TIETO- JA SÄHKÖTEKNIIKAN TIEDEKUNTA

Janne Mustaniemi
Eemeli Ristimella
Joonas Jyrkkä

Esineiden mittaaminen älypuhelimella

Kandidaatintyö
Tietotekniikan tutkinto-ohjelma
Toukokuu 2017

Mustaniemi Janne, Ristimella Eemeli, Jyrkkä Joonas (2017) Esineiden mittaaminen älypuhelimella. Oulun yliopisto, tietotekniikan tutkinto-ohjelma. Kandidaatintyö, 47 s.

TIIVISTELMÄ

Tämä työ esittelee ohjelmistoratkaisun esineiden mittaamiseen Android-älypuhelimelle. Mittaaminen tapahtuu kameran kuvaa hyödyntäen vertaamalla mitattavia kohteita referenssiesinein, jonka mitat tunnetaan. Mitattavat kohteet ja oleelliset mitat täydennetään kameranäkymään. Sovelluksen kannalta tärkeimmät prosessit ovat referenssiesineen ja mitattavan esineen tunnistaminen.

Referenssiesineen tunnistus tapahtuu värin perusteella. Aluksi kuva muutetaan harmaasävykuvaksi, minkä jälkeen sovelletaan Cannyn algoritmia reunojen tunnistamiseen. Syntyvä binäärikuva käsitellään dilaatio-operaatioilla, jonka jälkeen hyödynnetään OpenCV-kirjaston findContours-metodia, joka etsii kuvasta suljetut alueet (contours). Löytyneiden alueiden joukkoa karsitaan värisävyn perusteella.

Mitattavan esineen tunnistusprosessi alkaa myös harmaasävymuunnoksella, josta syntyvä kuva kynnystetään binäärikuvaksi. Syntyneeseen binäärikuvaan sovelletaan findContours-metodia ja löytyneet alueet suodatetaan pinta-alan mukaan.

Sovelluksesta toteutettiin kaksi versiota, jotka molemmat testattiin ennalta määritellyillä testipattereilla. Testien perusteella näyttää siltä, että sovellus toimii hyvin tasaisissa valaistusolosuhteissa, mutta kuvioitu pinta ja haastava valaistus saattavat hankaloittaa tunnistamista. Ongelmia syntyy myös referenssiesineen ja mitattavan esineen ollessa toistensa päällä ja silloin, kun mitattava esine ei erotu tarpeeksi hyvin taustasta.

Avainsanat: OpenCV, Android

Mustaniemi Janne, Ristimella Eemeli, Jyrkkä Joonas (2017) Measuring objects with a smartphone. University of Oulu, Degree Programme in Computer Science and Engineering. Bachelor's Thesis, 47 p.

ABSTRACT

This bachelor's thesis presents an Android-based solution to measure ubiquitous objects. The measurement process is executed by comparing measurable objects to a known reference object. The measures of the object will be added on top of camera feed. The crucial processes of the application are detecting the reference object and the measurable object.

The detection of the reference object is achieved on the basis of color information. The process starts with converting the camera feed to grayscale, and then applying Canny's edge detection algorithm. The resulting binary image is dilated, and the contours of the image are detected using findCountours-method from OpenCV-library. The detected contour areas are filtered by color and size.

The detection of a measurable object begins also with grayscale-transform followed by binary thresholding. FindContours-method is applied to the resulting binary image, and the areas detected are filtered by size.

Two versions of the application were implemented, and they both were tested with predefined test suites. According to the tests, it seems the application succeeds to carry out the measurement process. However, there is still work to be done to enhance the detection rate especially in a challenging lightning environment and on patterned surfaces.

Keywords: OpenCV, Android

SISÄLLYSLUETTELO

ALKULAUSE

Tämä työ toteutettiin Sulautettujen ohjelmistojen projekti-kurssilla. Kiitämme työn valvojaa Teemu Tokolaa ja professori Juha Röningiä.

Oulu, toukokuu 2017
Janne Mustaniemi, Eemeli Ristimella, Joonas Jyrkkä

LYHENTEIDEN JA MERKKIEN SELITYKSET

BRIEF	Binary Robust Independent Elementary Features-algoritmi
C	C-ohjelmointikieli
C++	C++-ohjelmointikieli
FAST	Features from Accelerated Segment Test -algoritmi
Java	Java-ohjelmointikieli
Javascript	Javascript-ohjelmointikieli
RGBA	Red-Green-Blue-Alpha värimalli
HSV	Hue-Saturation-Value-värimalli
JNI	Java Native Interface
OpenCV	Open Source Computer Vision-kirjasto
ORB	Oriented FAST and Rotated BRIEF-algoritmi
SIFT	Scale-Invariant Feature Transform-algoritmi
SURF	Speeded-Up Robust Features-algoritmi

1. JOHDANTO

Esineiden mittaaminen on yleinen ongelma, mutta jotta esine voidaan mitata, mittaajalla on oltava mukanaan jokin työkalu, esimerkiksi mittanauha tai viivoitin. Näin ei aina kuitenkaan ole, mutta yhä useammalla on nykyään mukanaan älypuhelin.

Tämä työ pyrki ratkaisemaan edellä kuvatun ongelman Android-käyttöjärjestelmälle suunnatulla sovelluksella, joka mittaa halutun esineen älypuhelimen kameraa hyödyntäen. Pelkästä kameran kuvasta yksinään ei voida päätellä pikseleiden kokoa oikeassa maailmassa: tarvitaan siis jokin toinen esine, referenssiesine, jonka mitat tiedetään. Kun referenssiesine tunnistetaan, voidaan myös laskea muut halutut mitat. Työn päätavoitteena oli siis toteuttaa esineitä mittaava sovellus referenssiesineen tunnistamiseen perustuen.

Referenssiesineenä voisi toimia jokin yleinen esine, esimerkiksi yhden euron kolikko. Voisi myös olla hyödyllistä antaa käyttäjän määrittää referenssiesine itse. Referenssiesine pitää kuitenkin fyysisesti mitata, eli käyttäjän unohtaessa ennalta määritellyn referenssiesineensä kotiin, hän tarvitsee joka tapauksessa jonkin fyysisen mittaustyökalun. Kuitenkin mahdollisuus määrittää referenssiesine itse antaa käyttäjälle vapauden valita mitä referenssiesinettä pitää mukanaan.

Toinen hyöty fyysisen mittaustyökalun korvaamisesta referenssiesineellä on itse mittaamisen automatisointi. Erityisesti jos mittaväline on lyhyempi kuin sillä mitattava reuna, esimerkkinä valkotaulun mittaaminen viivoittimella, prosessista tulee työläs. Lisäksi mittaaminen kameranäkymän kautta mahdollistaisi myös useamman reunan mittaamisen yhtä aikaa: kun pikselin pituus tiedetään, voidaan kaikki mahdolliset reunat mitata.

Entä mistä sovellus tietää mitkä reunat ovat tärkeitä? Jos kaikkien kuvassa olevien esineiden mitat näytetään, kuvasta voi tulla joissakin tapauksissa käyttökelvoton. Lisäksi turhien reunojen laskeminen vie laitteen laskentaresursseja. Yksi mahdollisuus voisi olla reunojen etsiminen referenssiesineen läheltä tai käyttäjä voisi auttaa sovellusta napauttamalla aluetta, jossa mitattava reuna on.

Mittojen laskeminen ei kuitenkaan yksinään riitä, vaan mitat pitää myös esittää käyttäjälle. Luonteva esitystapa voisi olla reunojen ja lukujen piirtäminen dynaamisesti kameranäkymän päälle. Toinen vaihtoehto voisi olla kuvan ottaminen mitattavasta esineestä, jonka päälle mitat piirrettäisiin.

Mittaamisen helpottamisen lisäksi sovellus toimi myös testinä konenäköteknologien käyttämiselle Android-alustalla. Erityisesti kamerakuvan prosessointi matalan suorituskyvyn laitteistolla tuotti haasteita. Yksi päämäärä tälle työlle olikin selvittää mitä näille laitteille on mahdollista toteuttaa.

2. HAHMONTUNNISTUKSEN TAUSTAA

Kaksi oleellista tehtävää, joista mittaussovelluksen tulee suoriutua, ovat referenssiesineen ja mitattavan kohteen tunnistaminen. Tässä osiossa kartoitetaan teknologioita, jotka mahdollistavat näiden tehtävien toteutumisen.

Suuri osa termeistä on englanninkielisiä, eikä niistä ole käytössä vakiintunutta suomennosta. Nämä termit on käännetty vapaasti suomeksi ja niiden alkuperäinen kieliasu on suluissa englanniksi.

2.1. Esikäsittely

Kuvia on mahdollista esikäsitellä ennen hahmontunnistusta. Tarkoituksena on yleensä kohinan poistaminen kuvasta tai kuvan muokkaaminen tarkoituksenmukaisemmaksi tulevaa prosessointia varten.

2.1.1. Eroosio ja dilaatio

Gonzalesin ja Woodsin mukaan eroosio ja dilaatio ovat morfologisia operaatioita kuvan pikseleille [1 s.523-524]. Toisin sanoen binäärinen eroosio poistaa kuvasta informaatiota määrittelemällä uudelleen pikseleiden harmaasävyarvot ennalta määritellyn rakenne-elementin perusteella. Itse operaatio tapahtuu asettamalla rakenne-elementti tarkasteltavan pikselin “päälle”. Jos kyseisen pikselin naapuruston arvot vastaavat rakenne-elementin arvoja, pikseli saa arvon yksi.

Dilaatiossa puolestaan kaikki pikselit, joiden arvo on yksi, käydään läpi ja niiden naapureiden arvot asetetaan vastaamaan rakenne-elementin B arvoja. Lisäksi tarkasteltavan pikselin arvo asetetaan rakenne-elementin origon arvoksi.

Eroosiota voidaan soveltaa kuvassa olevien pienten yksityiskohtien poistamiseen. Kun haluttu määrä eroosio-operaatioita on suoritettu, kuva käsitellään samalla määrällä dilaatioita informaation palauttamiseksi [1 s.528]. Morfologista eroosiota ja dilaatiota voidaan soveltaa binäärikuvan lisäksi myös harmaasävykuvalle [1 s.550].

2.1.2. Tasoittava spatiaalinen suodatus

Spatiaalisessa suodatuksessa ennalta määriteltyä maskia sovelletaan jokaiselle kuvan pikselille [1 s.116]. Maski voi perustua esimerkiksi keskiarvon laskemiseen tai Gaussin-funktioon. Tasoittavan suodatusta käytetään esikäsittelyyn, jolloin sillä sumennetaan tarpeettomia yksityiskohtia, ja kohinan poistamiseen [1 s.119].

2.2. Segmentointi

Segmentoinnissa käsiteltävä kuva jaetaan pienempiin alueisiin, esimerkiksi reunoihin [1 s.567, s.572]. Reunojen erottaminen on mielekästä erityisesti silloin kun kiinnostuksen kohteena on kuvassa olevien esineen muoto. Reunantunnistusta hyödynnetään esimerkiksi kasvojentunnistuksessa [2]. Reunantunnistuksen lisäksi kuvia voidaan segmentoida kaariviivojen jäljittämisellä (contour tracing) ja kynnystämisellä.

2.2.1. Reunantunnistus Cannyn algoritmilla

Cannyn reunantunnistus-algoritmi koostuu useasta vaiheesta ja mahdollistaa useiden intensiteetin reunojen tunnistamisen [3].

Seuraavaksi kuvataan algoritmin vaiheet perustuen Cannyn vuonna 1986 julkaistuun artikkeliin sekä OpenCV:n dokumentaatioon [3, 4]:

1. **Kuvan tasoittaminen.** Tasoittaminen tapahtuu Gauss-suodatuksella, joka vähentää yksityiskohtien ja kohinan määrää kuvasta. Käsittely suodattimella vähentää siis tunnistettavien reunojen määrää.
2. **Gradienttivektoreiden löytäminen.** Gradienttivektorit tunnistaan kuvasta Sobelin maskien [1 s.578] avulla, joiden perusteella saadaan pysty-ja vaakasuuntaisten ensimmäisten derivaattojen approksimaatio. Approksimaatioita käytetään pikselin gradientin magnitudin ja suunnan laskemiseen. Magnitudia voidaan myös approksimoida
3. **Ei-maksimikohtien vaimentaminen (Non-maximum suppression).** Jokainen kuvan pikseleiden gradientin suunnan mukainen tai vastakkainen pikseli vaimennetaan, kun se ei ole paikallinen maksimi. Lopputuloksena löydetyt reunat ohenevat.
4. **Kaksivaiheinen kynnystäminen.** Kaksivaiheisella kynnystämisellä määritellään mitkä tähän mennessä löydetyistä reunoista hyväksytään lopullisiksi reunoiksi. Gradientin magnitudeille määritetään kaksi rajaa, *A* ja *B*: Jos pikselin magnitudi ylittää rajan *A*, se merkitään reunaksi. Jos magnitudin arvo puolestaan on alle rajan *B*, arvo asetetaan nollaksi. Väliin jäävät pikselit merkitään “heikoiksi reunoiksi”.
5. **Reunojen jäljittäminen.** Jos edellisessä vaiheessa määritellyt heikot reunat ovat yhteydessä varsinaisiin reunoihin, jätetään ne näkyville, muuten ne poistetaan kuvasta.

2.2.2. Kaariviivojen tunnistus (contour/boyndary tracing)

Kaariviiva (contour) on lista pisteistä, jotka määrittävät suljetun käyrän/kaariviivan [5]. Pisteiden ei tarvitse olla vierekkäisiä, vaan kaariviivaa voidaan myös approksimoida sopivasti valikoiduilla pisteillä.

Ääriviivojen tunnistamiseen on useita eri algoritmeja, muun muassa myös Suzukin algoritmi, johon OpenCV-kirjaston käyttämä ääriviivojen etsimiseen tarkoitettu findContours-metodi perustuu [6, 5]. Cannyn reunantunnistus algoritmia voidaan käyttää ääriviivojen tunnistamisen esikäsittelynä [5].

2.2.3. Kynnystäminen värin perusteella

Värikuvien rakenne eroaa harmaasävykuvista, koska värin esittäminen vaatii enemmän informaatiota. Tästä syystä värikuvat ovat yleensä harmaasävykuvia suurempia. Ehkä tunnetuin värimalli on RGB-värimalli [1 s.290], jolla värit ilmaistaan lisäämällä toisiinsa kolmea eri pääväriä: punaista (Red), vihreää (Green) ja sinistä (Blue). Päävärien lisäksi on mahdollista käyttää myös Alpha-kanavaa, joka

ilmoittaa värin läpinäkyvyyden (opacity). Tässä tapauksessa värimallia kutsutaan RGBa-värimalliksi. RGB:tä käytetään erityisesti värien ilmaisuun digitaalisille laitteille, jota varten se on myös suunniteltu.

Värien ilmaiseminen onnistuu myös HSV-värimallilla [7], joka koostuu hue-, saturation- ja value-arvoista. Nämä arvot ilmaisevat vastaavasti värin sävyn, kylläisyyden ja tummuuden.

