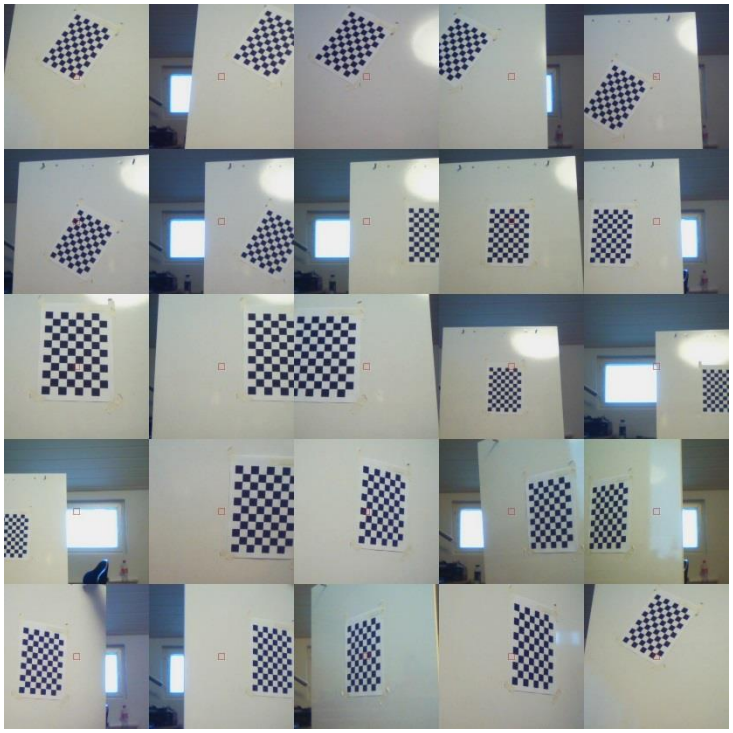
6.2.1 Konfiguraatiotiedosto

Kaikki liitteiden 1-5 ohjelmat käyttävät samaa ohjaustiedostoa. Sen avulla on helppo ohjata kalibrointiparametritiedostot ja kuvatiedostot oikeaan paikkaan. Config.h tiedostosisältö:

```
#define DATA_FOLDER "/home/udooer/data"
#define TRAIN_FOLDER "@TRAIN_FOLDER@"
#define TEMPLATE_FOLDER "@TEMPLATE_FOLDER@"
#define LEFT_FOLDER "/home/udooer/left/"
#define RIGHT_FOLDER "/home/udooer/right/"
```

6.2.2 Kuvien kerääminen kalibrointia varten

Aluksi on otettava kuvia shakkilaudasta mahdollisimman monipuolisesti, jotta todellinen loppu tulos olisi hyvä. Vain parilla kuvalla voi päästä hyvää RMS arvoon, mutta siitä huolimatta lopullinen etäisyyden määrittäminen ei välttämättä toimi kuin osalla kuva-alaa. Kuvia siis pitää ottaa toistakymmentä. Ja kuvissa on tärkeää, että shakkilauta on kuvissa eri kohdissa ja eri kulmissa. Kuvassa Kuva 15 on esimerkkisarja toisen kameran shakkilautakuvista.



Kuva 15 Kalibrointi kuvasarja

Shakkilaudan pitää näkyä molemmissa kuvissa kokonaisena, jos samoja kuvia on tarkoitus käyttää myös stereokalibrointiin. Samaan hakemistoon ei saa jäädä eri kuvasuhteella otettuja kuvia tai kalibrointi ei onnistu. Liitteessä 1 olevalla ohjelmalla kuvat tallentuvat omiin kansioihinsa numeroiden tiedostot kuvaus järjestyksessä. Kansioiden polku määritellään config.h tiedoston parametreilla LEFT_FOLDER ja RIGHT_FOLDER. Ennen ohjelman kääntämistä, pitää selvittää kameroiden liitäntäpisteet ja päivittää ne Lkamera ja Rkamera parametreihin, samoin pitää kuvankoko määritellä parametreihin VAAKA ja PYSTY (esimerkiksi 320/240). Kun ohjelma käynnistyy, ruudulle tulee kaksi ikkunaa, joissa on kameroiden kuvat. Painamalla ’C’ näppäintä tallentuu kuva tiedostoon molemmista kameroista samanaikaisesti. Kun kuvia on otettu riittävä määrä, painetaan ’Q’ näppäintä ja ohjelma loppuu.

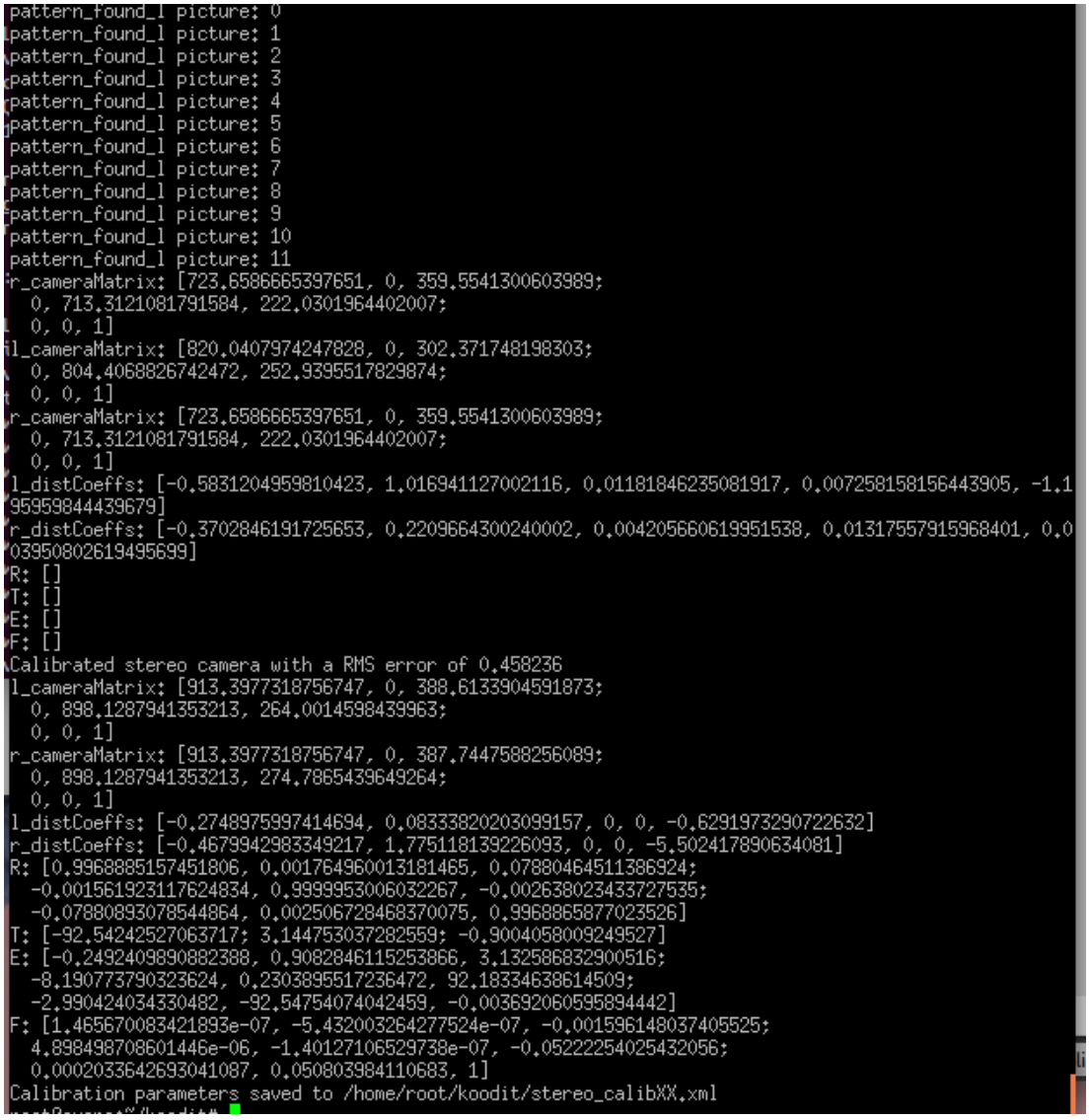
6.2.3 Kameran kalibrointi

Liitteessä 2 esitetyllä ohjelmalla suoritetaan molempien kameroiden kalibrointi erikseen. Molemmat kamerat kuitenkin kalibroidaan kerralla ja kameroiden parametrit talletetaan omiin tiedostoihinsa myöhempää käyttöä varten. Hakemisto on määritelty config.h tiedostossa DATA_FOLDER parametrilla. Ohjelma lukee kuvatiedostot, jotka tallennettiin edellisellä ohjelmalla, suorittaa shakkilautakuvion etsinnän, risteyskohtien tarkennuksen ja tallettaa tulokset vektoriin, joka viedään kalibrointifunktiolle kamera kerrallaan. Lopuksi kalibroinnin tulos talletetaan kamerakohtaiseen tiedostoon. Jos suoritetaan useampia kalibrointeja eri kokoisilla shakkilautakuvioilla, niin ne kaikki voidaan tallentaa samaan tiedostoon ja ne kaikki ovat käytettävissä myöhemmin. Ohjelma ilmoittaa näytöllä kamerakohtaisesti RMS arvon, jonka siis pitäisi olla mahdollisimman pieni.

