Tampereen yliopisto

Antti Rahtu

MOBIILILAITEEN TESTAUS FYYSISELLÄ ROBOTILLA

Teknisten tieteiden tiedekunta
Diplomityö
Maaliskuu 2019

TIIVISTELMÄ

Antti Rahtu: Mobiililaiteen testaus fyysisellä robotilla
Diplomityö
Tampereen yliopisto
Automaatiotekniikka
Maaliskuu 2019

Tässä työssä tutkittiin mobiililaitteiden fyysiseen manipulaatioon perustuvan testauksen kustannustehokkaita toteutusmahdollisuuksia. Tutkismusprosessi toteutettiin konstruktiivisella menetelmällä. Prosessissa tutkittavaa kysymystä lähestyttiin rakentamalla testirobotin prototyyppi. Laitteisto toteutettiin avoimen lähdekoodin ohjelmilla sekä halvoilla fyysisillä komponenteilla. Laite rakennettiin ryhmätyönä Wapice Oy:n osaamisenkehittämisprojektissa.

Prototyypissä käytettiin runkona 3D-tulostinta, johon kiinnitettiin metallitappi kosketustyökaluksi ja kamera testattavan laitteen tilan havainnointiin. Järjestelmän ohjelmisto rakennettiin käyttäen OpenCV:n template matching ja feature matching -toiminnallisuuksia kuvantunnistukseen, Pythonin yksikkötestauskirjastoa testauslogiikan pohjana ja PyQt:ta käyttöliittymän rakentamiseen.

Prototyyppi täyttää sille asetetun ydinvaatimuksen: Se kykenee havainnoimaan testattavan laitteen tilaa ja manipuloimaan sen näyttöä. Järjestelmä pystyy siis hakemaan testattavan laitteen näytöltä kuvaa ja pystyy myös fyysisesti koskemaan siihen ja siten manipuloimaan näyttöä.

Prototyypin rakentamisen kustannustehokkuus oli kuitenkin vaikea arvioida: Laitteiston rakentamiskustannukset olivat pienet - vain muutama sata euroa - , mutta kehityksen vaatima työpanos suhteellisen suuri. Prosessi toteutettiin osaamisenkehittämisprojektina, joten työskentely oli tehotonta taukojen ja niukkojen resurssien takia.

Prototyypin suurin puute oli testausprosessin epäluotettavuus: Varsinaista luotettavuustestiä ei edes tehty, koska jo alustavassa testauksessa järjestelmän luotettavuus todettiin tarkoitukseen riittämättömäksi. Parempaan luotettavuuteen olisi päästy vaihtamalla laadukkaampaan kameraan ja erityisesti parantamalla kameran ottaman kuvan esikäsittelyn laatua.

Avainsanat: ohjelmistotestaus, mobiililaite, kuvaan perustuva testaus, fyysinen testaus

ABSTRACT

Antti Rahtu: Mobile device testing using a physical robot
Master of Science Thesis
Tampere University
Automation Engineering
March 2019

This thesis investigates the viability and cost-effectiveness of mobile device testing without a software connection to the device being tested. In practice this means using a physical robot to manipulate the device and using a camera to detect its state. To investigate the issue, a prototype of a mobile device testing system was built and evaluated for its implementation and performance.

The prototype was implemented using a modified 3D printer as a robot used for manipulation, using a metal rod as the tool. The software was built using OpenCV template matching and feature matching functionality for image recognition, the Python unit testing library as the basis for testing logic and PyQt for building the interface.

The prototype fulfills its core requirements: It is capable of analyzing the status of the device being tested by retrieving an image from the display of the device being tested. It is also able to physically manipulate the display by touching a specified image found on screen.

The prototype can however not be considered to show the viability of this testing approach due to a number of issues. Two of the issues can be considered essential: The most important problem is that the prototype is too unreliable for testing. This is probably due largely to the fact that work on the project was stopped before proper image pre-processing was implemented, but it is impossible to say for sure how reliable the system would be properly implemented. The second most important problem is that even though the hardware was inexpensive, the project took a lot of development time.