Harmaasävykuvien lisäksi myös RGB- ja HSV-kuvat on mahdollista kynnystää binäärikuviksi. RGB-kuvan tapauksessa punaisen, vihreän ja sinisen sävyille määritetään rajat, kun taas HSV-kuvassa värin säätely tapahtuu rajaamalla sävyn arvot. Jos pikselin arvo on määriteltyjen rajojen sisällä, se asetetaan ykköseksi. Muulloin pikseli asetetaan nollaksi.

2.3. Hahmontunnistus piirteiden perusteella

Hahmon tunnistaminen voidaan toteuttaa myös piirteiden perusteella [Szeliski 8 s.205 - 256] Tällöin kuitenkin tarvitaan kaksi kuvaa, joita verrataan keskenään. Szeliskin mukaan prosessi koostuu kolmesta askeleesta: piirteiden havaitsemisesta, kuvaamisesta ja yhteensovittamisesta. Havaitsemisessa kuvasta etsitään alueita, jotka voidaan tunnistaa hyvin myös muista kuvista, joissa esiintyy samankaltainen alue. Alueet voivat olla esimerkiksi pisteitä tai reunoja.

Kuvaamisessa löydetyt pisteet/alueet pyritään muuttamaan stabiilimpaan muotoon. Vaihe pyrkii mahdollistamaan alueiden yhdistämisen, esimerkiksi tunnistettavan esinen koon, suunnan ja valotuksen muutoksista riippumatta. Tunnistaminen on mahdollista ilman tätä vaihetta, mutta deskriptoreiden muodostaminen parantaa tunnistamista [8 s.222].

Yhteensovittamisessa (matching) pyritään löytämään molemmista kuvista samankaltaiset deskriptorit. Vaihe voidaan suorittaa esimerkiksi vertailemalla molempien kuvien kaikkia deskriptoreita toisiinsa jokaista yhteensovitettavaa kuvaparia kohden [8 s.254] tai FLANN-pohjaisilla [9] algoritmeilla. Vaihtoehtoisesti voidaan myös etsiä tunnistettavan kuvan piirteitä ainoastaan käsiteltävän kuvan piirteiden läheisyydestä koko kuvan sijasta. Hakuavaruuden rajoittaminen on mahdollista erityisesti objektia jäljitettäessä videokuvasta, jolloin objektin edellinen sijainti tiedetään. Aliluvut 2.3.1 – 2.3.6 tarkastelevat algoritmeja, jotka suorittavat osan piirteisiin perustuvan hahmontunnistuksen kolmesta vaiheesta.

2.3.1. Kulmantunnistus

Harrisin kulmantunnistin (Harris corner detector) on ensimmäisiä piirteisiin perustuvan tunnistuksen menetelmiä vuodelta 1988 [10]. Harrisin mukaan kulmat ovat kuvan alueita, joiden intensiteetti vaihtelee suuresti jokaiseen suuntaan. Shi-Tomasi kulmantunnistin tuottaa parempia tuloksia Harrisin kulmantunnistimeen nähden [11]. Edellä mainittujen kulmantunnistusmenetelmien ongelmana on koon huomioon ottaminen: sama kuva erilaisilla resoluutioilla tuottaa erilaisia kulmia [8 s.216].

2.3.2. SIFT

Lowen SIFT-algoritmi (Scale-Invariant Feature Transform) [12] pyrkii huomioimaan kulmien/corners mittakaavan (scale) muuttumisen rotaation lisäksi. SIFT-algoritmin lähestymistapa mittakaavaongelmaan on useiden eri kokoisten ikkunafunktioiden määrääminen.

Seuraavaksi esitellään algoritmin vaiheet Lowen alkuperäisen algoritmin julkaisun perusteella [12]:

1. **Scale-space extrema detection.** Difference-of-Gaussian-funktiota hyödyntämällä etsitään mahdolliset avainpisteet, jotka ovat invariantteja koon ja orientaation muutoksille.
2. **Keypoint localization.** Avainpisteitä (keypoints) karsitaan niiden stabiiliuden perusteella.
3. **Orientation assignment.** Jokaiselle avainpisteelle määritetään orientaatio paikallisten gradienttien perusteella. Kaikki tulevat operaatiot suoritetaan suhteessa määrättyyn orientaatioon ja mittakaavaan (scale), jolloin saavutetaan invarianttius näihin muunnoksiin nähden.
4. **Keypoint descriptor.** Paikalliset kuvan gradientit mitataan valitulla mittakaavalla avainpisteiden ympäriltä ja ne muunnetaan kestämään paremmin muodon ja valaistuksen vaihteluita.

2.3.3. SURF

Bay H., Tuytelaars T. and Van Gool L kehittivät SURF-algoritmin (Speeded-Up Robust Features) [13] vaihtoehdoksi SIFT-algoritmille. Erona on erityisesti Difference of Gaussian-suodatuksen korvaaminen keskiarvo-suodatuksella ja aallokkeiden hyödyntäminen orientaation määrittämiseen.

Panchal:in ja Shah:in mukaan SURF-algoritmi on SIFT-algoritmia nopeampi [14]. SIFT kykenee kuitenkin tunnistamaan enemmän avainpisteitä.

2.3.4. FAST

Rosten ja Drummond esittävät FAST–piirteiden havaitsijan (feature detector) [15]. FAST (Features from Accelerated Segment Test) on suunniteltu erityisesti realiaikaisiin soveluksiin, jossa ruudunpäivitysnopeus on tärkeä tekijä.

Seuraavaksi luetellaan algoritmin vaiheet julkaisun [15] perusteella:

1. Olkoon intensiteetti *I*. Valitaan kuvasta pikseli ja tarkastellaan sen kelpaavuutta avainpisteeksi.
2. Valitaan kynnysarvo *t*.
3. Valitaan 16-pikselinen kehä tarkasteltavan pikselin ympäriltä.
4. Tarkasteltava pikseli määritetään kulmaksi, jos löydetään *n* kappaletta pikseleitä kuudentoista pikselin kehästä, jotka ovat kirkkaampia kuin $I + t$ tai tummempia kuin $I - t$.

Neljännessä vaiheessa voidaan käyttää myös vaihtoehtoisesti nopeampaa testiä, joka huomioi ainoastaan pikselit (x, y + 3), (x - 3, y), (x + 3, y) ja (x, y - 3) kun pikselin koordinaatit ovat x ja y. Testissä hyödynnetään koneoppimista ja ei-maksimikohtien vaimentamista (Non-maximal Suppression).

2.3.5. BRIEF

Calonderin ja Lepetitin BRIEF-algoritmi (Binary Robust Independent Elementary Features) kattaa deskriptoreiden muodostamisen ja niiden yhteensopivuuden testaaminen (matching) [16]. Avainpisteiden etsimiseen täytyy käyttää esimerkiksi SIFT- tai SURF-algoritmeja. BRIEF-algoritmin taustalla on deskriptoreiden koon pienentäminen ja tehokas yhteensopivuuden testaaminen(matching). Deskriptoreiden koon pienentäminen onnistuu myös pakkaamalla: esimerkiksi SIFT-deskriptoreiden kokoa on mahdollista pienentää käyttäen pääkomponenttianalyysia [17].

Jokaisesta löydetystä avainpisteestä muodostetaan binäärinen merkkijono. Tämä tapahtuu ensiksi valitsemalla avainpisteen naapurustosta n kappaletta paikkapareja (location pairs). Muuttuja n siis määrää syntyvän deskriptorin dimensiot ja se voi olla esimerkiksi 128, 256 tai 512. Itse deskriptori muodostetaan vertailemalla jokaisen parin intensiteettiarvoja I: jos $I(p) > I(q)$ niin deskriptorin kyseiseksi arvoksi asetetaan yksi, muussa tapauksessa nolla, kun p ja q ovat paikkaparin pisteet [16].

Deskriptoreiden yhteensopivuuden testaaminen tapahtuu laskemalla vertailtavien deskriptoreiden Hamming-etäisyyksiä. BRIEF:n mukaien deskriptoreiden muodostaminen ja yhteensopivuuden testaaminen on algoritmin kehittäjien mukaan SIFT- ja SURF algoritmeja tehokkaampaa. BRIEF:n heikkoutena on kuitenkin huono rotaation sietokyky [16].

2.3.6. ORB

Rubleen ja Rabaudin ORB-algoritmi (Oriented FAST and Rotated BRIEF) käyttää FAST-algoritmia avainpisteiden löytämiseen, Harrisin kulmantunnistinta löytyneen joukon karsimiseen ja kuvapyramidia moniskaalaisten piirteiden tuottamiseen [18]. Näiden vaiheiden jälkeen lasketaan intensiteetin keskipisteet avainpisteille momentteja hyödyntäen, jonka perusteella voidaan laskea niiden orientaatio. Deskriptorin muodostaminen tapahtuu samalla tavalla kuin BRIEF-algoritmilla mutta binääritestissä otetaan huomioon myös lasketut orientaatiot rotaatio-invarianttiuden saavuttamiseksi. ORB pyrkii siis yhdistämään FAST- ja BRIEF-algoritmien parhaat puolet ja samalla huomioimaan kuvan rotaation.

2.3.7. Algoritmien vertailua

Edellä esitellyt algoritmit vaihtelevat suorituskyvyltään ja tarkkuudeltaan. Jeong ja Moon esittävät FAST-pohjaista algoritmia verrattuna SIFT- ja SURF algoritmeihin, joiden ongelmaksi muodostui riittävä suorituskyky mobiilialustalla [19]. Tutkimus on kuitenkin julkaistu vuonna 2011, jonka jälkeen mobiililaitteiden suorituskyky on parantunut merkittävästi.

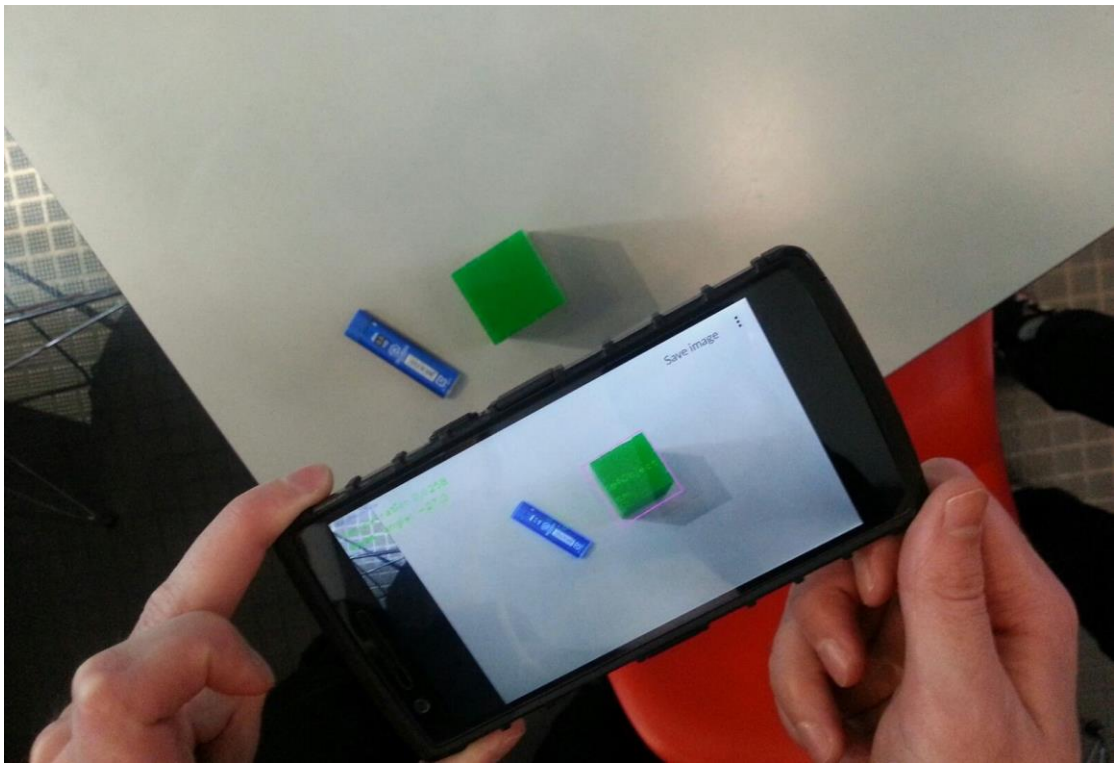
Saipullah myös pitää FAST algoritmia suorituskykyisempänä vaihtoehtona vertailujoukostaan [20]. ORB-algoritmin he toteavat robustisimmaksi mutta myös liian raskaaksi Android-älypuhelimelle. Kyseinen tutkimus on julkaistu vuonna 2013.

Thakker ja Kapadia puolestaan testasivat OpenCV-kirjaston algoritmeja ja totesivat, että Androidia on mahdollista yhdessä OpenCV:n kanssa käyttää konenäköjärjestelmien toteuttamiseen [21]. Tutkimus on julkaistu vuonna 2015.

Edellä mainittujen tutkimusten pohjalta vaikuttaa siltä, että ORB- ja FAST-algoritmit ovat todennäköisesti parhaat vaihtoehdot referenssiesineen määrittämiseen piirteiden perusteella.

3. RATKAISUN KUVAUS

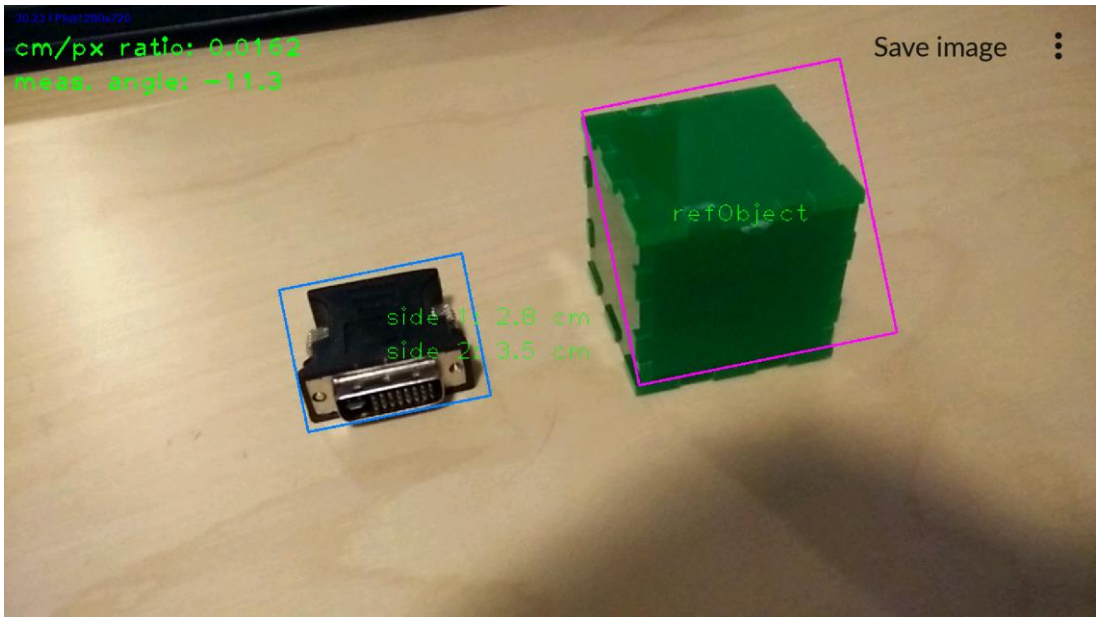
Mittaussovellus toteutettiin Java-kielellä Android-alustalle. Referenssiesineen ja mitattavan esineen tunnistamiseen käytettiin avoimen lähdekoodin OpenCV-konenäkökirjastoa. Sovelluksesta toteutettiin kaksi versiota: Ensimmäinen versio rakennettiin mahdollisimman nopeasti täyttämään mittaustapahtuman minimivaatimukset. Toisen version tarkoituksena oli parantaa tunnistustarkkuutta ja käyttöliittymää. Lisäksi toisessa versiossa on korjattu ensimmäisessä versiossa ilmenneitä ohjelmointivirheitä. Esimerkki sovelluksen käyttötilanteesta ilmenee kuvassa 1.