6.2.4 Kameroiden stereokalibrointi

Liitteen 3 ohjelma suorittaa kameroiden stereokalibroinnin. Kalibroinnin pohjaksi otetaan edellisellä ohjelmalla suoritetut kameran kalibrointitiedostot. Ohjelman alussa luetaan kalibrointitiedostot. Seuraavaksi ohjelma käy läpi kuvatiedostot ja jos se löytää shakkilautakuvion se näyttää sen näytöllä, kunnes painetaan jotakin näppäintä, lopuksi

ohjelma tallentaa stereokalibrointitiedot omaan tiedostoonsa, ’DATA_FOLDER’ hakemistoon. Kuva 16 esittää kuvakaappauksen ohjelman ajosta. Kuvasta näkee myös kamera- ja korjausmatriisien sisältöä.



Kuva 16 Kalibrointimatriisit

6.2.5 Kuvien asemointi samalle kohdalle korkeussuhteessa

Liitteen 4 ohjelma kohdistaa kuvat samalle tasolle vaakasuunnassa. Eli kun kuvat asetetaan rinnakkain, niin jos piirretään viiva molempiin kuviin viivat pitäisi olla samassa kohdassa molemmissa kuvissa. Tämä ei tarkoita, että viiva olisi esimerkiksi keskellä kuva-alaa vaan sama viiva läpäisee saman reaalimaailman kohteen molemmissa kuvissa. Eli kuvat siirretään korkeussuhteessa samalle tasolle. Tästä johtuen kuvien ala- tai ylälaitaan saattaa jäädä musta osuus. Samalla kuvat käännetään niin, että viivat

ovat vaakasuorassa. Tämän jälkeen kuvista on helppo etsiä pisteitä, koska niiden pitäisi olla samassa kohtaa matriisia.

Tässä tehtävässä kuvat asemoidaan vierekkäin, matkien ihmisen silmiä funktiolla olisi mahdollista myös siirtää kuvat pystysuunnassa kohdakkain. Molempia ei voi tehdä, mikäli kuvat siirrettäisiin myös pystysuunnassa samalle kohdalle, niin menetettäisiin mahdollisuus etäisyyden mittaamiseen, koska kuvissa ei olisi laskemiseen vaadittavaan kulmapoikkeamaa. Itse ohjelman toiminta on yksinkertainen, ohjelma lukee edellisten ohjelmien tuottamat matriisit, huomio ne ja laskee kuvat uudelleen. Lopuksi kuvat esitetään kaksi rinnakkain ja päällekkäin, vaakasuuntaisessa kuvassa punaisten viivojen pitäisi kulkea saman pisteen kautta molemmissa kuvissa. Pystysuuntaisessa kuvassa taas pitäisi näkyä se, että kuvissa on pieni poikkeama, jota hyväksi käyttäen etäisyys tullaan määrittelemään.

6.3 Etäisyyden mittaaminen

Liitteessä 5 oleva ohjelma suorittaa varsinaisen mittauksen. Ohjelmassa on käytetty kahta eri tapaa etäisyyden mittaamiseen. Ensimmäisessä etsitään kuvasta kohteita ja määritellään niiden etäisyys ja toisessa otetaan alue kuvasta ja lasketaan keskiarvoistamalla alueen etäisyys. Lisäksi ohjelmassa on mahdollisuus haarukoida muutamalla parametrilla mahdollisimman hyvää lopputulosta. Tällä hetkellä ohjelma tulostaa etäisyydet kahteen ikkunaan ja konsolille. Jos ohjelma kuitenkin laitettaisiin robottiin, kuten suunniteltiin, niin tulosten näyttäminen ikkunoissa kannattaisi jättää pois hidastamasta toimintaa, sen jälkeen, kun on saatu haettua hyvät parametriarvot. Konsolitulostus olisi taas kohtalaisen helppo muuttaa käyttämään socket rajapintaa.

Ensimmäinen etäisyyden laskentametodi käyttää SURF Descriptor + BFMatcher algoritmia, näiden avulla saadaan joukko kiinnostavia pisteitä. Saatu lista järjestetään ja siitä otetaan 10 lähimpänä olevaa pistettä, jotka piirretään kuvaan POINTS. Kuvaan piirretään pieni ympyrä kohteen ympärille ja viereen kirjoitetaan etäisyys linssi pinnasta ja kulma suhteessa kuvan keskipisteeseen.

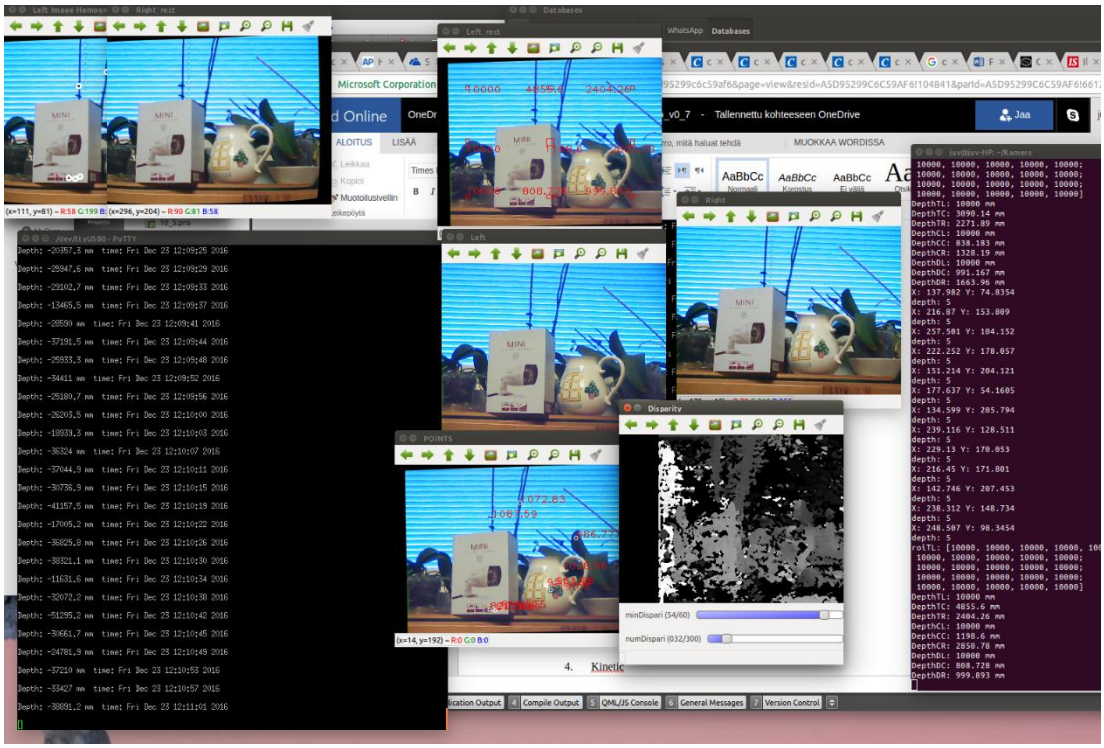
Toisessa mittaustavassa kuvasta on määritelty 9 aluetta (3x3 ruutua), joiden sisältä lasketaan keskiarvo etäisyys ja tulostetaan se näytölle. Tässä tavassa oli ongelmana se, että 3D projektio sisälsi paljon kohtia, joilla ei ollut onnistuttu laskemaan eroarvosta etäisyystietoa. Näihin ”virheellisiin” kohtiin ohjelma tallentaa arvon 10000, joka vääristi keskiarvoa välillä paljonkin ylöspäin. Lähimmäksi oikeaa päästiin, kun keskiarvo laskettaessa jätettiin 10000 arvot huomioimatta.

Pistemetodilla tuli helpommin käyttökelpoisia etäisyyksiä, mutta pisteiden paikka ja määrä vaihtelivat kuvasta toiseen voimakkaasti. Tästä vaihtelevasta tiedosta on vaikeampi tehdä ohjaustietoa robotille.

Jos lähtisin itse rakentamaan etäisyystietoa robotille, pyrkisin määrittelemään muutaman kiinteän alueen ja laskisin niistä keskiarvo periaatteella etäisyyden. Näiden lisäksi piste metodilla pyrkisin havaitsemaan lähempänä olevia pienempiä kohteita, eli pisteistä valittaisiin vain se joukko, joka on esimerkiksi alle metrin päässä ja tietyn kulman päässä kuvan keskilinjasta.

Ohjelma lukee käynnistyessään edellisten ohjelmien tuottamat matriisit kuvan korjaamiseksi, ottaa kuvat molemmilla kameroilla, suorittaa kuvien korjaukset huomioiden luetut matriisit ja liukusäätimillä asetetut arvot, laskee etäisyydet pisteille ja alueille, kirjoittaa etäisyys tiedot kuviin ja konsolille sekä piirtää kuvat näytölle. Ohjelma pyörii luupissa lukien uudet kuvat kameroilta, tehden korjaukset ja tulostaen kuvat näytölle. Ohjelman suoritus katkeaa painamalla ’Q’ näppäintä.