Keywords: program testing, mobile device, image based testing, physical testing

ALKUSANAT

Tämän työn pohjana käytetty prototyyppi tehtiin osana työnantajani Wapice Oy:n osaamisenkehitysprojektia. Haluan kiittää työnantajaani mahdollisuudesta osallistua projektiin ja kaikkia projektiin osallistuneita, erityisesti projektityötiimin johtajaa sekä tämän työn ohjaajaa Timo Aurasta. Haluan myös kiittää perhettäni ja ystäviäni kannustuksesta sekä ohjaajiani Matti Vilkkoa sekä David Hästbackaa rakentavasta palautteesta.

Tampereella, 24. maaliskuuta 2019

Antti Rahtu

SISÄLLYSLUETTELO

1 JOHDANTO

Mobiililaitteiden ohjelmistomarkkinat ovat voimakkaasti kasvava ala. Erityisenä haasteena mobiililaitteiden ohjelmistokehityksessä on laitteiden suuri fyysinen sekä ohjelmistollinen kirjo. Laitteita on eritehoisia ja erikokoisia eri käyttöjärjestelmillä ja useilta valmistajilta. Saman ohjelman olisi kuitenkin yleensä toimittava kaikilla laitteilla, ja käytettäessä alustariippumattomia työkaluja, jopa useilla käyttöjärjestelmillä. Tässä ympäristössä testauksen suunnittelu onkin hyvin tärkeää, koska ohjelmistoa pitäisi testata aina kaikilla alustoilla, joilla sen käyttöä halutaan tukea.

Manuaalinen testaus on aikaa vievää ja virhealtista, joten testauksen automatisointiin on tarjolla lukuisia työkaluja. Nämä työkalut lähestyvät testausta eri tavoilla, mutta yhteinen tekijä lähes kaikille on se, että ne toimivat käyttäen ohjelmallista yhteyttä testattavaan laitteeseen. Ne eivät siis käytä laitetta kuten ihminen vaan lukevat ohjelmallisesti joko fyysiseltä tai simuloidulta mobiililaitteelta sen tilan ja sitten ohjelmallisesti vaikuttavat siihen.

Ohjelmallista yhteyttä hyödyntävästi toteutetussa testausautomaatiossa on paljon hyötyjä: Testien suoritusnopeus riippuu vain kohdelaitteen nopeudesta, laitteen tilan lukemisessa tai tilaan vaikuttamisessa ei voi tulla virheitä ja laitteen tilaan voidaan vaikuttaa täysin mielivaltaisesti. Lisäksi erityisesti simuloidun laitteen tapauksessa testi on helppo suorittaa usealle laitteelle yhtäaikaisesti.

Lähestymistavassa on kuitenkin joitain haittoja: Osa työkaluista on alustariippuvaisia, joka tarkoittaa sitä että Android-laitteelle luodun testin suorittaminen esimerkiksi iOS-laitteella ei onnistu. Testaustyökalu saattaa vaikuttaa laitteen toimintaan testaustilanteessa ja siten esimerkiksi vasteaikojen ja muiden tehokkuusarvojen seuranta on vähemmän luotettavaa. Ehkä tärkein ongelma on kuitenkin se, että testaustyökalu ei käytä laitetta kuten ihmiskäyttäjä. Siten jotkin testattavan sovelluksen virheet jäävät huomaamatta ja kaikkien varsinaiseen laitteeseen liittyvien toimintojen testaus on mahdotonta.

Nämä haittapuolet eivät yleensä ole olennaisia verrattuna menetelmän hyötyihin. Markkinoilla on myös ratkaisuja, jotka vähentävät lähestymistavan ongelmia esimerkiksi olemalla alustariippumattomia tai lukemalla laitteen tilan tulkitsemalla sen näytöltä otettua kuvankaappausta eikä lukemalla sitä ohjelmallisesti.

Ongelmia ei kuitenkaan pysty poistamaan kokonaan ilman laitetta fyysisesti manipuloivaa ratkaisua. Tätä varten tarjolla on myös täysin laitetta fyysisesti manipuloivia ratkaisuja, mutta vapaasti saatavilla olevat ratkaisut eivät ole yhtä hiottuja kuin ohjelmalliset versiot

ja kaupallisesti tarjolla on pikemminkin palveluita kuin valmiita paketteja. Ihmiskäyttäjän kaltainen mobiitestaus on siis kirjallisuuskatsauksen perusteella vähemmän käytetty ja -tutkittu testausmenetelmä.