Kuva 1. Esimerkki sovelluksen käyttötilanteesta.

3.1. Rajoitukset

Jotta sovellus toimisi, referenssiesineen täytyy olla kuvassa, jolloin käyttäjän täytyy se siihen asettaa. Lisäksi referenssiesineen tulee olla suunnilleen samassa tasossa mitattavan kohteen kanssa, muuten mittaustulokset vääristyvät. Myöskin kuvauskulman muutos kohtisuorasta vääristää mittaustuloksia, mikä on nähtävissä kuvasta 2. Kamera ei myöskään saa olla liian kaukana mitattavasta esineestä, jotta tunnistus onnistuu.



Kuva 2. Referenssiesineen mitat vääristyvät kulman vaikutuksesta.

Android-älypuhelimia valmistetaan useilla eri komponenteilla ja niitä on valmistettu jo useita vuosia. Siksi niiden suorituskyky voi vaihdella hyvinkin paljon. Matalan suorituskyvyn laitteissa kuvien prosessointi voi viedä liikaa resursseja, jolloin sovellusta ei voi käyttää. Lisäksi älypuhelimessa on oltava kamera ja sovellus tarvitsee siihen käyttöoikeudet.

3.2. Android

Perustana Android-alustalle on Linux-ydin, joka suorittaa matalan tason toiminnot, kuten muistin hallinnan [22]. Ydin on yhteydessä HAL (Hardware Abstraction Layer)-kerrokseen, joka mahdollistaa matalan tason ominaisuuksien, esimerkiksi, kameran ja näytön, käyttämisen Java-rajapinnan välityksellä. Itse sovellukset suoritetaan omina prosessinaan ART (Android Runtime) - ympäristössä.

Android-sovellusten kehittäminen tapahtuu pääasiassa Java-ohjelmointi kielellä [23]. Jotta sovelluksen voisi ajaa, Android Software Development Kit kääntää lähdekoodista, muusta datasta ja resursseista älypuhelimelle asennettavan APK – tiedoston (Android Package) [23].

Javan lisäksi on mahdollista myös käyttää C- ja C++-ohjelmointikieliä sovellusten kehittämiseen Native Development Kitin (NDK) välityksellä [24]. NDK:lla käännetään C- tai C++-kooditiedostot natiivikirjastoksi, joka sitten liitetään asennettavaan APK-tiedostoon. Natiivikirjaston C++-funktioita on mahdollista käyttää Java Native Interface:n (JNI) kautta [25]. Intensiivistä laskentaa vaativissa tilanteissa, esimerkiksi fysiikkamallinnuksissa, voidaan natiivikoodilla suorituskykyä mahdollisesti parantaa [24].

Android sovellukselle on olemassa muitakin toteutusvaihtoehtoja Java-, C- ja C++-kielten lisäksi: muun muassa Javasricpt-pohjaiset React Native [26] ja Phonegap [27]. Päätimme kuitenkin käyttää OpenCV-kirjastoa konenäkö- ja kuvankäsittelyelementtien toteuttamiseen, jotka ovat sovelluksemme kannalta

keskeisessä roolissa [28]. Koska OpenCV:llä ei ole virallista Javascript-rajapintaa [28], hylkäsimme Javascriptiin perustuvan Android-sovelluksen toteuttamisen. Lisäksi OpenCV kirjasto tarjoaa kattavan dokumentaation ja esimerkkejä javapohjaiselle Android-kehittämiselle. Näistä syistä valitsimme sovelluksen toteutustavaksi tavanomaisen Android-sovelluksen rakenteen Java-kieltä hyväksikäyttäen. Valintaa tuki ryhmän jäsenten kokemus olio-ohjelmoinnista, vaikka kukaan jäsen ei projektia aikaisemmin ollut työskennellyt javapohjaisen Android-sovelluksen parissa.

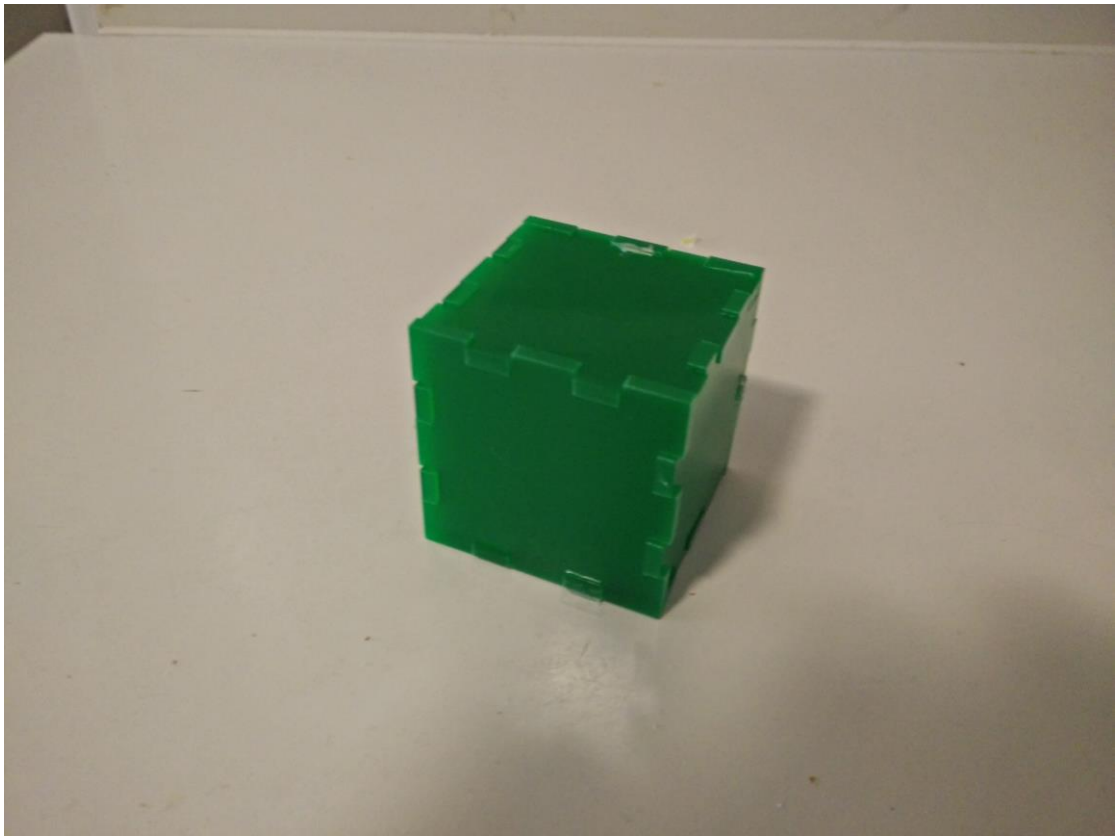
3.3. OpenCV-kirjasto

Sovellukseen liittyy olennaisena osana kuvankäsittely ja konenäkö. Esineen tunnistamiseen käytettiin OpenCV-kirjastoa, joka on avoimen lähdekoodin konenäkökirjasto usealle eri alustalle [28]. Kirjasto tarjoaa C, C++, Java, Python ja MATLAB rajapinnat, jotka mahdollistavat yli 2500 kuvankäsittely- ja konenäkö-prosessia. Kirjastolla on myös kattava dokumentaatio ja useita eri käyttöesimerkkejä, jotka ovat kirjaston käytön oppimisen kannalta oleellinen asia.

Käyttämämme kirjasto koostuu useista eri moduuleista, joista sovelsimme lähinnä Android-, Core-, Featureds2d- ja Imgproc-paketteja/moduuleja [29].

3.4. Ohjelmiston ensimmäinen versio

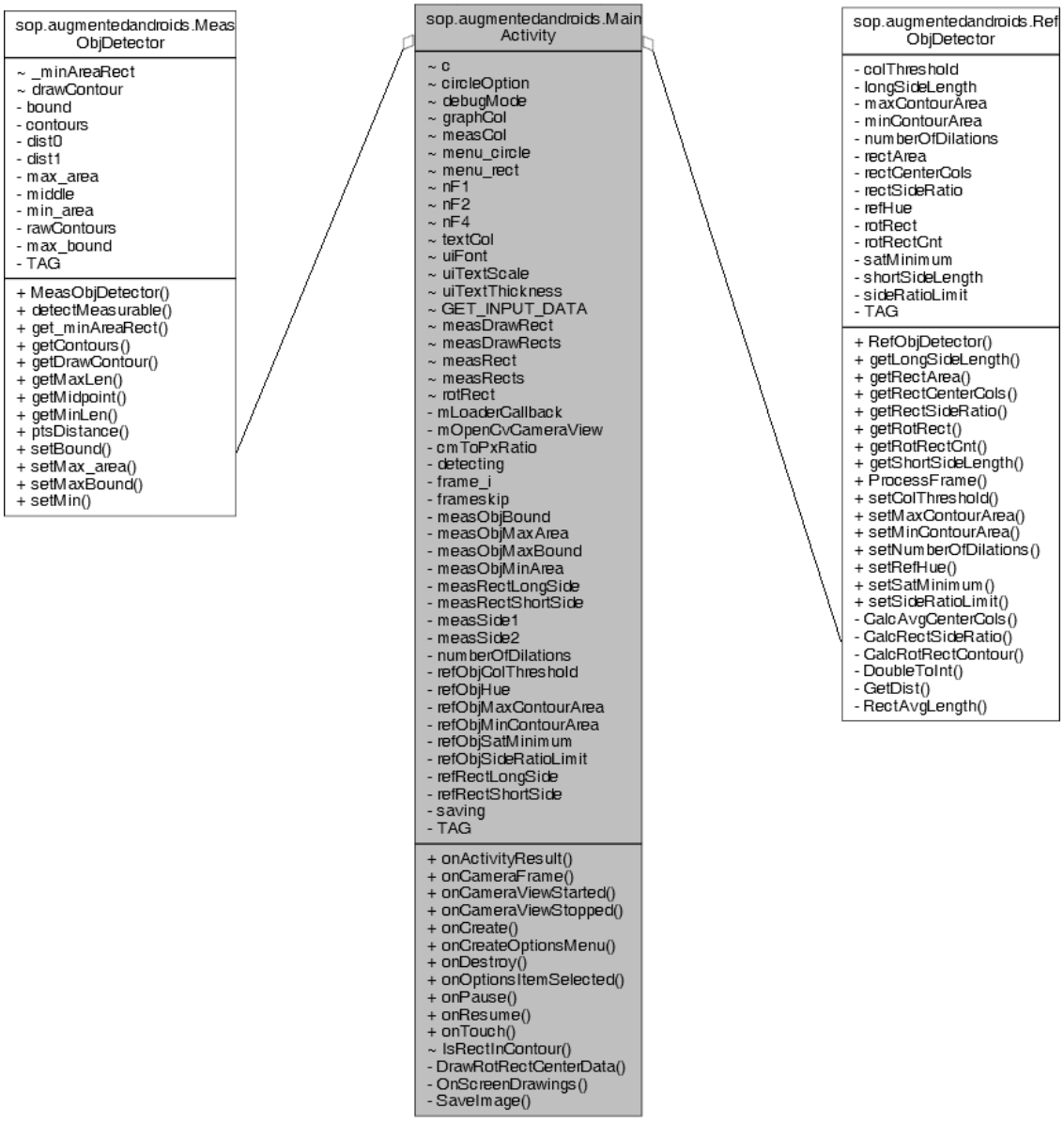
Alun perin tarkoituksena oli antaa käyttäjälle mahdollisuus määrittää referenssiesine itse. Ensimmäisessä versiossa tätä ominaisuutta ei kuitenkaan toteutettu, vaan aluksi päätettiin keskittyä sovelluksen perusominaisuuksiin eli esineiden tunnistamiseen ja mittaamiseen. Referenssiesineeksi määritimme vihreän 5cm x 5cm x 5cm kuution, joka tunnistetaan väri-informaation perusteella. Väri valittiin vihreäksi, koska sitä ei esiinny yhtä paljon kuin esimerkiksi valkoista tai mustaa. Referenssiesine näkyy kuvassa 3.



Kuva 3. Referenssiesine.

3.4.1. Luokkakaavio

Kuva 4 esittelee sovelluksen luokkakaavioon. MeasObjDetector-luokka toteuttaa mitattavan esineen tunnistamiseen ja RefObjDetector-luokka referenssiesineen tunnistamiseen vaadittavat metodit. Näiden luokkien oliot luodaan MainActivityyn, jonka tehtävänä on hoitaa kameranäkymän prosessointi ja mittojen päivittäminen.



Kuva 4. Sovelluksen luokkakaavio.

3.4.2. Referenssiesineen tunnistus

Referenssiesineen tunnistusalgoritmi:

1. Kameran kuva muutetaan harmaasävykuvaksi.
2. Harmaasävykuva kynnystetään binäärikuvaksi ja kuvasta etsitään reunat. Tämä tapahtuu Cannyn algoritmilla.
3. Suoritetaan numberOfDilations-parametrin mukainen määrä dilaatioita reunojen yhtenäistämiseksi, jotta seuraava vaihe onnistuu.
4. Etsitään kuvasta yhtenäiset alueet OpenCV:n findCountours-metodilla.
5. Silmukassa käsitellään kaikkien yhtenäiset suljetut ääriviivat (contours):
    a. Lasketaan niiden momentti.

b. Momentista lasketaan käsiteltävän alueen keskipiste.
c. Keskipisteen lähialueelta lasketaan keskimääräiset värisävy- ja kylläisyysarvot.
d. Jos kohdassa c laskettu värisävy on ennalta määriteltyjen sävyrajojen sisällä, tässä tapauksessa vihreä, lasketaan pikselin pituus (cm/pikseleitä) lyhempää referenssiesineen sivua käyttäen ja piirretään mitattavan esineen ääriviivat ruudulle
e. Jos löydetään suurempi alue, joka sopii määritellylle värisävyvälille, se päivitetään ruudulle.

3.4.3. Mitattavan esinen tunnistus

Mitattavan esineen tunnistusalgoritmi:

1. Sisään tuleva kuva muunnetaan harmaasävykuvaksi.
2. Kuva kynnystetään binäärikuvaksi.
3. Kuvasta etsitään findContours-metodilla yhtenäiset alueet (contours).
4. Käydään edellisessä vaiheessa löydetyt alueet läpi, lasketaan niiden keskipisteet ja näytetään suurin.