Kuva 17 näyttää ohjelman tulostamat ikkunat näytöllä, ikkuna missä näkyy liukusäätimet alalaidassa esittää kuvien välisen eron pohjalta väritetyn etäisyyskuvan, eli mitä vaaleampi väri niin sitä lähempänä kohteen pitäisi olla. Tuossa otoksessa säätöparametrit ei vielä olleet parhaassa mahdollisessa arvossaan, siitä johtuu erokuvan vasemman laidan voimakas valkoinen alue.



Kuva 17 Etäisyyden mittausohjelma käynnissä

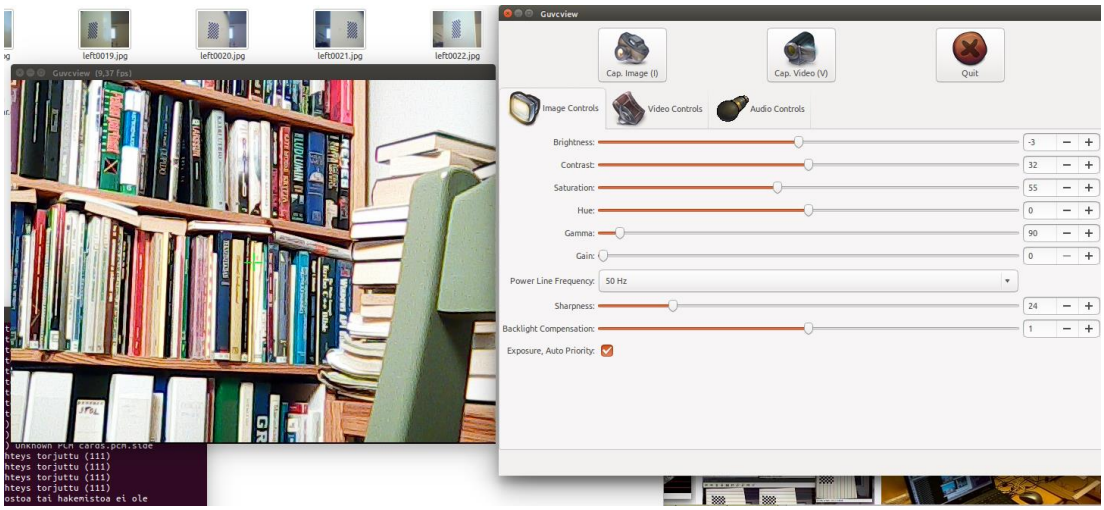
7 TYÖSSÄ KÄYTETYT APUOHJELMAT

7.1 Putty

Putty on hyvin yleisesti käytetty terminaaliemulaattori, sarjaliikennekonsoli ja tiedonsiirto-ohjelma. Tässä työssä käytin sitä terminaaliemulaattorina E-COM kameran kanssa. Käytin Putty ohjelmaa myös UDOO:n kanssa luoden sillä etäyhteyden, tällä tavalla toimittaessa ei tarvittu UDOO:lle erillistä näppäimistöä ja hiirtä. Netistä löytyy paljon lisätietoa, esimerkiksi https://fi.wikipedia.org/wiki/PuTTY

7.2 Guvcview

Guvcview ohjelma on freeware webkameraohjelma. Ohjelmassa näkyy kameran kuva ja toisessa ikkunassa liukusäätimet kuvan parametrien muuttamiseen. Ohjelma on erittäin havainnollinen, koska kuva muuttuu välittömästi, kun säätimiä liikuttaa. Kuva 18 alla on kuvakaappaus ohjelmasta käyttöpaneelista ja vieressä kameran kuva.



Kuva 18 Guvcview ohjelma

Alun perin latasin ohjelman koneelle, koska halusin varmistaa missä /dev/videoX ”portissa” kamerat milloinkin olivat kiinni. Kameroiden portti vaihtui lähes aina, kun kamerat irrotti koneesta ja laittoi uudelleen kiinni. Jos ohjelmaa ei halua tai voi asentaa, niin perus Linux-komennoillakin voi selvittää missä kamerat ovat ja mitkä ovat niiden ominaisuudet. Komennolla ls -l /dev/video* tulostuu jokaista koneessa

kiinni olevaa kameraa kohden yksi rivi. Tulostelemalla tuota komentoa, kytkemällä ja/tai irrottamalla kameroita saa selville mikä kamera on kytketty mihinkin osoitteeseen. Sen jälkeen voi tarkemmin lsusb-komennolla tutkia, mitkä ovat kameran parametrit. Alla oleva Kuva 19 esittää komennot ja osan parametrilistaa.

```
            lsusb
Bus 002 Device 003: ID 0c45:6340 Microdia Camera
Bus 002 Device 002: ID 8087:0020 Intel Corp. Integrated Rate Matching Hub
Bus 002 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
Bus 004 Device 001: ID 1d6b:0003 Linux Foundation 3.0 root hub
Bus 003 Device 002: ID 0c45:6340 Microdia Camera
Bus 003 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
Bus 001 Device 005: ID 04f2:b15e Chicony Electronics Co., Ltd
Bus 001 Device 004: ID 138a:0007 Validity Sensors, Inc. VFS451 Fingerprint Reade
r
Bus 001 Device 003: ID 03f0:231d Hewlett-Packard Broadcom 2070 Bluetooth Combo
Bus 001 Device 002: ID 8087:0020 Intel Corp. Integrated Rate Matching Hub
Bus 001 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
juv@juv-HP:~$ lsusb -v -s002:003 |more
Couldn't open device, some information will be missing

Bus 002 Device 003: ID 0c45:6340 Microdia Camera
Device Descriptor:
  bLength                18
  bDescriptorType         1
  bcdUSB               2.00
  bDeviceClass          239 Miscellaneous Device
  bDeviceSubClass         2 ?
  bDeviceProtocol         1 Interface Association
  bMaxPacketSize0        64
  idVendor           0x0c45 Microdia
  idProduct          0x6340 Camera
  bcdDevice            0.00
  iManufacturer           2
  iProduct                1
  iSerial                 0
  bNumConfigurations      1
  Configuration Descriptor:
    bLength                 9
    bDescriptorType         2
    wTotalLength          671
    bNumInterfaces          4
    bConfigurationValue     1
    iConfiguration          0
    bmAttributes         0x80
      (Bus Powered)
    MaxPower              500mA
    Interface Association:
      bLength                 8
      bDescriptorType        11
      bFirstInterface         0
      bInterfaceCount         2
      bFunctionClass         14 Video
      bFunctionSubClass       3 Video Interface Collection
      bFunctionProtocol       0
      iFunction               5
    Interface Descriptor:
      bLength                 9
      bDescriptorType         4
      bInterfaceNumber        0
      bAlternateSetting       0
      bNumEndpoints           1
      bInterfaceClass        14 Video
      bInterfaceSubClass      1 Video Control
      bInterfaceProtocol      0
      iInterface              5
      VideoControl Interface Descriptor:
        bLength                13
        bDescriptorType        36
        bDescriptorSubtype      1 (HEADER)
        bcdUVC               1.00
        wTotalLength           77
        dwClockFrequency       15.000000MHz
```

Kuva 19 lsusb komento

8 LOPPUPÄÄTELMÄT

Kaiken kaikkiaan työ oli mielenkiintoinen ja haastava. Tutkittavaa ja kokeiltavaa olisi ollut vielä moneksi illaksi ja viikonlopuksi. UDOO + USB kanssa tuli todettua, että tehtävä on mahdollinen, vähän paremmat USB kamerat, hyvä kiinnitys, lisävalo ja tarkka kameroiden kalibrointi, niin varmasti onnistuu kohteiden havainnointi ja etäisyyksien arviointi. E-COM kameran kanssa olisi aikaa vielä kulunut ohjelmien viritttelyyn ja Capellan omien openCV lisäyksien tutkimiseen, olisiko sieltä löytynyt ratkaisu millä kameran rajallinen laskentateho olisi saatu riittämään. Kinect-ohjainta oikeastaan vain pinta raapaistiin, mutta jo se kertoi, että sillä etäisyyden saaminen olisi varminta, mutta iso kysymysmerkki jäi siitä, miten ohjelmiston olisi saanut rakennettua.

Kuten jo aikaisemmin totesin, etäisyyden esittäminen kameroiden avulla robotille on mahdollista ja toivottavasti tämä raportti olisi hyvä pohja jollekin seuraavalle opinnäytetyölle ja robotille lopulta saataisiin ”silmät”.

Olen tähän opinnäytetyöhön kirjannut mahdollisimman paljon niitä kohtia, jossa minulla oli ongelmia ja mistä tietoa sain etsittyä. Yksi ongelma tiedon metsästyksessä on se, että tietoa on paljon ja monet ovat jo aika vanhoja ja tietojen päivittäminen on unohtunut ja siksi eivätkä enää päde uusimpien ohjelma versioiden kanssa. Ajatus oli tarjota pohjaa seuraavalle opinnäytetyölle, jotta seuraavan ei tarvitsisi kuluttaa aikaa perusasioiden kaivamiseen, vaan pääsi nopeammin varsinaiseen ongelman ratkaisuun ja robotti vihdoin osaisi huomioida esteet liikkuessaan.