Tämän työn tavoite on tutkia mobiililaitteiden fyysiseen manipulointiin perustuvia testauksen toteutustapoja. Asiaa tutkitaan rakentamalla fyysiseen testaukseen perustuva testausjärjestelmä. Järjestelmä toteutetaan avoimen lähdekoodin ohjelmistokomponenteilla sekä halvoilla fyysisillä laitteilla. Tutkimus toteutetaan konstruktiivisella menetelmällä.

Työssä edetään käyttäen konstruktiivista metodia tähän käyttötapaukseen soveltuvin osin. Sen prosessia seuraten ensin valitaan tutkimuskysymykseksi käytännön ongelma, jonka jälkeen tutustutaan alaan. Alan taustaa esitellään luvussa kaksi ja tarkennetaan tutkimuskysymystä ja prosessia. Alan esittelyä jatketaan luvussa kolme keskittyen saatavilla oleviin työkaluihin. Luvussa neljä esitellään metodin ydin eli luotava konstruktio, joka tässä tapauksessa on testaukseen kykenevä prototyyppi. Luvussa viisi esitellään konstruktion testaus sekä esitetään prosessin pohjalta saadut tulokset.

2 TAUSTA JA TAVOITTEET

Ohjelmointivirheen hinta on sitä suurempi mitä myöhäisemmässä vaiheessa se havaitaan. Virhe voi olla vaikeasti havaittavissa normaaliin ohjelmistokehityksen työtapaan kuuluvassa vapaamuotoisessa testauksessa tai näkyä jossain muualla kuin siellä mihin kehittäjä tekee muutoksia. Näistä syistä kaikkien muutosten jälkeen olisi periaatteessa testattava koko järjestelmä.

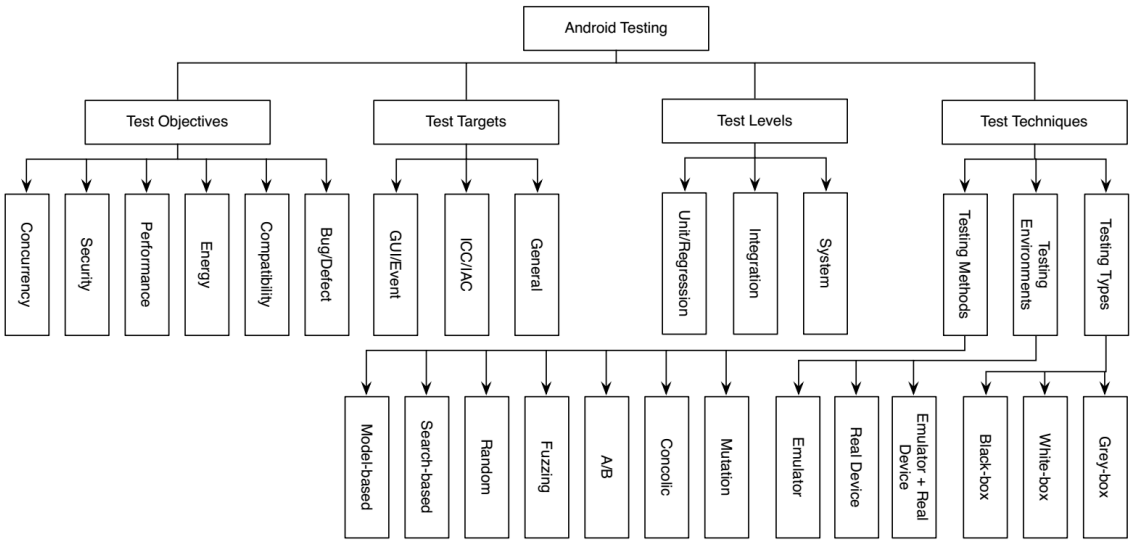
Koko järjestelmän testaus käsin olisi kuitenkin useimmissa ohjelmissa erittäin työlästä. Käsin testaaminen on myös hidasta ja epätarkkaa, joka tarkoittaa sitä että vaikka ohjelmaa testattaisiin laajasti jokaisen muutoksen jälkeen, kehittäjä joutuu joko odottamaan testauksen valmistumista tai tekemään muuta välissä. Tämä viive taas tarkoittaa sitä, että virheen löytyessä kehittäjän on palautettava mieleen tehdyt muutokset ennen kuin hän voi korjata virheen. Näistä syistä laaja manuaalinen testaus jokaisen muutoksen jälkeen ei ole realistista. Käytännössä käsin testauksen lisäongelmana on siten se, että virheen löytyessä on tehty useita muutoksia ja virhe saattaa johtua useasta niistä, tai jopa olla kahden tai useamman muutoksen yhteisvaikutusta.