Mitattavan esineen tunnistamisessa tarkkuus voisi parantua suodattamalla kohinaa prosessin aluksi esimerkiksi Gauss-suodatuksella. Tätä kokeiltiin ja todettiin, että tarkkuuden parantuminen oli pientä suhteessa suorituskyvyn heikentymiseen, jonka vuoksi kyseistä suodatusta ei käytetty.

3.4.4. Säädettävät parametrit

Referenssiesineen ja mitattavan kohteen tunnistusalgoritmit vaativat useita ennalta määriteltyjä parametreja toimiakseen. Koska parametrien määrittäminen on kokeellista, päätimme toteuttaa käyttöliittymän niiden säätämiseen ohjelman suorituksen aikana.

Taulukko 1. Referenssiesineen parametrit.

Parametri	Selitys
refObjHue	Värisävyn arvo
refObjColThreshold	Värisävyn kynnysraja. Esim. Jos 10 ja refObjHue 50, niin intervalli hyväksytyille sävyille on 40 - 60.
refObjSatMinimum	Minimi kylläisyyden raja.
numberOfDilations	Dilaatioiden lukumäärä Cannyn reunantunnistuksen

	jälkeen.
refObjMinContourArea	Minimi pinta-ala referenssiesineelle (pikseleinä)
refObjMaxContourArea	Maksimi pinta-ala referenssiesinelele (pikseleinä)
refObjSideRatioLimit	Maksimi kuvasta referenssiesineen sivujen suhde. Muuttuu referenssiesinettä tarkasteltavan kulman muuttuessa.

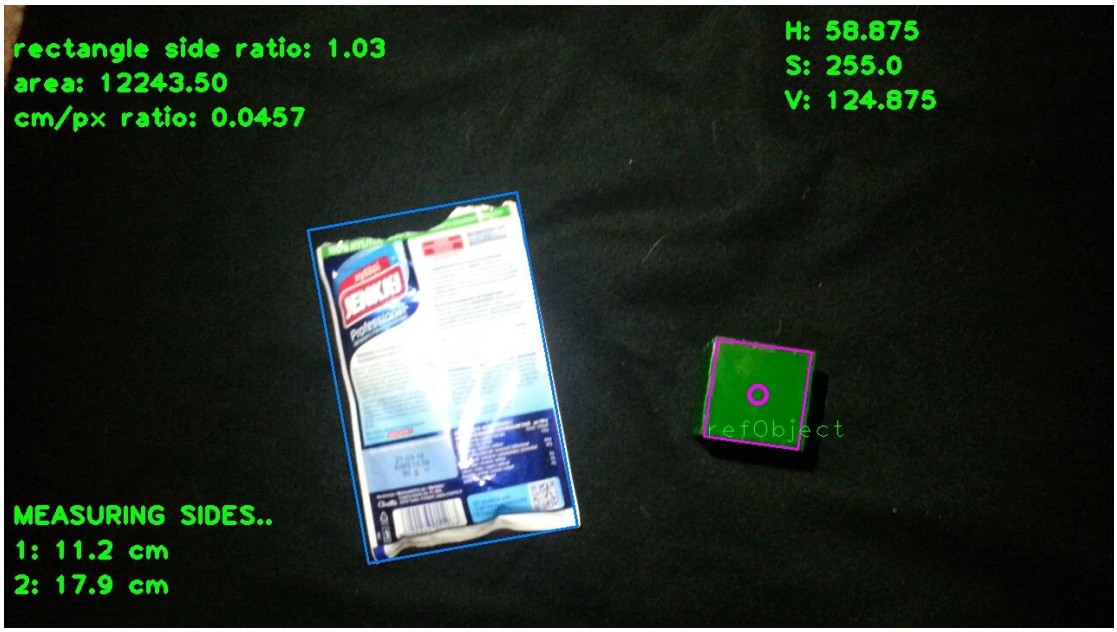
Taulukko 2. Mitattavan esineen parametrit.

Parametri	Selitys
measObjBound	Minimi kynnysarvo binäärikynnystämiselle
measObjMaxBound	Maksimi kynnysarvo binäärikynnystämiselle
measObjMinArea	Minimi pinta-ala mitattavalle esineelle (pikseleinä)
measObjMaxArea	Maksimi pinta-ala mitattavalle esineelle (pikseleinä)

Taulukoissa 1 ja 2 mainittujen parametrien lisäksi sovelluksella on vielä frameSkip-parametri, joka ilmoittaa kuinka monta tulevista kameran kuvista prosessoidaan ajanjaksossa. Esimerkiksi jos frameSkip on arvoltaan 5, niin algoritmi suoritetaan vain joka kuudennelle saapuvalle kameran kuvalle. Muuttuja on tarpeellinen, koska sen avulla voidaan helposti säädellä sovelluksen kuormitusta laitteelle.

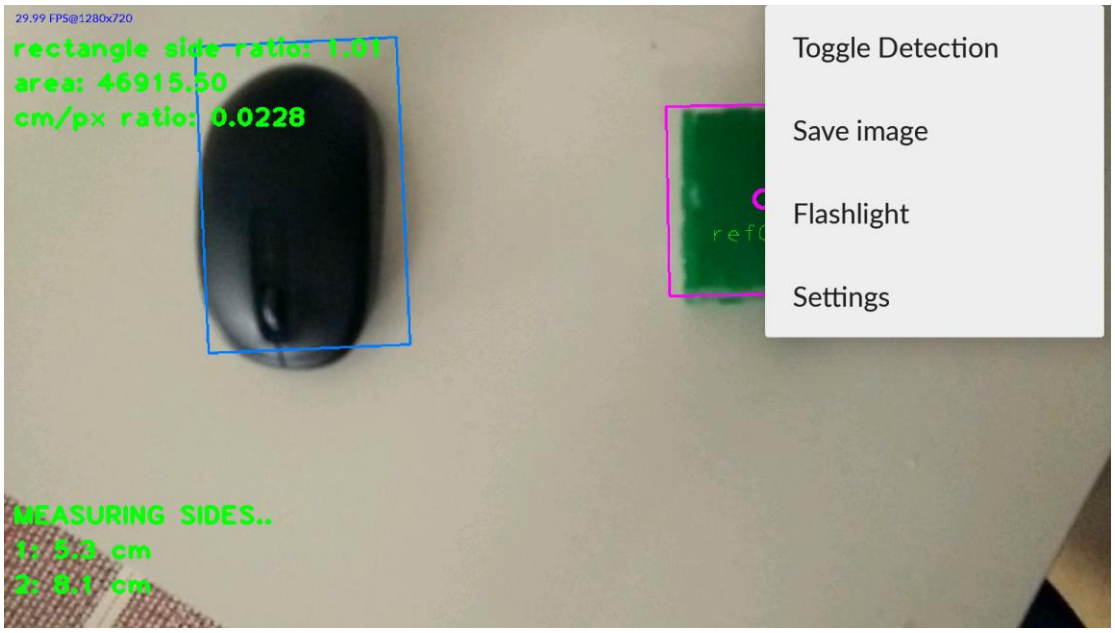
3.4.5. Käyttöliittymä

Kuva 5 näyttää päänäkymän sovelluksen käyttöliittymästä. Referenssiesineen reunat korostetaan pinkillä ja mitattavan esineen reunat sinisellä. Referenssiesineen HSV-arvot näkyvät oikeassa yläkulmassa. Mitattavan esineen sivut näkyvät vasemmassa alakulmassa. Vasemmasta yläkulmasta voi nähdä referenssiesineen sivujen suhteen, mitattavan esineen pinta-alan ja lisäksi pikselin pituuden.

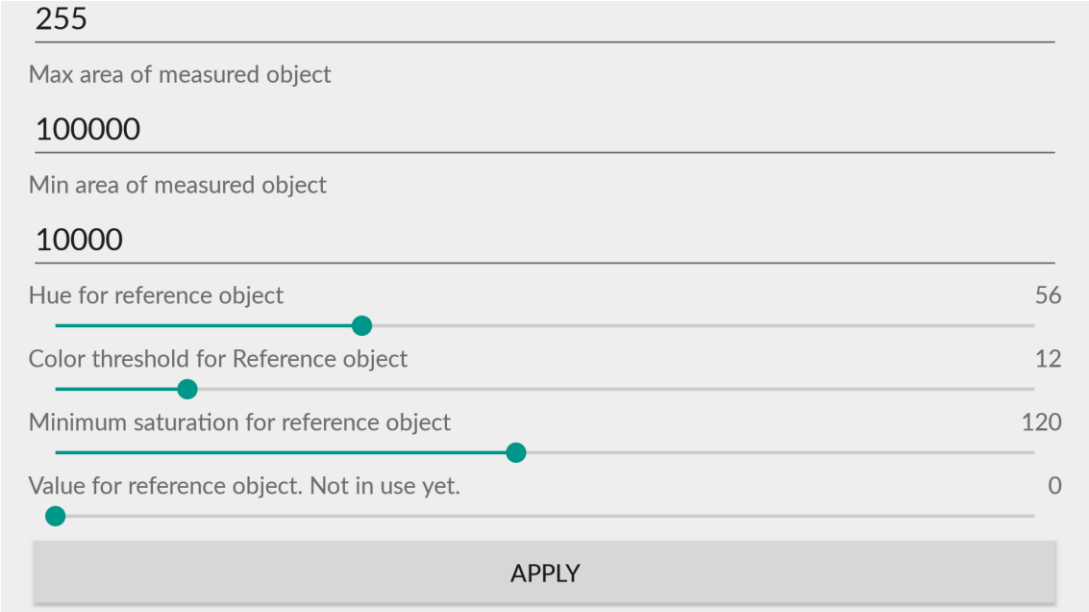


Kuva 5. Sovelluksen päänäkymä.

Kuvassa 6 näkyy sovelluksen päävalikko. ”Toggle Detection”–valinta sammuttaa ja käynnistää tunnistuksen, ”Save image”- tallentaa kameran kuvan ja ”Settings” aukaisee valikon parametrien säätämiseen, joka on nähtävissä kuvassa 7. Flashlight-ominaisuutta ei toteutettu.



Kuva 6. Menu päänäkymän päällä.



Kuva 7. Käyttöliittymä parametrien säätämiseen.

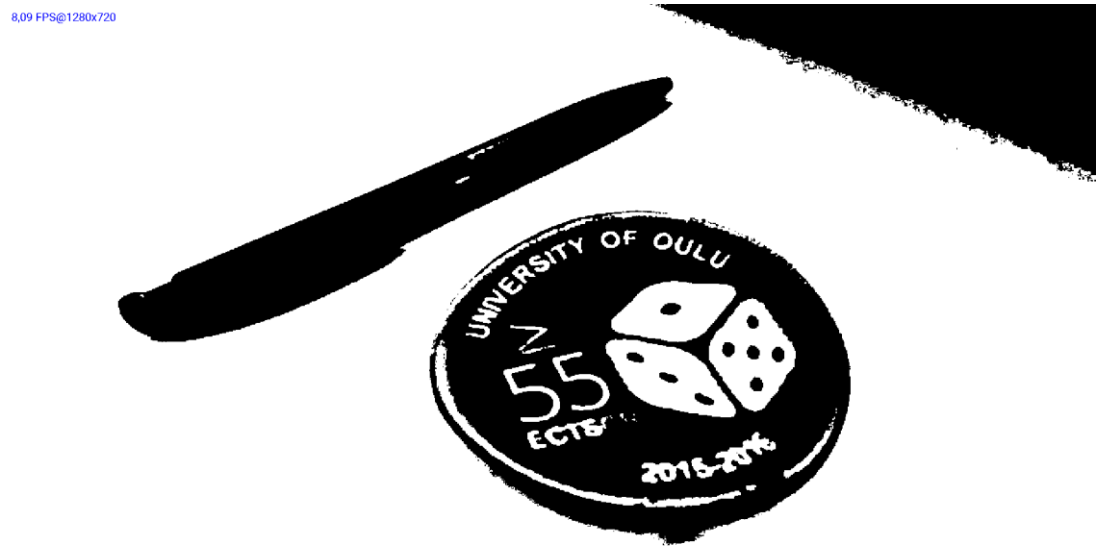
3.5. Toisen version muutokset

Toisessa versiossa parannettiin referenssiesineen tunnistusalgoritmia suodattamalla sisään tuleva kuva värisävyn perusteella ja lisäämällä eroosio-operaatio. Referenssiesineen tunnistusalgoritmin muutosten tarkoituksena oli varjojen eliminointi.

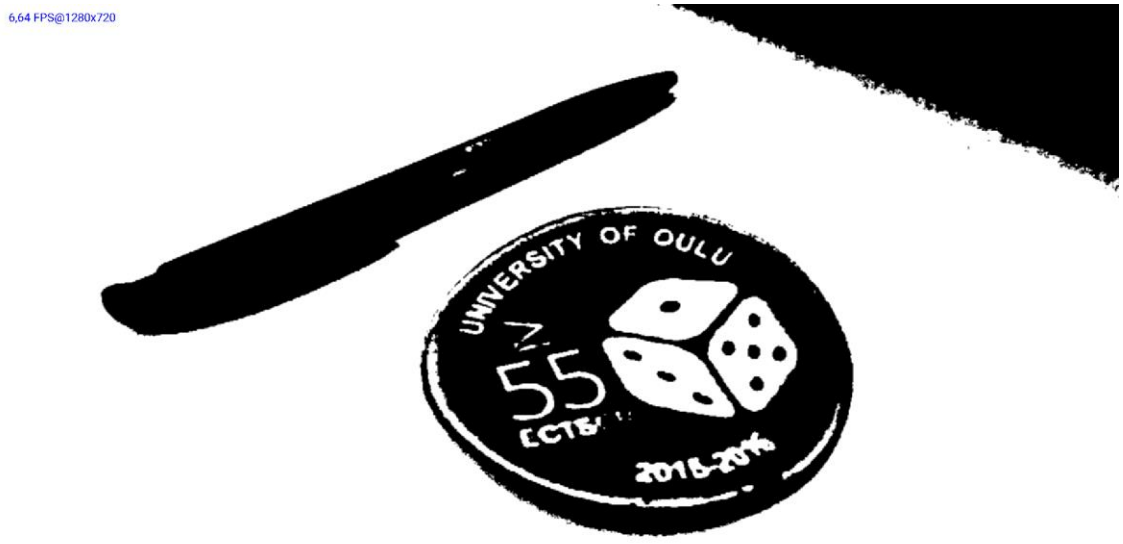
Uusittu referenssiesineen tunnistusalgoritmi:

1. Kameran kuvasta tehdään kaksi kopiota: harmaasävykuva ja HSV-väriavaruuteen muunnettu kuva.
2. HSV-kuvasta poistetaan kaikki kohdat, joiden värisävy ei ole lähellä referenssiesineen sävyä.
3. Tehdään eroosio-operaatio harmaasävykuvalle.
4. Harmaasävykuva kynnystetään binäärikuvaksi ja kuvasta etsitään reunat Cannyn algoritmilla.
5. Suoritetaan numberOfDilations-parametrin mukainen määrä dilaatioita reunojen yhtenäistämiseksi, jotta seuraava vaihe onnistuu.
6. Etsitään kuvasta yhtenäiset alueet openCV:n findCountours-metodilla.
7. Silmukassa käsitellään edellisessä vaiheessa löydetyt alueet:
    a. Lasketaan niiden momentti.
    b. Momentista lasketaan käsiteltävän alueen keskipiste.
    c. Keskipisteen lähialueelta lasketaan keskimääräiset värisävy- ja kylläisyysarvot.
    d. Jos kohdassa c laskettu värisävy on ennalta määriteltyjen sävyrajojen sisällä, tässä tapauksessa vihreä, lasketaan pikselin pituus (cm/pikseliä) ja piirretään esineen ääriviivat ruudulle
    e. Jos löydetään isompi alue, joka sopii määritellylle värisävyvälille, alue päivitetään ruudulle.