LÄHTEET

Aalto, yliopisto. (2004). *Fotogrammetrian perusteet*. Haettu 01 2017 osoitteesta https://foto.aalto.fi/opetus/300/luennot/2/2.html

Brahmbhatt, S. (2013). *Practical OpenCV.* Apress. Retrieved 10 2016

Clas Ohlson. (2016). *Web-kamera*. Noudettu osoitteesta Clas Ohlson verkkokauppa: http://www.clasohlson.com/fi/Web-kamera/Pr384754000

e-con systems. (n.d.). *Capella - Stereo Vision Camera Reference Design*. (e-con Systems) Retrieved 03 10, 2016, from e-con Systems: https://www.e-consystems.com/stereo-vision-camera.asp

e-con Systems India Pvt Ltd. (2012). Capella Product Brief. Chennai.

GitHub, OpenCV2Cookbook. (4. 1 2012). *laganiere/OpenCV2Cookbook* . Noudettu osoitteesta github: https://github.com/laganiere/OpenCV2Cookbook.

GitHub, practical opencv. (2016). *Apress/practical-opencv*. Retrieved from GitHub: https://github.com/apress/practical-opencv

ifixit. (2016). *Xbox 360 Kinect Teardown*. Haettu 14. 01 2017 osoitteesta www.ifixit.com: https://www.ifixit.com/Teardown/Microsoft-Kinect-Teardown/4066/

Kaehler, G. B. (2008). Learning OpenCV. Teoksessa G. B. Kaehler, *Learning OpenCV.* o'Reilly.

Kaehler, G. B. (2008). *Learning OpenCV.* Sebastopol: Published by O'Reilly Media, Inc.,.

kdab group. (17. 04 2012). *Setting up Kinect for programming in Linux (part 1)*. Noudettu osoitteesta www.kdab.com: https://www.kdab.com/setting-up-kinect-for-programming-in-linux-part-1/

Laganière, R. (2011). OpenCV 2 Computer Vision Application. Teoksessa R. Laganière, *OpenCV 2 Computer Vision Application.* Birmingham: Published by Packt Publishing Ltd.

MathWorks, Inc. (1994-2017). *MATLAB.* Retrieved from mathworks.com: https://se.mathworks.com/products/matlab.html

open kinect. (9. November 2011). *Open Kinect Main page fi*. Haettu 14. January 2017 osoitteesta Open Kinect: https://openkinect.org/wiki/Main_Page/fi

opencv.org. (2016). *opencv*. Haettu 6. 12 2016 osoitteesta opencv.org: http://opencv.org/

opencv.org. (2017). *About*. Haettu 04. 12 2016 osoitteesta opencv: http://opencv.org/about.html

stackoverflow, calibration of a webcam. (16. 4 2016). *How to verify the correctness of calibration of a webcam?* Noudettu osoitteesta stackoverflow: http://stackoverflow.com/questions/12794876/how-to-verify-the-correctness-of-calibration-of-a-webcam

stackoverflow, findChessboardCorners. (09 2013). *findChessboardCorners fails for calibration image*. Noudettu osoitteesta stackoverflow: http://stackoverflow.com/questions/17665912/findchessboardcorners-fails-for-calibration-image

UDOO.ORG. (2016). *UDOO QUAD/DUAL*. Haettu 05. 12 2016 osoitteesta http://www.udoo.org/udoo-dual-and-quad/

wikipedia, Kinect. (2015). *Kinect*. Haettu 06. 12 2016 osoitteesta Wikipedia: https://fi.wikipedia.org/wiki/Kinect

wikipedia, RMS. (28. Jan 2016). *Reprojection error*. Haettu 09. 09 2016 osoitteesta wikipedia: https://en.wikipedia.org/wiki/Reprojection_error

LIITE 1

```
// Program to collect stereo snapshots for calibration
// Author: Samarth Manoj Brahmbhatt, University of Pennsylvania
// kirjan esimerkki 10_3
// JuV 2016, päivitys OpenCV 2.x -> 3.x ja boost kirjaston poisto

#include <opencv2/opencv.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <opencv2/videoio.hpp>
#include "config.h"
#include <iomanip>
#include <QString>

#define  Lkamera
#define  Rkamera
#define  VAAKA 320 // 640
#define  PYSTY 240 // 480

using namespace cv; using namespace std;

int main() {
  // tälle riville pitää päivittää kameroiden liitännät, jotka voi varmistaa
  // esim. guicview-ohjelmalla
  VideoCapture capr(Rkamera), capl(Lkamera);


  // kuvan kooksi piti valita vähän pienempi,
  // että USB kaistanleveys riittää
  capl.set(CV_CAP_PROP_FRAME_HEIGHT, VAAKA);
  capl.set(CV_CAP_PROP_FRAME_WIDTH, PYSTY);
  capr.set(CV_CAP_PROP_FRAME_HEIGHT, VAAKA);
  capr.set(CV_CAP_PROP_FRAME_WIDTH, PYSTY);

  // ikkunat kuvia varten
  namedWindow("Left");
  namedWindow("Right");
  moveWindow("Left",50,50);
  moveWindow("Right",400,50);
  namedWindow("Left",WINDOW_AUTOSIZE);
  namedWindow("Right",WINDOW_AUTOSIZE);

  cout << "Press 'c' to capture ..." << endl;

  char choice = 'z';
  int count = 0;

  while(choice != 'q')
  {
    //grab frames quickly in succession
    // ihan saman aikaisia kuvia ei saada, mutta aika lähelle päästään
    // kun kuvat otetaan peräkkäisillä riveillä
    capl.grab();
    capr.grab();

    //execute the heavier decoding operations
    Mat framel, framer;

    capl.retrieve(framel);
    capr.retrieve(framer);

    if(framel.empty() || framer.empty()) break;
    // piirretään kuvien keskelle 20x20 neliö
```

```
        int xmin = framel.cols/2 - 10,
          xmax = framel.cols/2 + 10,
          ymin = framel.rows/2 - 10,
          ymax = framel.rows/2 + 10;
        rectangle(framel, Point(xmin, ymin), Point(xmax, ymax), Scalar(0, 0, 255));
        rectangle(framer, Point(xmin, ymin), Point(xmax, ymax), Scalar(0, 0, 255));
        // ja tulostetaan kuvat näytölle
        imshow("Left", framel);
        imshow("Right", framer);

        if(choice == 'c')
        {
            //save files at proper locations if user presses 'c'
            stringstream l_name, r_name;

            l_name << "left" << setw(4) << setfill('0') << count << ".jpg";
            r_name << "right" << setw(4) << setfill('0') << count << ".jpg";
            // talletetaan kuvat tiedostoon
            imwrite(string(LEFT_FOLDER) + l_name.str(), framel);
            imwrite(string(RIGHT_FOLDER) + r_name.str(), framer);

            cout << "Saved set " << count << "Nimi: " << string(LEFT_FOLDER)
                 + l_name.str() << endl;
            count++;
        }
        choice = char(waitKey(1));
    }
    // vapautetaan kamerat muiden ohjelmien käyttöön
    capl.release();
    capr.release();
    return 0;
}
```