Näiden ongelmien takia on kehitetty työkaluja testauksen automatisointiin. Ideaalitapauksessa automaattinen testaus korjaa kaikki mainitut ongelmat: Jokaiselle ohjelman osalle luotu testausautomaatio voidaan suorittaa koko ohjelmalle jokaisen muutoksen jälkeen ennen kuin se siirretään kehitystiimin yhteiseen työversioon.

Ohjelmistotestauksen automaation tutkimus painottuu nykyisin erilaisiin tapoihin automatisoida testien luontia niin yleisemmin [4] kuin erityisesti mobiililaitteita [10] käsitellessä. Näissä menetelmissä testaustyökalun käyttäjä ei kirjoita jokaista testiaskelta itse, vaan testaukseen käytettävä ohjelmisto tekee ainakin osan testiaskeleista itsenäisesti. Näin testien luontia saadaan nopeutettua ja testikattavuutta parennettua.

Automaattisen testauksen käytäntöjä ja työkaluja on lukuisia testaustarkoituksen, käyttäjäkunnan, maksullisuuden, kohdejärjestelmän ja tekijän mukaan eri toimintaperiaatteella, eduilla ja ongelmilla. Näiden perusteellinen läpikäynti ei ole tarkoituksenmukaista tässä yhteydessä, mutta esimerkin vuoksi Kong et al. [10] jakavat tutkimuksessaan alan tutkimuksia kuvassa 2.1 olevan taksonomian mukaan.

Tässä työssä Kong et al. käyttämä taksonomia on tarpeettoman tarkka, mutta on hyödyllistä esitellä siitä kaksi käsitettä eli kuvan testaustaso (Test level) ja testaustyyppi (Testing type). Testaustaso viittaa siihen, kuinka suurta osaa testattavasta ohjelmasta testataan kerralla. Testaustyyppi puolestaan rajaa, kuinka paljon tietoa testaava järjestelmä käyttää



Kuva 2.1. *Testauksen taksonomiaa[10]*

testattavan ohjelman sisäisestä rakenteesta.

Testaustaso on skaala yksikkötestauksesta järjestelmätestaukseen. Yksikkötestauksessa testataan mahdollisimman pientä osaa ohjelmasta tavoitteena testata kaikki koodirivit erikseen. Järjestelmätestauksessa testataan koko järjestelmää.

Testaustyyppi on skaala mustalaatikkotestauksesta valkolaatikkotestaukseen. Mustalaatikkotestauksessa testattava järjestelmän ajatellaan olevan musta laatikko, jonka sisään ei nähdä ja siten voidaan testata vain sen vasteita eri syötteisiin. Manuaalinen testaus, jossa ihminen testaa valmista järjestelmää käsin on esimerkki mustalaatikkotestauksesta.

Valkolaatikkotestauksessa hyödynnetään tietoa järjestelmän sisäisestä rakenteesta ja muokataan ja tutkitaan sen tilaa suoraan, mahdollisesti jopa täysin eri tavalla kuin se toimii normaalissa käytössä. Yksikkötestaus, jossa yksittäisiä koodin osia testataan eri syötteillä erillään muusta ohjelmasta, on esimerkki valkolaatikkotestauksesta.

Käsitteet eivät vastaa suoraan toisiaan, mutta niillä on vahva yhteys toisiinsa siten, että yleistäen mitä pienempiä osia testataan, sitä valkolaatikkomaisempaa testaus on ja mitä suurempia osia testataan niin sitä mustalaatikkotestausmaisempaa testaus on. Tämän työn aiheena oleva mobiililaitteen testaus fyysisellä robotilla voidaan sanoa olevan täysin mustalaatikko- ja järjestelmätestausta.