Mitattavan esineen tunnistaminen sen sijaan säilyi muuten samana mutta kohinan pienentämiseksi harmaasävykuva käsitellään toisessa versiossa keskiarvosuodattimella. Kuvassa 8 nähdään, että keskiarvosuodatus selkiyttää haalarimerkin reunoja, jolloin se on helpompi tunnistaa.

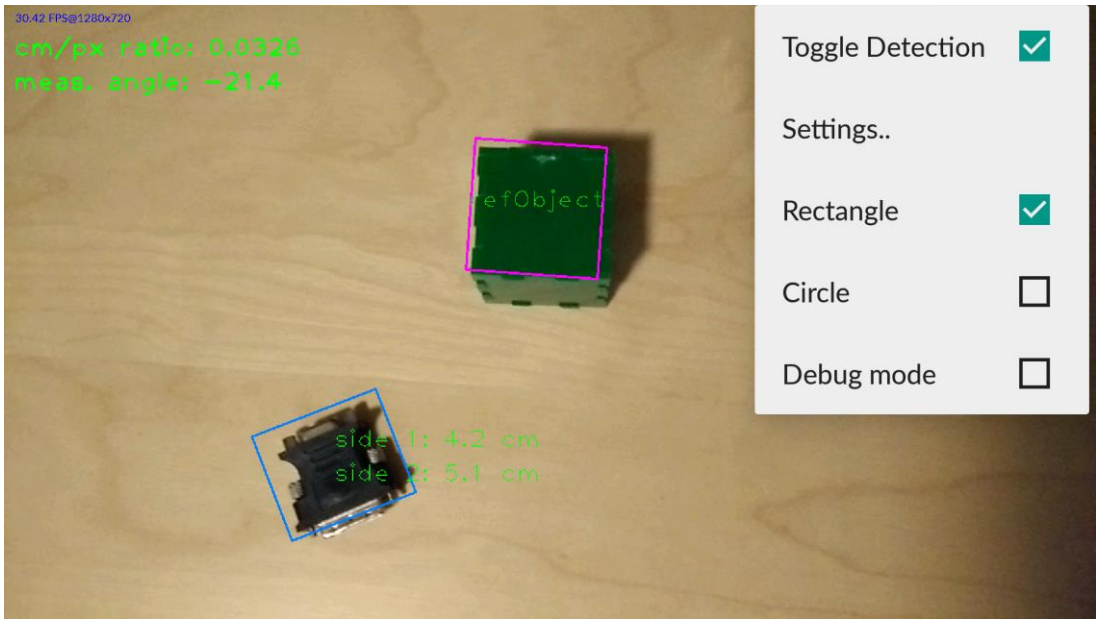


Kuva 8. Kynnystetty kameran kuva kynästä ja haalarimerkistä.



Kuva 9. Keskiarvosuodatettu kameran kuva kynästä ja haalarimerkistä.

Toisen version käyttöliittymän muutokset näkyvät kuvassa 10. Suurin osa informaatiosta piilotettiin Debug-valinnan taakse. Lisäksi mitat näytetään nyt mitattavan esineen vieressä. Mitattavan esineen päälle sovitettavaksi voi myös valita ympyrän suorakulmion sijasta.



Kuva 10. Uusittu päänäkymä ja -valikko.

Erona ensimmäiseen version parametreihin refObjSatMinimum-arvo poistettiin, koska havaittiin, että referenssiesineen kylläisyyden muutos on hyvin pientä olosuhteista riippumatta. Sen sijaan lisättiin MaxBound-arvo, jolla mitattavan esineen kynnystämisen ylärajaa voi säätää.

4. TULOKSET

Sovelluksen testaus suoritettiin kolmella testipatterilla, joissa arvioitiin referenssiesineen tunnistusta, kohde-esineen tunnistusta, sekä mittauksen tarkkuutta. Testien perusteella huomattiin, että mittaus pääsääntöisesti onnistuu, mutta erilaiset valaistusolosuhteet ja kuvioitu tausta aiheuttavat ongelmia. Lisäksi huomattiin, että sovelluksen toinen versio toimii ensimmäistä paremmin.

4.1. Testaussuunnitelma

Sovelluksen toiminnan testaamiseen muodostettiin kompleksisuudeltaan vaihtelevia testejä. Kehityksen alkuvaiheissa tavoiteltiin ainoastaan yksinkertaisten testien läpäisyä. Kaikissa testeissä kamera on suunnattu siten, että sillä kuvataan kohtisuoraan testissä käytettyyn tasoon, paperiarkkiin tai muuhun tasoon nähden vakioetäisyydeltä, ellei testin kuvauksessa ole erikseen muuta sanottu.

Kahden ensimmäinen testipatterin tarkoituksena on karkealla tasolla selvittää sovelluksen toimivuus erilaisissa olosuhteissa testaajan arvioinnin perusteella. Kolmas testipatteri mittaa sovelluksen mittaustarkkuutta numeerisella asteikoilla. Ryhmässä katsottiin, että seuraavaksi määritellyt testit riittivät sovelluksen toiminnan tasoon nähden. Jatkokehitystä ajatellen on todennäköisesti kehitettävä tarkempia testejä. Seuraavaksi kuvataan suunnittelemamme testipatterit.

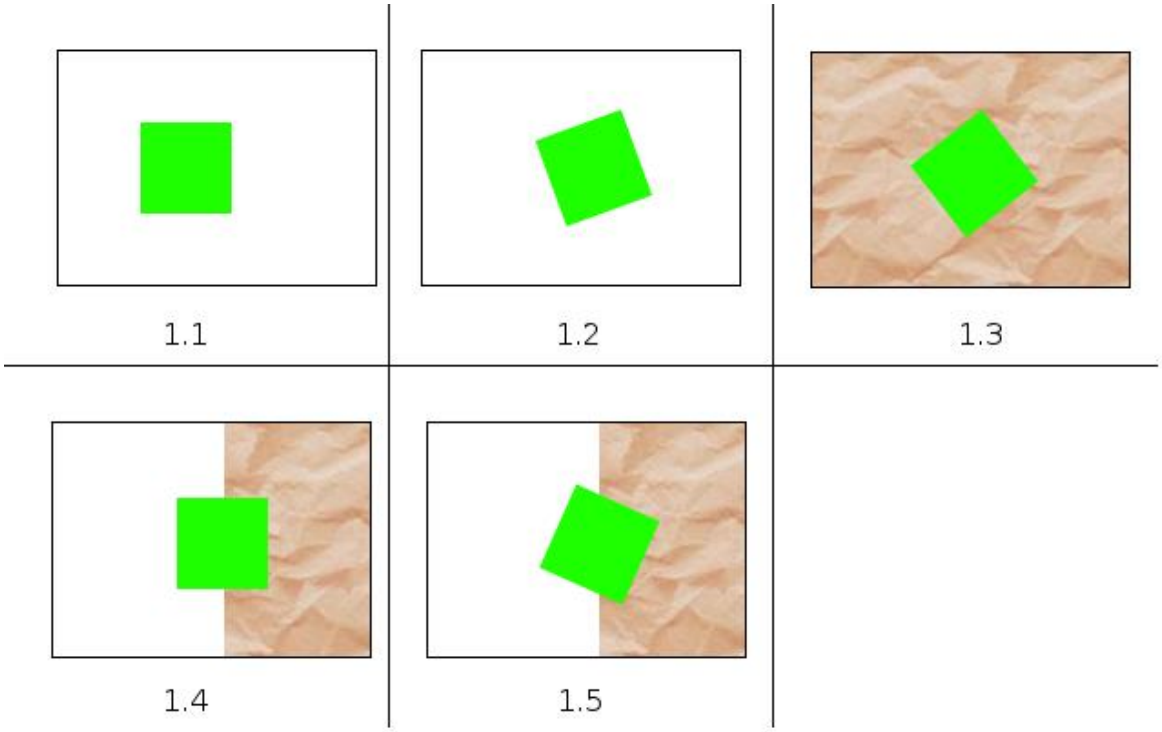
4.1.1. Testipatteri 1

Testipatteri 1:n testeillä testataan referenssiesineen havaitseminen ja sen dimensioiden mittaaminen. Taulukossa 3 on kuvattu testipatteri 1:n testit alkaen yksinkertaisimmasta ja päättyen kompleksisimpaan. Kuvassa 12 on havainnollistettu taulukon 3 testikuvauksia.

Taulukko 3. Referenssiesineeseen tunnistamiseen liittyvät testit.

Nro.	Kuvaus	Läpäisykriteerit
1.1	Referenssiesine on asetettu puhtaan A4-arkin päälle. Esine on 0 asteen kulmassa kameran kuvan reunoihin nähden.	Referenssiesineen havaitseminen
1.2	Referenssiesine on asetettu puhtaan A4-arkin päälle. Esine on noin 70 asteen kulmassa kameran kuvan reunoihin nähden.	Referenssiesineen havaitseminen
1.3	Referenssiesine on asetettu kuvioidun tason päälle (tarkennetaan myöhemmin). Esine on satunnaisessa nollasta poikkeavassa kulmassa kameran kuvan reunoihin nähden.	Referenssiesineen havaitseminen
1.4	Referenssiesine on asetettu puolittain puhtaan A4-arkin päälle ja puolittain kuvioidun tason päälle siten, että esineen reunat ovat samansuuntaisesti paperiarkkiin nähden.	Referenssiesineen havaitseminen

1.5	Referenssiesine on asetettu puolittain puhtaan A4-arkin päälle ja puolittain kuvioidun tason päälle siten, että esineen reunat ovat satunnaisessa nollasta poikkeavassa kulmassa paperiarkkiin nähden.	Referenssiesineen havaitseminen



Kuva 11. Ensimmäisen testipatterin visualisointi.

4.1.2. Testipatteri 2

Testipatteri 2:n testeillä testataan kohde-esineen havaitseminen eri tilanteissa. Taulukossa 4 on kuvattu testipatteri 2:n testit alkaen yksinkertaisimmasta ja päättyen kompleksisimpaan. Kuvassa 13 on havainnollistettu taulukon 4 testikuvauksia.

Taulukko 4. Kohde-esineeseen havaitsemiseen liittyvät testit.

Nro.	Kuvaus	Läpäisykriteerit
2.1	Referenssiesine ja kohde-esine ovat asetettu puhtaan A4-arkin päälle erilleen toisistaan. Esineet ovat mielivaltaisissa kulmissa kameran kuvan reunoihin nähden.	Referenssiesineen havaitseminen ja dimensioiden mittaaminen. Kohde-esineen havaitseminen.
2.2	Referenssiesine ja kohde-esine ovat asetettu puhtaan A4-arkin päälle siten, että niiden reunat ovat kosketuksissa toisiinsa. Esineet ovat	Referenssiesineen havaitseminen ja dimensioiden mittaaminen.

	mielivaltaisessa kulmassa kameran kuvan reunoihin nähden.	Kohde-esineen havaitseminen.
2.3	Referenssiesine ja kohde-esine ovat asetettu kuvioidun tason päälle erilleen toisistaan. Esineet ovat mielivaltaisissa kulmissa kameran kuvan reunoihin nähden.	Referenssiesineen havaitseminen ja dimensioiden mittaaminen. Kohde-esineen havaitseminen.
2.4	Referenssiesine ja kohde-esine ovat asetettu kuvioidun tason päälle siten, että niiden reunat ovat kosketuksissa toisiinsa. Esineet ovat mielivaltaisessa kulmassa kameran kuvan reunoihin nähden.	Referenssiesineen havaitseminen ja dimensioiden mittaaminen. Kohde-esineen havaitseminen.

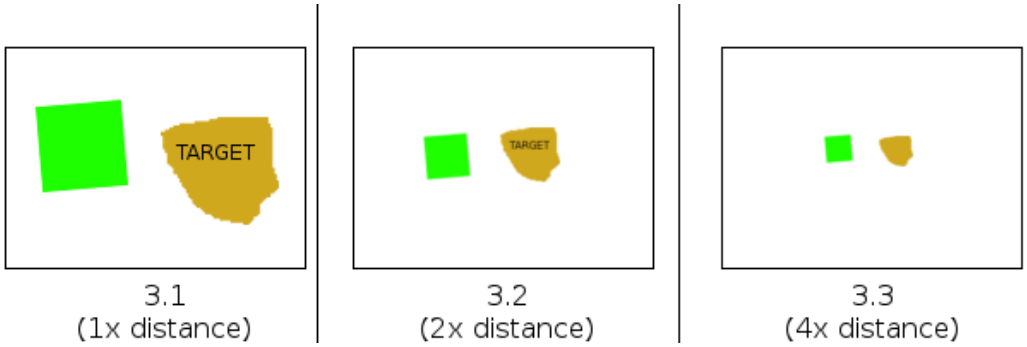
Kuva 12. Toisen testipatterin visualisointi.

4.1.3. Testipatteri 3

Testipatteri 3:n testeillä testataan kohde-esineen dimensioiden mittaaminen eri tilanteissa. Taulukossa 5 on kuvattu testipatteri 3:n testit alkaen yksinkertaisimmasta ja päättyen kompleksisimpaan. Kuvassa 14 on havainnollistettu taulukon 5 testikuvauksia.

Taulukko 5. Kohde-esineen dimensioiden mittaamiseen liittyvät testit.

Nro.	Kuvaus	Läpäisykriteerit
3.1	Referenssiesine ja kohde-esine ovat asetettu puhtaan A4-arkin päälle erilleen toisistaan satunnaisissa kulmissa. Kuvausetäisyys 25cm.	Kohde-esineen dimensioiden mittaaminen oikein jokaisessa kulmassa.
3.2	Sama kuin testi 3.1, sillä erolla, että nyt kamera kuvaa etäisyydeltä 50cm.	Kohde-esineen dimensioiden mittaaminen oikein jokaisessa kulmassa.
3.3	Sama kuin testi 3.1, sillä erolla, että nyt kamera kuvaa etäisyydeltä 100cm.	Kohde-esineen dimensioiden mittaaminen oikein jokaisessa kulmassa.