LIITE 2

```
// Program illustrate single camera calibration
// kirjan esimerkki 10_1
// Author: Samarth Manoj Brahmbhatt, University of Pennsylvania
// JuV 2016: päivitys opencv 2.x → 3.x, boost kirjaston poisto

#include <opencv2/opencv.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <opencv2/calib3d/calib3d.hpp>
#include <QDirIterator>
#include <QString>
#include "config.h"

#include RUUDUNKOKO 22.5f
#include VAAKA 7
#include PYSTY 10



using namespace cv;
using namespace std;
//using namespace boost::filesystem;

class calibrator
{
  private:
    QString path; //path of folder containing chessboard images
    vector<Mat> images; //chessboard images
    Mat cameraMatrix, distCoeffs; //camera matrix and distortion coefficients
    bool show_chess_corners; //visualize the extracted chessboard corners?
    float side_length; //side length of a chessboard square in mm
    int width, height; //number of internal corners of the chessboard
                       //a long width and height


    vector<vector<Point2f> > image_points; //2D image points
    vector<vector<Point3f> > object_points; //3D object points

  public:
    calibrator(QString, float, int, int); //constructor, reads in the images
    void calibrate(); //function to calibrate the camera
    Mat get_cameraMatrix(); //access the camera matrix
    Mat get_distCoeffs(); //access the distortion coefficients
    void calc_image_points(bool); //calculate internal corners
                                  // of the chessboard image
};

calibrator::calibrator(QString _path, float _side_length, int _width, int
_height){

side_length = _side_length;
width = _width;
height = _height;

path = _path;
cout << path.toStdString() << endl;
// Read images files
QDirIterator it(path);
while (it.hasNext()){
    QString filename = path + it.fileName();
    if (it.fileName().contains("jpg")){
```

```
            images.push_back(imread(filename.toStdString()));
        }
        it.next();
    }
    /* sama boost-kirjastolla jätetty esimerkin vuoksi
      for(directory_iterator i(path), end_iter; i != end_iter; i++){
        string filename = path + i->path().filename().string();
        cout << "READfile: " << filename << endl;
        images.push_back(imread(filename));
      } */
    }

    void calibrator::calc_image_points(bool show) {
    // Calculate the object points in the object
    // co-ordinate system (origin at top left corner)
    vector<Point3f> ob_p;

    for(int i = 0; i < height; i++){
        for(int j = 0; j < width; j++){
          ob_p.push_back(Point3f(j * side_length, i * side_length, 0.f));
        }
    }
    if(show) namedWindow("Chessboard corners");

    cout << "Pictures: " << images.size() << endl;
    cout << "Width: " << width << endl;
    cout << "Height: " << height << endl;

    for(int i = 0; i < (int)images.size();){
        cv::Mat im = images[i]; vector<Point2f> im_p;
        // seuraava funktion etsii kuvasta annetun kokoista shakkilauta kuviota
        bool pattern_found = findChessboardCorners(im,
                 Size(width, height), im_p, CALIB_CB_ADAPTIVE_THRESH +
                  CALIB_CB_NORMALIZE_IMAGE + CV_CALIB_CB_FILTER_QUADS);
                  // + CALIB_CB_FAST_CHECK);
        if(pattern_found)
        {
          cout << "pattern found picture: " << i << endl;
          object_points.push_back(ob_p);
          // tarkennetaan kulmien paikkaa
          Mat gray;
          cvtColor(im, gray, CV_BGR2GRAY);
          cornerSubPix(gray, im_p, Size(5,5), Size(-1, -1),
                 TermCriteria(CV_TERMCRIT_EPS + CV_TERMCRIT_ITER, 30, 0.1)); //0.1
          image_points.push_back(im_p);

          if(show)
          { //tulostetaan kuvat näytölle
            Mat im_show = im.clone();
            drawChessboardCorners(im_show, Size(width, height), im_p, true);
            imshow("Chessboard corners", im_show);
             while(char(waitKey(1)) != ' ') {}
          }i++;
        }else{
          cout << "pattern NOT found pictures: " << i << endl;
          //if a valid pattern was not found,
          //delete the entry from vector of images
          images.erase(images.begin() + i);
        }
      }
    }

    void calibrator::calibrate()
    {
      vector<Mat> rvecs, tvecs;
```

```
  if (image_points.size() >0){
    float rms_error = calibrateCamera(object_points, image_points,
images[0].size(), cameraMatrix,distCoeffs, rvecs, tvecs);
      cout << "RMS reprojection error " << rms_error << endl;
   }else  cout << "image points is empty" << endl;
}

Mat calibrator::get_cameraMatrix(){return cameraMatrix;}

Mat calibrator::get_distCoeffs(){return distCoeffs;
}

int main(int argc, char *argv[]){
  // oikea kamera
  calibrator calibR(DATA_FOLDER + QString("right/"), RUUDUNKOKO, VAAKA, PYSTY);
  calibR.calc_image_points(true);
  cout << "Calibrating RIGHT camera" << endl;
  calibR.calibrate();

  filename = DATA_FOLDER + string("cam_calibR.xml");
  cout << "FILE: " << filename << endl;

  FileStorage fsR(filename, FileStorage::WRITE);

  fsR << "cameraMatrix" << calibR.get_cameraMatrix();
  fsR << "distCoeffs" << calibR.get_distCoeffs();
  fsR.release();

  cout << "Saved calibration matrices to " << filename << endl;
   // vasen kamera
   calibrator calibR(DATA_FOLDER + QString("left/"), RUUDUNKOKO, VAAKA, PYSTY);
  calibL.calc_image_points(true);
  cout << "Calibrating LEFT camera" << endl;
  calibL.calibrate();

  filename = DATA_FOLDER + string("cam_calibL.xml");
  cout << "FILE: " << filename << endl;

  FileStorage fsL(filename, FileStorage::WRITE);

  fsL << "cameraMatrix" << calibL.get_cameraMatrix();
  fsL << "distCoeffs" << calibL.get_distCoeffs();
  fsL.release();

  cout << "Saved calibration matrices to " << filename << endl;

  return 0;
}
```

LIITE 3

```
// Program illustrate single camera calibration
// kirjan esimerkki 10_4
// Author: Samarth Manoj Brahmbhatt, University of Pennsylvania
// JuV 2016: päivitys opencv 2.x → 3.x, boost kirjaston poisto

#include <vector>
#include <opencv2/core/core.hpp>
#include <opencv2/imgproc/imgproc.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <opencv2/features2d/features2d.hpp>
#include <opencv2/xfeatures2d.hpp>
#include <opencv2/calib3d/calib3d.hpp>
#include "config.h"

#include <iomanip>
#include <iostream>
#include <QDirIterator>
#include <QString>
#include "QtWidgets/QtWidgets"
#include "QtWidgets/QApplication"


using namespace cv;
using namespace std;


class calibrator
{
    private:
    QString l_path, r_path; //path for folders containing left and right
checkerboard images
    vector<Mat> l_images, r_images; //left and right checkerboard images
    Mat l_cameraMatrix, l_distCoeffs, r_cameraMatrix, r_distCoeffs; //Mats
for holding individual camera calibration information
    bool show_chess_corners; //visualize checkerboard corner detections?
    float side_length; //side length of checkerboard squares
    int width, height; //number of internal corners in checkerboard along
width and height
    vector<vector<Point2f> > l_image_points, r_image_points; //left and
right image points
    vector<vector<Point3f> > object_points; //object points (grid)
    Mat R, T, E, F; //stereo calibration information

    public:
    calibrator(QString, QString, float, int, int); //constructor
    bool calibrate(); //function to calibrate stereo camera
    void calc_image_points(bool); //function to calculae image points by
detecting checkerboard corners
};

calibrator::calibrator(QString _l_path, QString _r_path, float _side_length,
int _width, int _height){
    side_length = _side_length;
    width = _width;
    height = _height;

    l_path = _l_path;
    r_path = _r_path;

    // Read images
    QDirIterator it(l_path);
    while (it.hasNext()){
        QString l_filename = l_path + it.fileName();
```

```
        if(it.fileName().contains("jpg")){
            cout << "Lfile: " << l_filename.toStdString() << endl;
            QString tmpS = it.fileName();
            tmpS.replace(0, 4, "right");

            QString r_filename = r_path + tmpS;
            cout << "Rfile: " << r_filename.toStdString() << endl;
            Mat lim = imread(l_filename.toStdString()),
                rim = imread(r_filename.toStdString());
            if(!lim.empty() && !rim.empty()){
                l_images.push_back(lim);
                r_images.push_back(rim);
            }else cout << "NOK_l: " << lim.empty() << " NOK_r: " <<
rim.empty() << endl;
        }
        it.next();
    }
}
void calibrator::calc_image_points(bool show){
    // Calculate the object points in the object co-ordinate system (origin
at top left corner)
    vector<Point3f> ob_p;
    for(int i = 0; i < height; i++){
        for(int j = 0; j < width; j++){
            ob_p.push_back(Point3f(j * side_length, i * side_length, 0.f));
        }
    }

    if(show){
        namedWindow("Left Chessboard corners");
        namedWindow("Right Chessboard corners");
    }

    for(int i = 0; i < int(l_images.size());){
        Mat lim = l_images[i], rim = r_images[i];
        vector<Point2f> l_im_p, r_im_p;
//etsitään shakki kuvio kummankin kameran kuvasta, tässä käytin samoja
parametri asetuksia kuin edellisessäkin yksittäisten kameroiden
kalibroinnissa
        bool l_pattern_found = findChessboardCorners(lim, Size(width,
height), l_im_p,
             CALIB_CB_ADAPTIVE_THRESH +
             CALIB_CB_NORMALIZE_IMAGE +
             CV_CALIB_CB_FILTER_QUADS); //+ CALIB_CB_FAST_CHECK);
        bool r_pattern_found = findChessboardCorners(rim, Size(width,
height), r_im_p,
             CALIB_CB_ADAPTIVE_THRESH +
             CALIB_CB_NORMALIZE_IMAGE +
             CV_CALIB_CB_FILTER_QUADS);//+ CALIB_CB_FAST_CHECK);

cout << "l_pattern_found: " << l_pattern_found <<"  " << r_pattern_found <<
" :r_pattern_found"    << endl;
        if(l_pattern_found && r_pattern_found){
            cout << "pattern_found_l picture: " << i << endl;
            object_points.push_back(ob_p);

//risteys kohtien tarkempi selvitys tehdään myös molemmille kuville
            Mat gray;
            cvtColor(lim, gray, CV_BGR2GRAY);
            cornerSubPix(gray, l_im_p, Size(5, 5), Size(-1, -1),
TermCriteria(CV_TERMCRIT_EPS +
            CV_TERMCRIT_ITER, 30, 0.1));
            cvtColor(rim, gray, CV_BGR2GRAY);
            cornerSubPix(gray, r_im_p, Size(11, 11), Size(-1, -1),
TermCriteria(CV_TERMCRIT_EPS +
            CV_TERMCRIT_ITER, 30, 0.1));
            l_image_points.push_back(l_im_p);
            r_image_points.push_back(r_im_p);
```

```
            if(show){
                Mat im_show = lim.clone();
                drawChessboardCorners(im_show, Size(width, height), l_im_p,
true);
                imshow("Left Chessboard corners", im_show);
                im_show = rim.clone();
                drawChessboardCorners(im_show, Size(width, height), r_im_p,
true);
                imshow("Right Chessboard corners", im_show);
                while(char(waitKey(1)) != ' ') {}
            }
            i++;
        }
        else{
            l_images.erase(l_images.begin() + i);
            r_images.erase(r_images.begin() + i);
        }
    }
}

bool calibrator::calibrate(){

    // luetaan edellisellä ohjelma pätkällä tehdyt kameroiden kalibrointi
tiedostot
    string filename = DATA_FOLDER + string("cam_calibL.xml");
    FileStorage fs(filename, FileStorage::READ);
    fs["cameraMatrix"] >> l_cameraMatrix;
    fs["distCoeffs"] >> l_distCoeffs;
    fs.release();

    filename = DATA_FOLDER + string("cam_calibR.xml");
    fs.open(filename, FileStorage::READ);
    fs["cameraMatrix"] >> r_cameraMatrix;
    fs["distCoeffs"] >> r_distCoeffs;
    fs.release();

    if(!l_cameraMatrix.empty() && !l_distCoeffs.empty() &&
!r_cameraMatrix.empty() && !r_distCoeffs.empty()) {
        cout << "object_points: " << object_points.empty() << endl
             << "l_image_points: " << l_image_points.empty() << endl
             << "r_image_points: " << r_image_points.empty() << endl;

        cout << "l_cameraMatrix: " << endl << l_cameraMatrix << endl
             << "r_cameraMatrix: " << endl << r_cameraMatrix<< endl
             << "l_distCoeffs: " << endl << l_distCoeffs<< endl
             << "r_distCoeffs: " << endl <<r_distCoeffs<< endl
             << "R: " << R<< endl
             << "T: " << T<< endl
             << "E: " << E<< endl
             << "F: " << F<< endl;

        double rms = stereoCalibrate(object_points, l_image_points,
r_image_points, l_cameraMatrix, l_distCoeffs, r_cameraMatrix, r_distCoeffs,
l_images[0].size(), R, T, E, F
            ,CALIB_FIX_ASPECT_RATIO +
            CALIB_ZERO_TANGENT_DIST +
            //CALIB_SAME_FOCAL_LENGTH +
            ,CALIB_RATIONAL_MODEL +
            ,CALIB_FIX_K3 + CALIB_FIX_K4 + CALIB_FIX_K5,
            TermCriteria(TermCriteria::COUNT+TermCriteria::EPS, 100, 1e-6));
        cout << "Calibrated stereo camera with a RMS error of " << rms <<
endl;
        cout << "l_cameraMatrix: " << endl << l_cameraMatrix << endl
             << "r_cameraMatrix: " << endl << r_cameraMatrix<< endl
             << "l_distCoeffs: " << endl << l_distCoeffs<< endl
             << "r_distCoeffs: " << endl << r_distCoeffs<< endl
             << "R: " << R<< endl
```

```
                << "T: " << T<< endl
                << "E: " << E<< endl
                << "F: " << F << endl;
// talletetaan stereo kalibroinnin parametrit omaan tiedostoonsa
        string filename = DATA_FOLDER + string("stereo_calibXX.xml");

        fs.open(filename,   FileStorage::WRITE);
        fs << "l_cameraMatrix" << l_cameraMatrix;
        fs << "r_cameraMatrix" << r_cameraMatrix;
        fs << "l_distCoeffs" << l_distCoeffs;
        fs << "r_distCoeffs" << r_distCoeffs;
        fs << "R" << R;
        fs << "T" << T;
        fs << "E" << E;
        fs << "F" << F;
        cout << "Calibration parameters saved to " << filename << endl;
        return true;
    }
    else return false;
}


int main(int argc, char *argv[]){

    QApplication app(argc, argv);
    QDesktopWidget * desk = qApp->desktop() ;

    calibrator calib(LEFT_FOLDER, RIGHT_FOLDER, 22.5f, 7, 10);
    calib.calc_image_points(true);
    bool done = calib.calibrate();

    if(!done) cout << "Stereo Calibration not successful because individial
calibration matricescould not be read" << endl;

return 0;
}
```

LIITE 4

```
// Program illustrate single camera calibration
// kirjan esimerkki 10_5
// Author: Samarth Manoj Brahmbhatt, University of Pennsylvania
// JuV 2016: päivitys opencv 2.x → 3.x, boost kirjaston poisto

#include <vector>
#include <opencv2/core/core.hpp>
#include <opencv2/imgproc/imgproc.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <opencv2/features2d/features2d.hpp>
#include <opencv2/xfeatures2d.hpp>
#include <opencv2/calib3d/calib3d.hpp>
#include "config.h"

#include <iomanip>
#include <iostream>
#include <QDirIterator>
#include <QString>
#include "QtWidgets/QtWidgets"
#include "QApplication"

using namespace cv;
using namespace std;

class rectifier {
    private:
        Mat map_l1, map_l2, map_r1, map_r2; //pixel maps for
rectification
        string path;

    public:
        rectifier(string, Size); //constructor
        void show_rectified(Size); //function to show live rectified
feed from stereo camera

};

rectifier::rectifier(string filename, Size image_size) {

// Read individal camera calibration information from saved XML file

    Mat l_cameraMatrix, l_distCoeffs, r_cameraMatrix, r_distCoeffs;
    Mat R, T;

    FileStorage fs(filename, FileStorage::READ);

    cout << "filenameREAD: " << filename << endl;

    fs["l_cameraMatrix"] >> l_cameraMatrix;
    fs["l_distCoeffs"] >> l_distCoeffs;
    fs["r_cameraMatrix"] >> r_cameraMatrix;
    fs["r_distCoeffs"] >> r_distCoeffs;
    fs["R"] >> R;
    fs["T"] >> T;
    fs.release();

    if(l_cameraMatrix.empty() || r_cameraMatrix.empty()
       || l_distCoeffs.empty() ||r_distCoeffs.empty()
```

```
        || R.empty() || T.empty())
        cout << "Rectifier: Loading of files not successful" << endl;
    // Calculate transforms for rectifying images
    Mat Rl, Rr, Pl, Pr, Q;

    stereoRectify(l_cameraMatrix, l_distCoeffs,
                  r_cameraMatrix, r_distCoeffs,
                  image_size, R, T, Rl,Rr, Pl, Pr, Q);

    // Calculate pixel maps for efficient rectification of
    // images via lookup tables
    initUndistortRectifyMap(l_cameraMatrix, l_distCoeffs,
                            Rl, Pl, image_size, CV_16SC2,
                            map_l1, map_l2);
    initUndistortRectifyMap(r_cameraMatrix, r_distCoeffs,
                            Rr, Pr, image_size, CV_16SC2,
                            map_r1, map_r2);

    fs.open(filename, FileStorage::APPEND);
    fs << "Rl" << Rl;
    fs << "Rr" << Rr;
    fs << "Pl" << Pl;
    fs << "Pr" << Pr;
    fs << "Q" << Q;
    fs << "map_l1" << map_l1;
    fs << "map_l2" << map_l2;
    fs << "map_r1" << map_r1;
    fs << "map_r2" << map_r2;
    fs.release();
}

void rectifier::show_rectified(Size image_size) {
    VideoCapture capr(2), capl(3);

    //reduce frame size
    capl.set(CV_CAP_PROP_FRAME_HEIGHT, image_size.height);
    capl.set(CV_CAP_PROP_FRAME_WIDTH, image_size.width);
    capr.set(CV_CAP_PROP_FRAME_HEIGHT, image_size.height);
    capr.set(CV_CAP_PROP_FRAME_WIDTH, image_size.width);

    while(char(waitKey(1)) != 'q') {
        //grab raw frames first
        capl.grab();
        capr.grab();
        //decode later so the grabbed frames are less apart in time
        Mat framel, framel_rect, framer, framer_rect;

        capl.retrieve(framel);
        capr.retrieve(framer);
        if(framel.empty() || framer.empty())
        {
           cout << "EMPTY" <<endl;
           break;
        }
        // Remap images by pixel maps to rectify
        remap(framel, framel_rect, map_l1, map_l2, INTER_LINEAR);
        remap(framer, framer_rect, map_r1, map_r2, INTER_LINEAR);

        // Make a larger image containing the left and right
        // rectified images side-by-side
        Mat combo(image_size.height, 2 * image_size.width, CV_8UC3);
```

```
            Mat combo2(2*image_size.height, image_size.width, CV_8UC3);

            imshow("framel_rect", framel_rect);
            imshow("framer_rect", framer_rect);
            hconcat(framel_rect,framer_rect,combo);
            vconcat(framel_rect,framer_rect,combo2);

            for(int y = 0; y < combo.rows; y += 20)
               line(combo, Point(0, y), Point(combo.cols, y),
                     Scalar(0, 0, 255));
            imshow("Combo", combo);

            for(int x = 0; x < combo2.cols; x += 20)
                line(combo2, Point(x, 0), Point(x, combo2.rows),
                     Scalar(0, 0, 255));
            imshow("Combo2", combo2);

        }
        capl.release();
        capr.release();
}

int main(int argc, char *argv[]){

string filename = DATA_FOLDER + string("stereo_calibXX.xml");
QApplication app(argc, argv);
QDesktopWidget * desk = qApp->desktop() ;


        Size image_size(320, 240);
        rectifier rec(filename, image_size);
        rec.show_rectified(image_size);

return 0; }
```