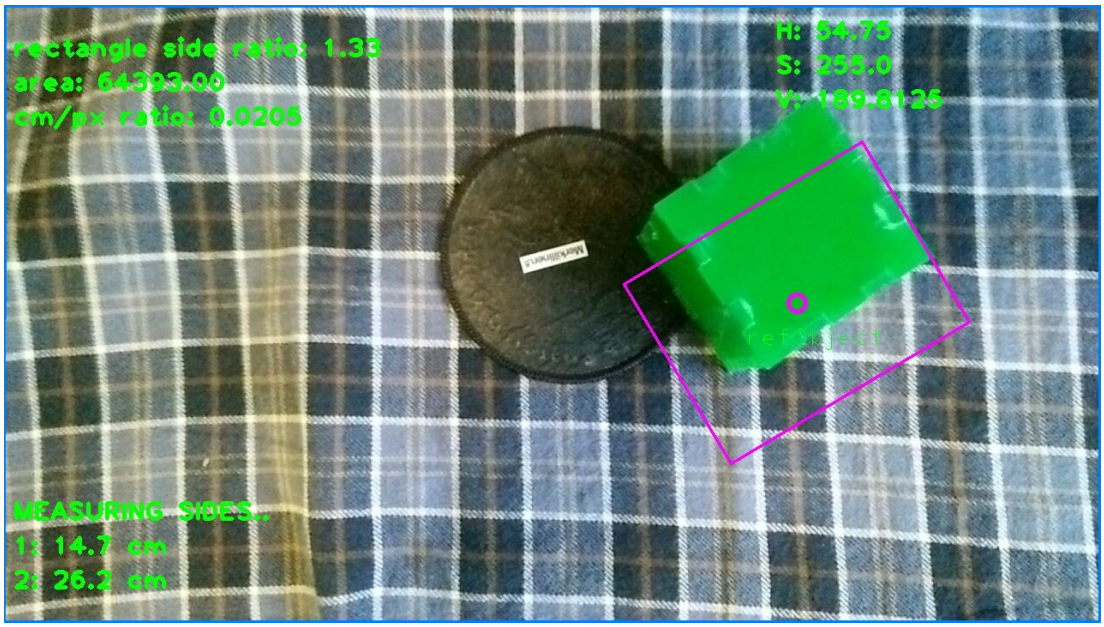
Kuva 13. Kolmannen testipatterin visualisointi.

4.2. Testien tulokset

Toisen ohjelmistoversio vaikuttaa testien perusteella tuovan parannuksia ensimmäiseen versioon nähden esineiden tunnistamisessa. Toisen version mittausvirhe vaihtelee 2,3- ja 18,2-prosentin välillä riippuen mitattavan esineen koosta ja älypuhelimen etäisyydestä mitattavaan kohteeseen. Tarkempien tulosten saamiseksi olisi voitu määritellä useampia testejä, mutta katsottiin, että nykyiset testit riittivät sovelluksen kehitykseen. Jatkokehitys-luvussa on määritelty mahdollisesti toteutettavia testausideoita.

4.2.1. Ensimmäisen version tulokset

Ensimmäinen ja toinen testipatteri suoritettiin kolme kertaa erilaisissa olosuhteissa. Ensimmäisen testipatterin perusteella referenssiesineen tunnistaminen onnistuu. Kuitenkin algoritmin suorittamisesta jokaiselle kameran kuvalle johtuen, tunnistus ei ole täysin jatkuvaa, vaan on mahdollista, että välillä tunnistus ei onnistu. Ensimmäisen version testien tulokset on taulukoitu liitteeseen 1.



Kuva 14. Mittaus ei onnistu. Esineet ovat toistensa päällä.

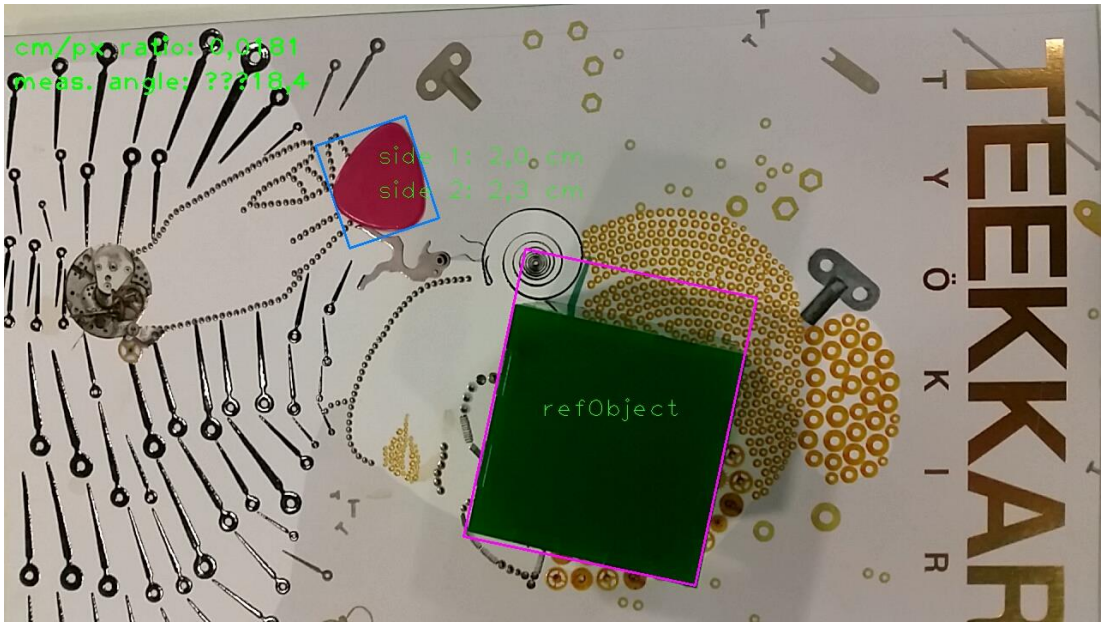
4.2.2. Toisen version tulokset

Toisen version testikierros suoritettiin samassa tilassa yhtä aikaa kolmella eri Android -puhelimella. Referenssiesine tunnistettiin ensimmäisessä testipatterissa vielä paremmin kuin aiemmalla versiolla; varjojen vaikutus saatiin eliminoitua. Toisessa testipatterissa havaittiin hieman edistystä verrattuna ensimmäiseen ohjelmaversioon, mutta osaa testeistä ei vieläkään läpäisty. Kolmannessa testipatterissa onnistuttiin mittaamaan kahden kohde-esineen dimensiot eri etäisyyksiltä.

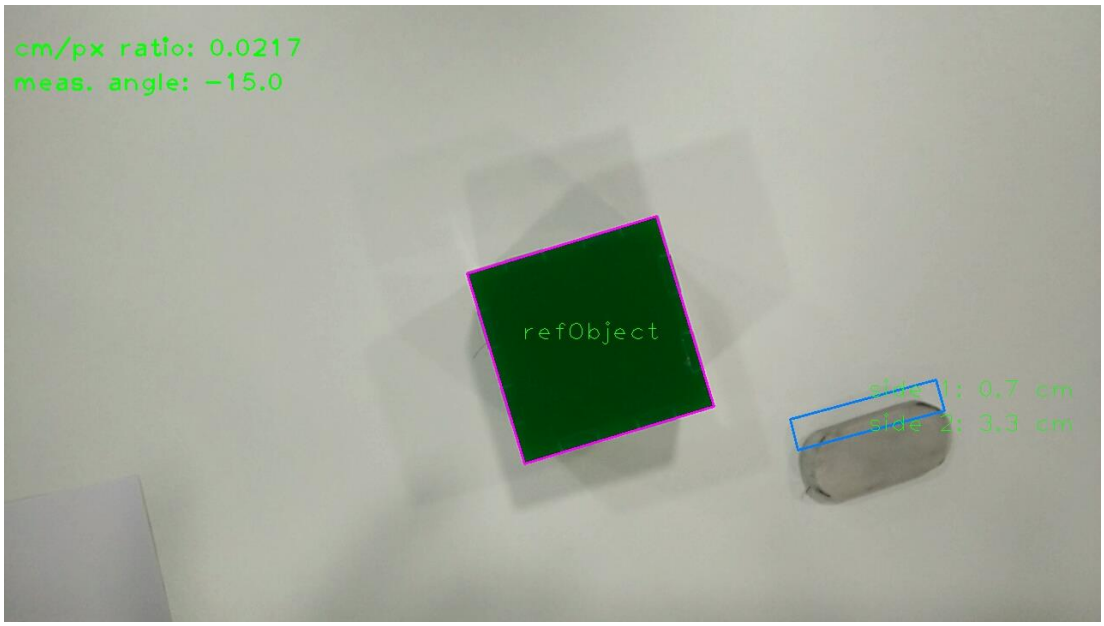
Kohde-esineet tunnistettiin melko hyvin kohdassa 2.1 viimeistään hienosäätämällä asetuksista tunnistettavan esineen kynnysarvoa. Kohdissa 2.2 ja 2.4 tunnistus epäonnistui selkeästi, kun referenssiesineen reunat olivat kosketuksissa kohde-esineen reunoihin. Kohdassa 2.3 testattiin tunnistusta kuvioidun tason päällä vaihtelevin tuloksin. Havaittiin, että tunnistuksen onnistuminen riippuu suuresti kuvioinnin muodoista sekä vähemmässä määrin kohde-esineen koosta.

Testipatterissa 3 käytetiin mitattavina kohde-esineinä tummaa kaukosäädintä ja violettia plektraa. Esineet mitattiin käsivaralta etäisyyksiltä 50 cm ja 100 cm. Kaukosäätimen koon vuoksi vain plektra onnistuttiin mittaamaan myös etäisyydeltä

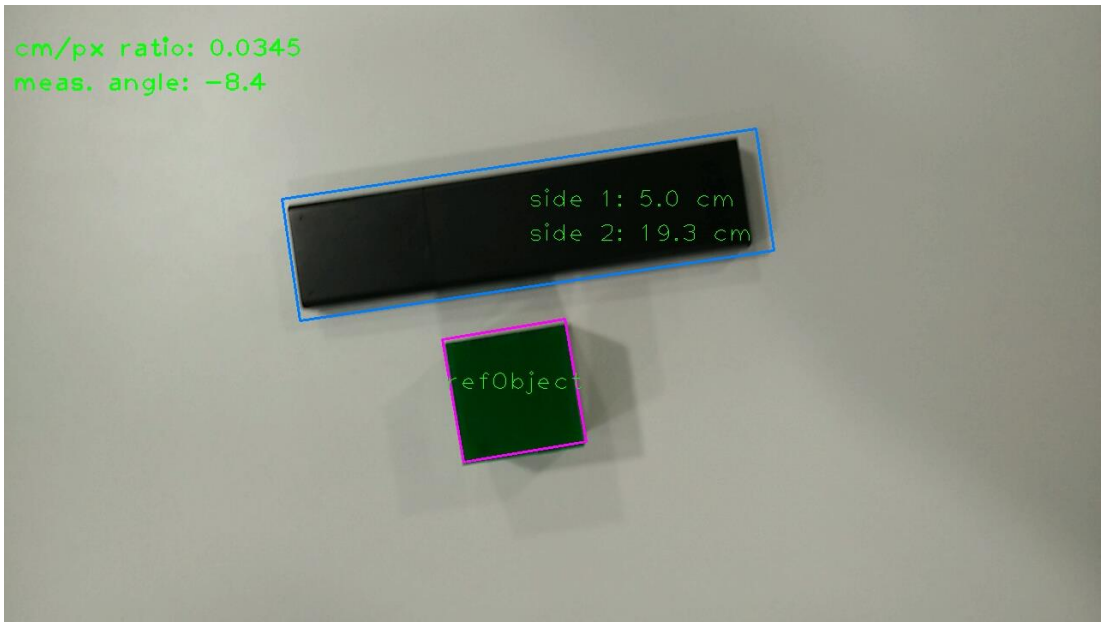
25 cm. Havaittiin mittaustulosten olevan tarkimmillaan noin 50 cm etäisyydellä, jolloin mittausvirhe oli enintään 4,6 %. Metrin etäisyydellä virhe oli välillä 2,6 % ja 18,2 %. Toisen version testien tulokset on taulukoitu liitteeseen 2.



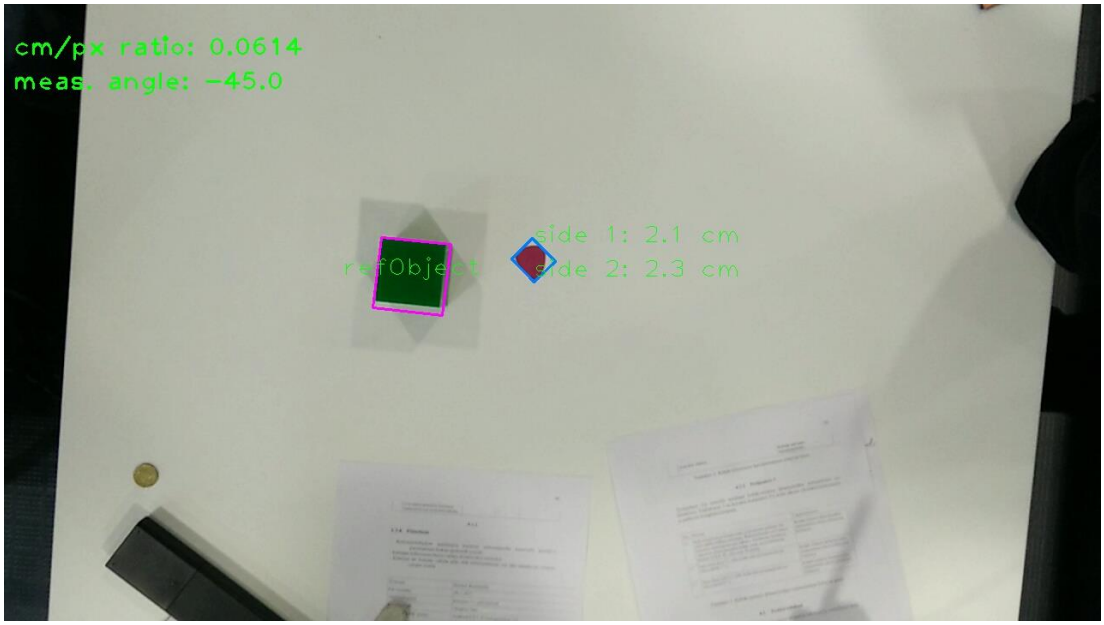
Kuva 15. Referenssiesine ja kohde-esine tunnistetaan kuvioidulla pohjalla hyvin. Koska mittakaavan määrittämiseen käytetään referenssiesineen lyhempää sivua, mittaus onnistuu.



Kuva 16. Mitattavan esineen mittaus ei onnistu, koska kohde-esineen värisävyt ovat liian lähellä taustaväriä.



Kuva 17. Mitattavan esineen varjot heikentävät mittaustarkkuutta.



Kuva 18. Sekä referenssiesine että mitattava esine onnistuttiin mittaamaan metrin etäisyydeltä varjoista huolimatta.

4.3. Pohdinta

Uudempi ohjelmaversio sisältää selviä parannuksia ensimmäiseen verrattuna, mikä myös havaittiin testauksessa. Referenssiesineen tunnistukseen on lisätty ensimmäisten testien jälkeen kaivattu varjojen eliminointi sekä paranneltu kohinanpoisto. Mitattavan esineen tunnistaminen ei ole enää toisessa ohjelmaversiossa yhtä satunnaista, ja tunnistamisen helpottamiseksi eri tilanteissa on lisätty käyttäjäasetuksia ohjelman valikkoon.

5. PROJEKTIN KUVAUS

Tämä työ on tehty osana sulautettujen ohjelmistojen projekti-kurssia, jonka pääaiheena oli augmentoitu todellisuus. Aikaansaamamme sovelluksen AR-osuus on suhteellisen vähäinen mutta sovellus toimii ratkaisuna esineen tunnistus osaongelmaan.

Vaihtoehto toteuttamiselle olisi ollut Raspberry Pi 2 kosketusnäytöllä, kameralla ja wi-fi lisälaitteella. Sovellus päätettiin kuitenkin toteuttaa Android-älypuhelimille. Suurin syy tähän oli sovelluksen jatkokehitys. Jos näyttäisi siltä, että sovellus olisi toimiva ja sille löytyisi selviä käyttötapauksia, sen voisi julkaista Google Play-kaupassa. Esineiden mittaaminen älypuhelimella, joka käyttäjällä on jatkuvasti mukana, on huomattavasti helpompaa kuin kannettavalla tietokoneella. Vaikka Raspberry Pi 2 lisälaitteineen muistuttaa tablettia, tämänkaltaisten laitteiden levinneisyys on suhteellisen pientä verrattuna älypuhelimiin.

Android toteutukseen liittyi kuitenkin haasteita, joita Raspberry Pi:lla ei olisi ollut. Kukaan tiimin jäsenistä ei ollut aikaisemmin toteuttanut Android-sovellusta. Lisäksi OpenCV-kirjaston integroimiseen sovellukseen meni oletettua kauemmin aikaa.

Seuraavaksi kuvataan ohjelmiston kehitysprosessia, työnjakoa tiimin sisällä ja projektin ajankäyttöä.

5.1. Ohjelmiston kehitysprosessi

Kenelläkään ryhmän jäsenistä ei ollut kokemusta Android-kehittämisestä tai konenäön soveltamisesta, joten perinteisen vesiputousmallin [30] seuraaminen ei ollut järkevä vaihtoehto. Tiedettiin siis alun perin, että projektin aikana todennäköisesti kohdataan haasteita, joita ei voi alun perin suunnitteluvaiheessa ennakoida. Tästä syystä päädyttiin eräänlaiseen ketterään kehitysprosessiin [31].

Seuraavaksi toteutettavat ominaisuudet päätettiin ryhmän palavereissa ja niiden toteuttamiselle asetettiin aikarajat. Päälinjat ominaisuuksille suunniteltiin tiimissä ja ominaisuuden toteuttava henkilö vastasi yksityiskohtaisesta suunnittelusta ja testaamisesta. Koodin testaus tarkoitti ryhmässämme lähinnä ryhmän sisäistä koodin arviointia. Suunnittelu, toteuttaminen ja testaaminen suoritettiin suurelta osin samanaikaisesti. Palaverin päätökset kirjattiin myös ylös Slack-viestintäohjelmalla [32], jossa muukin kommunikointi pääasiassa tapahtui. Versionhallintaan ja koodin säilytykseen käytettiin GitHub-palvelua [33].

5.2. Työnjako

Kaikki ryhmän jäsenet osallistuivat työn toteuttamiseen yhtäläisellä panostuksella. Vaikka varsinainen työskentely oli hyvin itsenäistä, merkittävät päätökset tehtiin tiimissä. Jonkinlaisia rooleja kuitenkin muodostui projektin aikana. Työskentelyn kannalta oli tehokkaampaa, että henkilö suoritti samankaltaisia tehtäviä. Janne Mustaniemi vastasi pääosin referenssiesineen tunnistamisesta, Joonas Jyrkkä mitattavan esineen tunnistamisesta ja Eemeli Ristimella käyttöliittymästä parametreille ja kandidaatintyön kirjoittamisesta.

5.3. Ajankäyttö

Projektin ajankäytöstä suurin osa kului sovelluksen kehitykseen ja kandintyön kirjoittamiseen. Tavoitteena oli mahdollisimman hyvä sovellus, joka rajatun aikamäärän puitteissa olisi ryhmän taidoilla mahdollista tehdä. Tämän vuoksi päätettiin minimoida tarvittavan testaamisen määrän, ja keskittyä tarkan mittavan datan hankkimisen sijaan enemmänkin heuristiseen testaamiseen. Päätökseen tarkempien testien poisjättämisestä vaikutti myös kontrolloidun testiympäristön ja -välineiden puute. Esimerkiksi tarkalleen säädeltävällä valaistuksella olisi ollut mahdollista selvittää rajatapaukset, jolloin sovelluksen mittausprosessi ei enää onnistu.

6. TULEVA KEHITYS

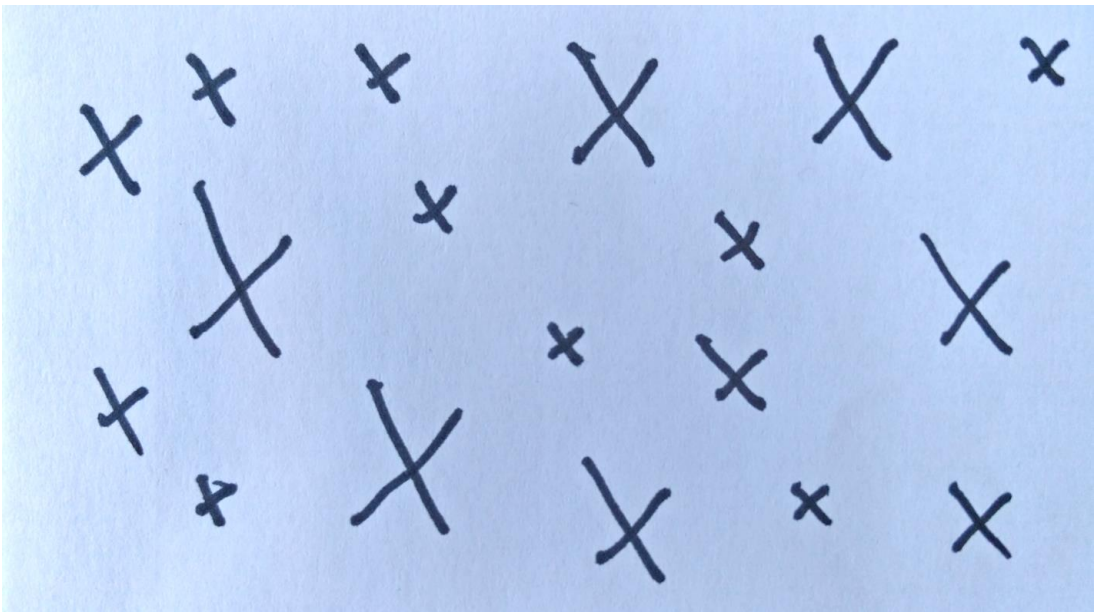
Testeissä selvisi, että referenssiesineen ja mitattavan esineen tunnistamisessa on kehitettävää. Referenssiesineen tunnistuksen voisi toteuttaa myös piirteisiin perustuvalla hahmontunnistuksella. Mitattavan esineen tunnistusta voisi puolestaan kehittää antamalla käyttäjälle mahdollisuus napauttaa näytettävää mitattavaa esinettä.

Määriteltyjen testipattereiden tarkkuuta voisi parantaa lisäämällä toistoja testaukseen ja luomalla säädellympi testausympäristö. Myös käyttöliittymää voisi määritellä käyttäjätestauksen ja muokata sitä saadun palautteen perusteella. Jos suorituskyky sovelluksen jatkokehityksessä muodostuu ongelmaksi, on mahdollista harkita JNI:n käyttämistä.

6.1. Referenssiesineen tunnistamisen jatkokehitys

Alun perin tarkoituksena oli antaa käyttäjän määrittää kustomoitu referenssiesine. Väriin perustuva referenssiesineen tunnistus soveltuu huonosti tai ei ollenkaan käyttäjän itse valitsemille referenssiobjekteille, koska niiden kokoa, muotoa ja väriä ei voi etukäteen tietää. Tämän vuoksi työssä taustoitettiin myös piirteisiin perustuvia hahmontunnistusmenetelmiä, jotka voisivat soveltua kyseiseen ongelmaan.

Käyttäjän mahdollisuus määrittää referenssiesine itse voisi olla hyödyllistä sovelluksen tuotteistamisen näkökulmasta: tällä hetkellä on käytettävä ennalta määritellyn sävyistä referenssiesinettä, jolloin sen hankkiminen voi muodostua ongelmaksi. Piirteiden perusteella esineen tunnistamisen hoitavat algoritmit mahdollistaisivat myös jonkin ennalta määritellyn kuvion käyttämisen referenssiesineenä.



Kuva 19. Mahdollinen referenssiesine ORB-pohjaiselle tunnistusalgoritmille. Paperiarkki jossa mustia ristejä.

Esimerkiksi kuvassa 20 esitelty referenssiesine olisi mahdollista tulostaa, joka helpottaisi sovelluksen jakamista. Pelkän valkoisen paperipalan käyttäminen referenssiesineenä, ja sen tunnistaminen nykyisellä toteutuksella, olisi mahdollista muuttamalla sävyn ja saturaation raja-arvoja. Ongelmaksi kuitenkin muodostuu valkoisten esineiden ja pintojen yleisyys.

Jos ominaisuus käyttäjän itse määrittämästä referenssiesineestä toteutettaisiin, prosessi voisi esimerkiksi alkaa ottamalla kuva halutusta referenssiesineestä, josta käyttäjä rajaisi halutun esineen. Tämän jälkeen referenssiesinettä voitaisiin analysoida: värisävyn ollessa riittävän tasainen voitaisiin käyttää väriin ja muussa tapauksessa piirteisiin perustuvaa hahmontunnistusta.

6.2. Mitattavan esineen tunnistamisen jatkokehitys

Tällä hetkellä mitattavan esineen tunnistaminen vaatii measObjBound- ja measObjMaxBound-parametrien säätämistä eri valaistusolosuhteita varten: tunnistusalgoritmissa on osana kynnystäminen edellä mainituilla parametreilla. Parametrit voitaisiin poistaa tai ainakin pienentää, jos käytettäisiin adaptiivista kynnystämistä.

Tällä hetkellä löydetyistä suljetuista alueista valitaan suurin alue. Tästä seuraa ongelmia tunnistamisessa, jos kuvassa on useita esineitä. Olisi todennäköisesti hyödyllistä, jos käyttäjä voisi näyttöä napauttamalla kertoa, minkä esineen haluaa mitata. Tässä tapauksessa napautuksen koordinaatit otettaisiin ylös ja niiden perusteella laskettaisiin etäisyydet löydettyihin aluesiin(contours) ja näytettäisiin lähimpänä oleva.

6.3. Testien jatkokehitys

Jotta sovelluksen tunnistustarkkuutta voisi parantaa, tarvitaan myös tarkempia testejä, joilla tunnistustarkkuutta mitattaisiin. Tarkkuutta voisi ilmaista esimerkiksi prosentteina: kameran kuvat, jossa tunnistus onnistuu, verrattuna kaikkiin käsiteltäviin kuviin. Tunnistuksen onnistumisen määrittelyn tekisi testaaja silmämääräisesti. Tämänkaltainen testaaminen vaatisi useita toistoja samanlaisessa testiympäristössä erilaisille esineille, jotta tulokset olisivat relevantteja. Samaa menetelmää voisi käyttää myös mitattavan esineen tunnistamiseen ja lisäksi mittaustarkkuuden määrittämiseen.

Sovelluksen monimutkaistuessa olisi suositeltavaa kehittää yksikkötestit, joilla sovellus testattaisiin pienemmissä osissa. Esimerkiksi RefObjDetector- ja MeasObjDetector -luokkien metodeille on mahdollista kehittää testit, jotka tutkisivat muun muassa metodin palauttamaa datatyyppiä ja arvoa metodin eri parametreilla. Lopuksi olisi mahdollista testata pienempien komponenttien yhteensopivuus integrointitestaamisella.

Sovelluksen suorituskykyä voisi mitata tarkemmin laskemalla keskimääräisiä FPS-arvoja sovelluksen suorituksen aikana. FPS-arvot olisi mahdollista ohjata tekstitiedostoon ennalta määritellyltä ajanjaksolta ja laskea niiden perusteella halutut mittarit.

6.4. Käyttöliittymän jatkokehitys

Nykyinen käyttöliittymä koostuu mittausnäkymästä ja tekstikentistä parametrien määrittämiseen. Parametreille onkin mahdollista määrittää sopiva konfiguraatioita esimerkiksi eri valaistusolosuhteita varten. Nämä konfiguraatiot lisättäisiin käyttöliittymään, jolloin parametrien säätäminen olisi nopeampaa yleisimpiin tilanteisiin nähden.

Mitattavan esineen tunnistaminen on mahdollista suorittaa pelkästään prosessoitavalle kuvalle jatkuvan tunnistamisen lisäksi. Tämä tarkoittaisi, että käyttäjän komennosta kuva tallennettaisiin ja näytettäisiin tarvittavilla mitoilla älypuhelimen näytöllä. Paremman lähestymistavan löytämiseksi voidaan kehittää sopiva käyttäjätestausmenetelmä. Tämän lisäksi käyttäjätestauksessa voitaisiin tutkia muita mahdollisia käyttöliittymävaihtoehtoja. Lopullisen käyttöliittymän määrittäminen olisi todennäköisesti iteratiivinen prosessi sisältäen useita suunnittelu-, toteutus- ja testausvaiheita.

6.5. Pääsilmukan toteuttaminen natiivimetodeilla

Osa Android sovelluksesta on mahdollista toteuttaa myös JNI-kutsuja hyödyntäen. Suurin hyöty prosessista saadaan, kun sovelluksen pääsilmukka (referenssiesineentunnistus, mitattavan esineen tunnistus, mittaaminen ja mittojen näyttäminen) toteutettaisiin, esimerkiksi C++ kielellä. Tämä voisi mahdollisesti parantaa sovelluksen suorituskykyä. Kuitenkin algoritmin optimoinnilla todennäköisesti saisi parempia tuloksia aikaan, jos koetaan, että suorituskykyä tulee tarve parantaa. Huomattava kuitenkin on, että jos Java-toteutus koetaan riittävän tehokkaaksi, ei lisäoptimointia kannatta tutkia, sillä se kuluttaa turhaan kehitysaikaa.

7. YHTEENVETO

Tässä työssä toteutettiin esineitä mittaava Android-sovellus, joka perustuu referenssiesineen tunnistamiseen kameran kuvasta. Referenssiesineenä toimii ennalta määritelty 5x5x5 (cm) vihreä kuutio, jonka täytyy olla samassa tasossa mitattavan esineen kanssa.

Testien perusteella näyttää siltä, että sovellus toimii hyvin tasaisissa valaistusolosuhteissa, mutta kuvioitu pinta ja haastava valaistus saattavat hankaloittaa tunnistamista. Ongelmia syntyy myös referenssiesineen ja mitattavan esineen ollessa toistensa päällä ja silloin, kun mitattava esine ei erotu tarpeeksi hyvin taustasta. Tarkkuus vaihtelee 2,3–18,2 prosentin riippuen mitattavan esineen etäisyydestä kameraan ja mitattavan esineen koosta.

Lisäksi työssä kartoitettiin mahdollisia toteutustapoja käyttäjän itse valitsemansa referenssiesineen käyttämiseen. Löydettyjen lähteiden perusteella ORB- ja FAST algoritmit vaikuttavat lupaavimmalta [13, 14]. Lisätietoa aiheesta voi lukea 2.3- ja 6.1-luvuista. On myös selvää, että sovelluksen tämänhetkinen taso ei riitä mahdolliseen kaupallistamiseen, vaan tarvitaan vielä jatkokehitystä erityisesti tunnistustarkkuudessa.