LIITE 5

```
// kirjan esimerkki 10_7
// Author: Samarth Manoj Brahmbhatt, University of Pennsylvania
// JuV 2016: päivitys opencv 2.x → 3.x, boost kirjaston poisto
// JuV 2017: lisätty kiinnostavien pisteiden haku ja niiden pohjalta etäisyyden
//           mittaus

#include <vector>
#include <opencv2/core/core.hpp>
#include <opencv2/imgproc/imgproc.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <opencv2/features2d/features2d.hpp>
#include <opencv2/xfeatures2d.hpp>
#include <opencv2/calib3d/calib3d.hpp>
#include <opencv2/photo.hpp>
#include "config.h"
#include <math.h>
#include <iomanip>
#include <iostream>
#include <QDirIterator>
#include <QString>
#include <QCoreApplication>
#include "QtWidgets/QtWidgets"
#include "QApplication"

using namespace cv;
using namespace std;
using namespace cv::xfeatures2d;

double myDep (Mat InputPCloud){

    int from_to[] = {2, 0};
    Mat tmpROI(InputPCloud.size(), CV_32FC1);
    mixChannels(&InputPCloud, 1, &tmpROI, 1, from_to, 1);

    tmpROI = tmpROI.reshape(0,1);
    vector<double> vecInput;
    tmpROI.copyTo(vecInput);
    int kpl=0;
    int sum=0;
    for (int i=0;i<(int)vecInput.size();i++){
        if (vecInput[i] <10000) {
            kpl++;
            sum+=vecInput[i];
        }
    }
    if (kpl >0 && sum > 0)
        return sum/kpl;
    else
        return 0;
}

class disparity {

    private:

        Mat map_l1, map_l2, map_r1, map_r2, Q;

        //StereoSGBM stereo;
        Ptr<StereoSGBM> stereo = StereoSGBM::create(0,16,3);
        int min_disp, num_disp,num_Cap,num_Size;

    public:

        disparity(string);//, Size);
```

```
        void set_minDisp(int minDisp) { stereo->setMinDisparity(minDisp); }
        void set_numDisp(int numDisp) { stereo->setNumDisparities(numDisp);
}
        void set_preFilterCap(int numCap) { stereo->setPreFilterCap(numCap);
}
        void set_preFilterSize(int numSize) { stereo -
>setBlockSize(numSize);}
        void show_disparity(Size);
};

void on_minDisp(int min_disp, void * _disp_obj) {
    disparity * disp_obj = (disparity *) _disp_obj;
    //if (min_disp < 31) min_disp = 31;
    setTrackbarPos("minDisparity", "Disparity", min_disp);
    //disp_obj -> set_minDisp(min_disp - 30);
    disp_obj -> set_minDisp(min_disp);

}

void on_numDisp(int num_disp, void * _disp_obj) {
    disparity * disp_obj = (disparity *) _disp_obj;
    if (num_disp < 16) num_disp=16;
    num_disp = (num_disp / 16) * 16;
    setTrackbarPos("numDisparity", "Disparity", num_disp);
    disp_obj -> set_numDisp(num_disp);

}
void on_preFilterCap(int num_Cap, void * _disp_obj) {
    disparity * disp_obj = (disparity *) _disp_obj;
    //if (num_Cap < 1) num_Cap=1;
    //if (num_Cap > 63) num_Cap=63;
    setTrackbarPos("preFilterCap", "Disparity", num_Cap);
    disp_obj -> set_preFilterCap(num_Cap);

}
void on_preFilterSize(int num_Size, void * _disp_obj) {
    disparity * disp_obj = (disparity *) _disp_obj;
    if (num_Size < 1) num_Size=1;
    if (num_Size > 63) num_Size=63;
    setTrackbarPos("preFilterSize", "Disparity", num_Size);
    disp_obj -> set_preFilterSize(num_Size);

}


disparity::disparity(string filename){//, Size image_size) {

    FileStorage fs(filename, FileStorage::READ);

    cout << "file: " << filename << endl;

    fs["map_l1"] >> map_l1;
    fs["map_l2"] >> map_l2;
    fs["map_r1"] >> map_r1;
    fs["map_r2"] >> map_r2;
    fs["Q"] >> Q;

    if(map_l1.empty() || map_l2.empty() || map_r1.empty() || map_r2.empty()
|| Q.empty())
        cout << "WARNING: Loading of mapping matrices not successful" <<
endl;


     stereo->setPreFilterCap(45);//52//63
     stereo->setBlockSize(3); //37//9//3
     stereo->setUniquenessRatio(10);//10
     stereo->setSpeckleWindowSize(100);
     stereo->setSpeckleRange(32); //8 /32
```

```
    stereo->setDisp12MaxDiff(1);
    stereo->setMode(StereoSGBM::MODE_SGBM);
}

void disparity::show_disparity(Size image_size) {

    VideoCapture capr(2), capl(3);

    //reduce frame size

    capl.set(CV_CAP_PROP_FRAME_HEIGHT, image_size.height);
    capl.set(CV_CAP_PROP_FRAME_WIDTH, image_size.width);
    capr.set(CV_CAP_PROP_FRAME_HEIGHT, image_size.height);
    capr.set(CV_CAP_PROP_FRAME_WIDTH, image_size.width);

    min_disp = 30;
    num_disp = ((image_size.width / 8) + 15) & -16;

    namedWindow("Disparity", CV_WINDOW_AUTOSIZE);
    moveWindow("Disparity",50,50);
    namedWindow("Disparity",WINDOW_AUTOSIZE);
    namedWindow("POINTS", CV_WINDOW_AUTOSIZE);
    moveWindow("POINTS",50,50);
    namedWindow("POINTS",WINDOW_AUTOSIZE);
    namedWindow("Left_rect",WINDOW_AUTOSIZE);
    moveWindow("Left_rect",50,50);
    namedWindow("Left_rect",WINDOW_AUTOSIZE);


    createTrackbar("minDisparity + 30", "Disparity", &min_disp, 100,
                   on_minDisp, (void *)this);
    createTrackbar("numDisparity", "Disparity", &num_disp, 300,
                   on_numDisp, (void *)this);
    createTrackbar("preFilterCap", "Disparity", &num_Cap, 100,
                   on_preFilterCap, (void *)this);
    createTrackbar("preFilterSize", "Disparity", &num_Size, 100,
                   on_preFilterSize, (void *)this);

    on_minDisp(min_disp, this);
    on_numDisp(num_disp, this);
    on_preFilterCap(num_Cap, this);
    on_preFilterSize(num_Size, this);

    while(char(waitKey(1)) != 'q') {

        //grab raw frames first
        capl.grab();
        capr.grab();
        //decode later so the grabbed frames are less apart in time

        Mat framel, framel_rect, framer, framer_rect;

        capl.retrieve(framel);
        capr.retrieve(framer);
        if(framel.empty() || framer.empty())
            break;

        remap(framel, framel_rect, map_l1, map_l2, INTER_LINEAR);
        remap(framer, framer_rect, map_r1, map_r2, INTER_LINEAR);

        Mat disp, disp_show, disp_compute,pointcloud;

        cv::Mat RAWimage1 = framel_rect.clone();
        cv::Mat RAWimage2 = framer_rect.clone();

        cv::Mat image1 = RAWimage1.clone();
        cv::Mat image2 = RAWimage2.clone();
```

```
Mat image2_gray,image2_blur;
cvtColor(image2,image2_gray,CV_BGR2GRAY);
blur (image2_gray,image2_blur,Size(3,3));

Mat image2dst;
image2dst = Scalar::all(0);
image2.copyTo(image2dst,image2_blur);

Mat image1_gray,image1_blur;
cvtColor(image1,image1_gray,CV_BGR2GRAY);
blur (image1_gray,image1_blur,Size(3,3));

Mat image1dst;
image1dst = Scalar::all(0);
image1.copyTo(image1dst,image1_blur);

// vector of keypoints
std::vector<cv::KeyPoint> keyPoints1;
std::vector<cv::KeyPoint> keyPoints2;

Ptr<SURF> Surf = SURF::create (3000);
// Detection of the SURF features
Surf->detect(image1_blur,keyPoints1);
Surf->detect(image2_blur,keyPoints2);

Ptr<SURF> SurfDesc = SURF::create();

cv::Mat descrip1, descrip2;
SurfDesc->compute(image1_blur,keyPoints1,descrip1);
SurfDesc->compute(image2_blur,keyPoints2,descrip2);

cv::BFMatcher Matcher( cv::NORM_L2, false );
std::vector< cv::DMatch > Matches;
Matcher.match( descrip1, descrip2, Matches );
sort(Matches.begin(),Matches.end());

// Convert keypoints into Point2f
std::vector<cv::Point2f> Points1, Points2;
int sortID=0;
for (std::vector<cv::DMatch>::const_iterator it= Matches.begin();
     it!= Matches.end(); ++it) {