8. LÄHTEET

[1] Gonzalez R & Woods R (2002) Digital Image Processing (2nd Edition).

[2] Yang, M. H., Kriegman, D. J., & Ahuja, N. (2002). Detecting faces in images: A survey. IEEE Transactions on pattern analysis and machine intelligence, 24(1), 34-58.

[3] Canny, J. (1986). A computational approach to edge detection. IEEE Transactions on pattern analysis and machine intelligence, (6), 679-698.

[4] Canny OpenCV tutorial. URL: http://docs.opencv.org/trunk/da/d22/tutorial_py_canny.html. Vierailtu 3.5.2017.

[5] Bradski, G., & Kaehler, A. (2008). Learning OpenCV: Computer vision with the OpenCV library. " O'Reilly Media, Inc.".

[6] Suzuki, S. (1985). Topological structural analysis of digitized binary images by border following. Computer vision, graphics, and image processing, 30(1), 32-46.

[7] Smith, A. R. (1978). Color gamut transform pairs. ACM Siggraph Computer Graphics, 12(3), 12-19.

[8] Szeliski, R. (2010). Computer vision: algorithms and applications. Springer Science & Business Media.

[9] Muja, M., & Lowe, D. G. (2009). Fast approximate nearest neighbors with automatic algorithm configuration. VISAPP (1), 2(331-340), 2.

[10] Harris, C., & Stephens, M. (1988, August). A combined corner and edge detector. In Alvey vision conference (Vol. 15, No. 50, pp. 10-5244).

[11] Shi, J. (1994, June). Good features to track. In Computer Vision and Pattern Recognition, 1994. Proceedings CVPR'94., 1994 IEEE Computer Society Conference on (pp. 593-600). IEEE.

[12] Lowe, D. G. (2004). Distinctive image features from scale-invariant keypoints. International journal of computer vision, 60(2), 91-110.

[13] Bay, H., Tuytelaars, T., & Van Gool, L. (2006). Surf: Speeded up robust features. Computer vision–ECCV 2006, 404-417.

[14] Panchal, P. M., Panchal, S. R., & Shah, S. K. (2013). A comparison of SIFT and SURF. International Journal of Innovative Research in Computer and Communication Engineering, 1(2), 323-327.

[15] Rosten, E., & Drummond, T. (2006). Machine learning for high-speed corner detection. Computer vision–ECCV 2006, 430-443.

[16] Calonder, M., Lepetit, V., Strecha, C., & Fua, P. (2010). Brief: Binary robust independent elementary features. Computer Vision–ECCV 2010, 778-792.

[17] Ke, Y., & Sukthankar, R. (2004, June). PCA-SIFT: A more distinctive representation for local image descriptors. In Computer Vision and Pattern Recognition, 2004. CVPR 2004. Proceedings of the 2004 IEEE Computer Society Conference on (Vol. 2, pp. II-II). IEEE.

[18] Rublee, E., Rabaud, V., Konolige, K., & Bradski, G. (2011, November). ORB: An efficient alternative to SIFT or SURF. In Computer Vision (ICCV), 2011 IEEE International Conference on (pp. 2564-2571). IEEE.

[19] Jeong, K., & Moon, H. (2011, May). Object detection using FAST corner detector based on smartphone platforms. In Computers, Networks, Systems and Industrial Engineering (CNSI), 2011 First ACIS/JNU International Conference on (pp. 111-115). IEEE.

[20] Saipullah, K. M. (2013). Comparison of feature extractors for real-time object detection on android smartphone. Journal of Theoretical and Applied Information Technology, 135-142.

[21] Thakker, S., & Kapadia, H. (2015, September). Image processing on embedded platform Android. In Computer, Communication and Control (IC4), 2015 International Conference on (pp. 1-6). IEEE.

[22] Android plartform architecture. URL: https://developer.android.com/guide/platform/index.html, Vierailtu 3.5.2017.

[23] Android application fundamentals. URL: https://developer.android.com/guide/components/fundamentals.html. Vierailtu 3.5.2017.

[24] Android Native Development Kit, https://developer.android.com/ndk/guides/index.html. 3.5.2016

[25] JNI. URL: http://docs.oracle.com/javase/7/docs/technotes/guides/jni/spec/intro.html#wp9502. Vierailtu 3.5.2017

[26] React Native. URL: https://facebook.github.io/react-native/. Vierailtu 3.5.2017.

[27] Phone gap. URL: http://phonegap.com/. Vierailtu 3.5.2017.

[28] Open CV yleistä. URL: http://opencv.org/about.html. Vierailtu 3.5.2017.

[29] Java docs. URL: http://docs.opencv.org/java/2.4.9/. Vierailtu 3.5.2017.

[30] Vesiputousmalli. URL: https://en.wikipedia.org/wiki/Waterfall_model. Vierailtu 28.2.2017.

[31] Ketterä kehitys. URL:https://en.wikipedia.org/wiki/Agile_software_development. Vierailtu 3.5.2017.

[32] Slack. URL: https://slack.com/. Vierailtu 3.5.2017.

[33] Github. URL: https://github.com/. Vierailtu 3.5.2017.

9. LIITTEET

Liite 1. Ensimmäisen ohjelmaversion testitulokset

Testaaja	Janne Mustaniemi
Päivämäärä	25.1.2017
Ohjelmaversio	Ohjelmaversio 1 + päivitykset
Puhelinmalli	Samsung Galaxy S3
Puhelimen OS versio	Android 6.0.1 (CyanogenMod 13)
Valaistuksen kuvaus	Normaali huonevalaistus sekä ikkunasta tuleva luonnonvalo

Testipatteri 1		
Nro.	**Tulos**	**Huomiot**
1.1	PASS	
1.2	PASS	
1.3	PASS	Kuvioidun tason kuviointi koostui neliöruudukosta. Tunnistus heikkeni, kun esineen asetti siten että sen reunat olivat linjassa kuvion neliöiden reunojen kanssa.
1.4	PASS	
1.5	PASS	
Päätelmät		- Testipatteri 1:n osalta ei ole tarvetta parannuksiin.

Testipatteri 2, kohde-esineitä: kahvipaketti, tumma käyntikortti, pyöreä valkoinen purkki		
Nro.	**Tulos**	**Huomiot**
2.1	PASS	Varjot heikentävät tunnistusta valkoisella tasolla
2.2	FAIL	Referenssiesine joutuu mukaan mitattavan esineen pinta-alaan
2.3	PASS	Tunnistus heikompi silloin, kun mitattavan esineen väri on lähellä kuvioidun tason väriä
2.4	FAIL	Referenssiesine joutuu mukaan mitattavan esineen pinta-alaan
Päätelmät		- Referenssiesineen tunnistusta huonosti tarkentuneella kameralla pyrittävä parantamaan hiukan (pehmeät reunat). - Estetään referenssiesineen valinta mitattavaksi esineeksi.

	- Estetään tai lisätään valinta sille, että referenssiesine voi olla mitattavan esineen rajojen sisällä.

Testaaja	Eemeli Ristimella
Päivämäärä	29.1.2017
Ohjelmaversio	Ohjelmaversio 1 + päivitykset
Puhelinmalli	Oneplus One
Puhelimen OS versio	Android 6.0.1 (CyanogenMod 13)
Valaistuksen kuvaus	Heikko huonevalaistus yllä olevasta kattolampusta sekä ikkunasta tuleva heikko luonnonvalo

Testipatteri 1		
Nro.	**Tulos**	**Huomiot**
1.1	PASS	
1.2	PASS	
1.3	PASS	Tason kuviointi koostui tumman flanellipaidan ruutukuviosta.
1.4	PASS	Tunnistuksen tarkkuus heikkeni verrattuna pelkkään ruutukuvioon tai valkoiseen A4-arkkiin. Varjot aiheuttivat ongelmia.
1.5	PASS	Tunnistus tarkkuus heikkeni verrattuna pelkkään ruutukuvioon tai valkoiseen A4-arkkiin. Varjot aiheuttivat ongelmia.
Päätelmät		- Varjot tuottavat hankaluuksia tunnistamiseen. - Haastavasta valaistuksesta huolimatta referenssiesine pystytään tunnistamaan, mutta ei yhtäjaksoisesti.

Testipatteri 2, kohde-esine: vihreä teepussi		
Nro.	**Tulos**	**Huomiot**
2.1	FAIL	Mitattavaa esinettä ei tunnistettu.
2.2	FAIL	Mitattavaa esinettä ei tunnistettu.
2.3	FAIL	Mitattavaa esinettä ei tunnistettu.

2.4	FAIL	Mitattavaa esinettä ei tunnistettu.
Testipatteri 2, kohde-esine: haalarimerkin takaosa		
Nro.	**Tulos**	**Huomiot**
2.1	PASS	Varjot heikentävät tunnistusta valkoisella tasolla
2.2	FAIL	Mitattavaa esinettä ei tunnistettu.
2.3	PASS	
2.4	FAIL	Mitattavaa esinettä ei tunnistettu.
Testipatteri 2, kohde-esine: banaani		
Nro.	**Tulos**	**Huomiot**
2.1	FAIL	Mitattavaa esinettä ei tunnistettu
2.2	FAIL	Mitattavaa esinettä ei tunnistettu
2.3	FAIL	Mitattavaa esinettä ei tunnistettu
2.4	FAIL	Mitattavaa esinettä ei tunnistettu
Päätelmät		- Mitattavan esineen tunnistus toimii pääasiassa ainoastaan tasasävyisille objekteille. - Mitattavan esineen ja referenssiesineen reunat eivät saa olla toistensa päällä, jotta mittaus onnistuisi. - Sovelluksen käyttäminen valaistukseltaan haastavassa ympäristössä on epävarmaa. - Referenssiesineen ja mitattavan esineen samankaltainen värisävy lisää epävarmuutta.

Testaaja	Joonas Jyrkkä
Päivämäärä	1.2.2017
Ohjelmaversio	Ohjelmaversio 1 + päivitykset
Puhelinmalli	Samsung S5
Puhelimen OS versio	Android 6.0.1
Valaistuksen kuvaus	Kirkas LED valaisin osoittamassa sivulle, huoneen perusvalaistus katosta heikko/himmeähkö

Testipatteri 1		
Nro.	**Tulos**	**Huomiot**

1.1	PASS	
1.2	PASS	
1.3	PASS	Tason kuviointi koostui kuvioidusta matosta.
1.4	PASS	Tunnistettava referenssiesine katosi välillä, mutta löytyi hetken kuluttua uudelleen
1.5	PASS	Tunnistettava referenssiesine katosi välillä, mutta löytyi hetken kuluttua uudelleen
Päätelmät		- Referenssiesine pystytään tunnistamaan mutta ei yhtäjaksoisesti.

Testipatteri 2, kohde-esine: huulirasva		
Nro.	**Tulos**	**Huomiot**
2.1	PASS	
2.2	FAIL	Mitattava esine fuusioitui osaksi referenssiesinettä.
2.3	PASS	
2.4	FAIL	Mitattavaa esinettä ei tunnistettu.
Testipatteri 2, kohde-esine: rannekello		
Nro.	**Tulos**	**Huomiot**
2.1	PASS	
2.2	FAIL	Mitattava esine fuusioitui osaksi referenssiesinettä.
2.3	PASS	
2.4	FAIL	Mitattavaa esinettä ei tunnistettu.
Testipatteri 2, kohde-esine: tussi		
Nro.	**Tulos**	**Huomiot**
2.1	PASS	
2.2	FAIL	Mitattava esine fuusioitui osaksi referenssiesinettä.
2.3	FAIL	Tekstit häiritsivät mittausta (tarkoituksella valittu vaikea esine)
2.4	FAIL	Mitattava esine fuusioitui osaksi referenssiesinettä.
Päätelmät		- Mitattavan esineen ja referenssiesineen reunat eivät saa olla päällekkäin.

	- Varjot aiheuttavat mitattavan esineen ja referenssiesineen fuusioitumista yhdeksi esineeksi.

Liite 2. Version 2 testaus

Testaaja	Eemeli Ristimella Janne Mustaniemi Joonas Jyrkkä
Päivämäärä	16.2.2017
Ohjelmaversio	Ohjelmaversio 2 + päivitykset
Puhelinmalli	Oneplus One Samsung Galaxy S3 Samsung S5
Puhelimen OS versio	Android 6.0.1 (CyanogenMod 13) Android 6.0.1 (CyanogenMod 13) Android 6.0.1
Valaistuksen kuvaus	Normaali huonevalaistus.

Testipatteri 1		
Nro.	**Tulos**	**Huomiot**
1.1	PASS	
1.2	PASS	
1.3	PASS	
1.4	PASS	
1.5	PASS	
Päätelmät		- Referenssiesine pystytään tunnistamaan yhtäjaksoisesti - Varjojen vaikutus on saatu eliminoitua

Testipatteri 2, kohde-esineitä: plektra, kaukosäädin, kolikko, pyyhekumi		
Nro.	**Tulos**	**Huomiot**
2.1	PASS	Kohde-esineen tunnistus heikompi, jos sen värisävyt ovat lähellä taustaväriä.
2.2	FAIL	Esineitä ei joko tunnistettu lainkaan, tai sitten molempien yhteinen pinta-ala tunnistui referenssiesineenä.
2.3	DEPENDS	Onnistuminen riippui suuresti taustan kuviotyypistä. Kohde-

		esineeksi saattoi tunnistua alueita taustakuvioinnista, etenkin jos haluttu kohde-esine oli pieni.
2.4	FAIL	Esineitä ei joko tunnistettu lainkaan, tai sitten molempien yhteinen pinta-ala tunnistui referenssiesineenä.
Päätelmät		- Mitattavan esineen ja referenssiesineen reunat eivät edelleenkään saa olla päällekkäin. - Jos tausta on kuvioitu, on kuvioinnin tyypillä suuri merkitys mitattavan esineen tunnistuksessa.

Testipatteri 3, kohde-esine: plektra (2,2 x 2,5 cm)		
Nro.	**Tulos**	**Huomiot**
3.1	2,0 x 2,1 cm	Virheet 2mm ja 4mm (9,1% ja 16.0%)
3.2	2,3 x 2,5 cm	Virheet 1mm ja 0mm (4.6% ja -)
3.3	2,6 x 2,7 cm	Virheet 4mm ja 2mm (18,2% ja 7,4%)
Testipatteri 3, kohde-esine: kaukosäädin (4,5 x 19,7 cm)		
Nro.	**Tulos**	**Huomiot**
3.1	-	Kohde-esine ei mahtunut kuvaan tältä etäisyydeltä.
3.2	4,6 x 19,2 cm	Virheet 1mm ja 5mm (2,3% ja 2.6%)
3.3	5,1 x 20,2 cm	Virheet 6mm ja 5mm (13.4% ja 2,6%)
Päätelmät		- Referenssiesineeseen nähden suuren kappaleen (kaukosäädin) mittaaminen onnistui 6mm mittausvirheellä. - Pienen ja epäsäännöllisemmän muotoisen kappaleen (plektra) mittaaminen onnistui 4mm mittausvirheellä - Kohde-esineen varjot voivat lisätä mittausvirhettä (kts. Kuva kohdassa ”Toisen version tulokset”.