         // Get the position of left keypoints
         float x= keyPoints1[it->queryIdx].pt.x;
         float y= keyPoints1[it->queryIdx].pt.y;
         Points1.push_back(cv::Point2f(x,y));
         // Get the position of right keypoints
         x= keyPoints2[it->trainIdx].pt.x;
         y= keyPoints2[it->trainIdx].pt.y;
         Points2.push_back(cv::Point2f(x,y));
         if (sortID++ > 10 )
               break;
}

stereo->compute(image1, image2, disp);

normalize(disp, disp_show, 0, 255, CV_MINMAX, CV_8U);
disp.convertTo(disp_compute, CV_32F, 1.f/16.f);

reprojectImageTo3D(disp_compute, pointcloud, Q,true);

Mat pointImage = image1;

std::vector<cv::Point2f> Mypoints1in, Mypoints2in;

Mypoints1in = Points1;
int pointsAmount = Mypoints1in.size();
```

```
int xC =framel.cols/2;
int yC = framel.rows/2;

for(int k=0; k<pointsAmount; k++){

                ostringstream ss;
                float x = Mypoints1in[k].x;   //first value
                float y = Mypoints1in[k].y;   //second value
                Vec3f MyPoint =  pointcloud.at<Vec3f>(y,x);

                double X = xC-x;
                double Y = yC-y;
                double angle = (atan2(X,Y)*180)/3.14159;

                if (MyPoint[2] < 10000) {
                    cv::circle(pointImage,Mypoints1in[k],3,
                               cv::Scalar(255,255,255),1);
                    ss << MyPoint[2] << ", " << angle;
                    putText(pointImage, ss.str(), Mypoints1in[k],
                            FONT_HERSHEY_PLAIN, 1,
                            Scalar(0,0,255,255));
                }
                cout << "Z: " << ss.str() << endl;

}
imshow("POINTS", pointImage);

int finderS = 5;
int finderP = 40; // length between finder and border;
int xmin = framel.cols/2 - finderS,
    xmax = framel.cols/2 + finderS/2,
    ymin = framel.rows/2 - finderS/2,
    ymax = framel.rows/2 + finderS/2;

ostringstream ss;

// the upper left corner
Mat pointcloudTMP = pointcloud(Range(finderP, finderP+finderS),
                               Range(finderP,finderP+finderS));
rectangle(framel_rect, Point(finderP, finderP),
          Point(finderP+finderS,finderP+finderS), Scalar(0, 0, 255));
ss << myDep(pointcloudTMP);
cout << "DepthZ_TL: " << ss.str() << " mm" << endl;
putText(framel_rect, ss.str(),
          Point(finderP,finderP+finderS*2+finderS*2),
          FONT_HERSHEY_PLAIN, 1,  Scalar(0,0,255,255));
ss.str("");

// top center
pointcloudTMP = pointcloud(Range(finderP, finderP+finderS),
                               Range(xmin,xmax));
rectangle(framel_rect, Point(xmin, finderP),
                Point(xmax,finderP+finderS), Scalar(0, 0, 255));
ss << myDep(pointcloudTMP);
int len =ss.str().length()*10/2;
cout << "DepthZ_TC: " << ss.str() << " mm" << endl;
putText(framel_rect, ss.str(), Point(xmin-len,
                finderP+finderS+finderS*2), FONT_HERSHEY_PLAIN, 1,
                Scalar(0,0,255,255));
ss.str("");



// the upper right corner
pointcloudTMP = pointcloud(Range(finderP, finderP+finderS),
    Range(framel.cols-finderP-finderS,framel.cols-finderP));
rectangle(framel_rect, Point(framel.cols-finderP-finderS, finderP),
                Point(framel.cols-finderP,finderP+finderS),
```

```
    Scalar(0, 0, 255));
ss << myDep(pointcloudTMP);
len =ss.str().length()*10;
cout << "DepthZ_TR: " << ss.str() << " mm" << endl;
putText(framel_rect, ss.str(), Point(framel.cols-finderP-len,
        finderP+finderS*2), FONT_HERSHEY_PLAIN, 1,
        Scalar(0,0,255,25));
ss.str("");


// central left
pointcloudTMP = pointcloud(Range(ymin, ymax),
                Range(finderP,finderP+finderS));
rectangle(framel_rect, Point(finderP, ymin),
          Point(finderP+finderS, ymax), Scalar(0, 0, 255));
ss << myDep(pointcloudTMP);
cout << "DepthZ_CL: " << ss.str() << " mm" << endl;
putText(framel_rect, ss.str(), Point(finderP,ymax+finderS*2),
        FONT_HERSHEY_PLAIN, 1,  Scalar(0,0,255,255));
ss.str("");

// cetral central
pointcloudTMP = pointcloud(Range(ymin, ymax), Range(xmin, xmax));
rectangle(framel_rect, Point(xmin, ymin), Point(xmax,ymax),
          Scalar(0, 0, 255));
ss << myDep(pointcloudTMP);
cout << "DepthZ_CC: " << ss.str() << " mm" << endl;
len =ss.str().length()*10/2;
putText(framel_rect, ss.str(), Point(xmin-len,ymax+finderS*2),
        FONT_HERSHEY_PLAIN, 1,  Scalar(0,0,255,255));
ss.str("");

// central right
pointcloudTMP = pointcloud(Range(ymin,ymax),
            Range(framel.cols-finderP-finderS,framel.cols-finderP));
rectangle(framel_rect, Point(framel.cols-finderP-finderS, ymin),
          Point(framel.cols-finderP,ymax), Scalar(0, 0, 255));
ss << myDep(pointcloudTMP);
len =ss.str().length()*10;
cout << "DepthZ_CR: " << ss.str() << " mm" << endl;
putText(framel_rect, ss.str(),
         Point(framel.cols-finderP-len, ymax+finderS*2),
         FONT_HERSHEY_PLAIN, 1,  Scalar(0,0,255,255));
ss.str("");


// Left down corner
pointcloudTMP = pointcloud(
             Range(framel.rows-finderP-finderS,framel.rows-finderP),
             Range(finderP,finderP+finderS));
rectangle(framel_rect, Point(finderP,framel.rows-finderP-finderS),
          Point(finderP+finderS, framel.rows-finderP),
          Scalar(0, 0, 255));
ss << myDep(pointcloudTMP);
cout << "DepthZ_DL: " << ss.str() << " mm" << endl;
putText(framel_rect, ss.str(),
        Point(finderP, framel.rows-finderP-finderS),
        FONT_HERSHEY_PLAIN, 1,  Scalar(0,0,255,255));
ss.str("");

// down central
pointcloudTMP = pointcloud(
     Range(framel.rows-finderP-finderS, framel.rows-finderP),
     Range(xmin, xmax));
rectangle(framel_rect, Point(xmin, framel.rows-finderP-finderS),
          Point(xmax,framel.rows-finderP), Scalar(0, 0, 255));
ss << myDep(pointcloudTMP);
len =ss.str().length()*10/2;
```

```
            cout << "DepthZ_DC: " << ss.str() << " mm" << endl;
            putText(framel_rect, ss.str(),
                    Point(xmin-len,framel.rows-finderP-finderS),
                    FONT_HERSHEY_PLAIN, 1,  Scalar(0,0,255,255));
            ss.str("");

            // down right
            pointcloudTMP = pointcloud(
                  Range(framel.rows-finderP-finderS, framel.rows-finderP),
                  Range(framel.cols-finderP-finderS,framel.cols-finderP));
            rectangle(framel_rect,
               Point(framel.cols-finderP-finderS, framel.rows-finderP-finderS ),
               Point(framel.cols-finderP,framel.rows-finderP),
               Scalar(0, 0, 255));
            ss << myDep(pointcloudTMP);
            len =ss.str().length()*10;
            cout << "DepthZ_DR: " << ss.str() << " mm" << endl;
            putText(framel_rect, ss.str(),
                    Point(framel.cols-finderP-len, framel.rows-finderP-finderS),
                    FONT_HERSHEY_PLAIN, 1,  Scalar(0,0,255,255));
            ss.str("");



            imshow("Disparity", disp_show);
            imshow("Left_rect", framel_rect);

        }
        capl.release();
        capr.release();

}

int main() {

    string filename = DATA_FOLDER + string("stereo_calibXX.xml");

    Size image_size(320, 240);//    Size image_size(640, 480);
    disparity disp(filename );
    disp.show_disparity(image_size);

    return 0;
}
```