2.1 Mobiililaitteiden testaus

Mobiiliohjelmistojen testauksessa koodin testaus eli edellisessä luvussa mainittu yksikkötestaus, voidaan yleensä tehdä kuten työpöytäsovelluksissakin käyttäen esimerkiksi useista kehitysympäristöistä valmiina löytyviä yksikkötestaustyökaluja. Ohjelman testaus eli edellisessä luvussa mainittu järjestelmätestaus on kuitenkin haasteellista, koska toisin

kuin työpöytäsovelluksilla, ohjelmaa ei ole käytännöllistä kehittää sillä alustalla, jolle se on suunniteltu käytettäväksi. Varsinainen ohjelman testaus vaatii siis ohjelman ajamista mobiililaitteella tai ainakin varsinaisen laitteen emulointia ohjelmallisesti.

Mobiiliohjelmistot ovat vahvasti automaattitestauksesta hyötyvä kohde, koska kehitettävän ohjelman on toimittava laajalla skaalalla eri alustoja. Pelkkiä Android-laitteita oli jo vuonna 2015 yli 24000 tyyppiä [21]. Ohjelmistoversioiden ja alustan fyysisten ominaisuuksien lisäksi ohjelman saattaa olla tarpeen toimia jopa eri käyttöjärjestelmille käytettäessä työkaluja, jotka sallivat saman ohjelman kääntämisen eri alustoille.

Tämän työn aiheena oleva mobiililaitteiden testaus fyysisellä robotilla voidaan nyt ajatella olevan täysin mustalaatikkopohjainen lähestymistapa mobiililaitteiden käyttöliittymätestaukseen. Vastaavien tutkimusten löytäminen on haastavaa, koska vaikka alan tutkimukset usein käyttävät jakoa musta- ja valkolaatikkotestaukseen, käytännössä tällä ei tarkoiteta tässä yhteydessä hyödyllisiä menetelmiä suoran ohjelmallisen yhteyden välttämiseen, vaan ero tehdään testattavan ohjelman näkökulmasta, yleensä sen pohjalta käytetäänkö testattavan ohjelman koodia suoraan. Tällä tarkoitetaan sitä, että testattavaa ohjelmaa suorittavan laitteen käyttölittymä käskytetään ohjelmallisesti käyttäytymään kuin se olisi havainnut kosketuksen ja välittämään ohjelman käyttöjärjestelmälle välittämä informaatio, kuten virheviestit testauslogiikalle. Jako tehdään siis ohjelman, ei sitä suorittavan laitteen kannalta. [21]

Suurin osa tutkimuksesta ei vaikuta täyttävän tätä tässä työssä käytettyä tiukempaa määritelmää mustalaatikkotestauksesta. Esimerkiksi vaikka Tramontana et al. [21] luokittelee 57 heidän käsittelemästään 131 ratkaisusta mustalaatikkopohjaisiksi, vain viisi heidän esittelemäänsä ratkaisua tarkkailee testattavan laitteen tilaa siten, että testauslogiikka olisi tehtävissä myös ilman käyttöliittymäsignaaleja.

Nämä Tramontana et al. esittelemät 5 vaihtoehtoa ovat kuvankaappausten analysointiin perustuvia ratkaisuja. Kuvankaappauksiin pohjaavien ratkaisuiden toiminta perustuu prosessiin, jossa laite ohjelmallisesti ottaa kuvankaappauksen ja laitetta ohjataan kuvankaappausten analysoinnin pohjalta ohjelmallisesti simuloiduin sormen painalluksin. Ratkaisu ei ole täysin mustalaatikkomainen, koska tilan lukeminen ja manipulointi tapahtuu edelleen ohjelmallisesti. Koska kuitenkin käsitellään käyttäjällekin näkyvää kuvaa ja käytetään ohjelmaa simuloiduin kosketuksin, näin saadaan suuri osa mustalaatikkotestauksen hyödyistä ilman fyysistä laitteistoa. Kuvankaappauksiin perustuvia ratkaisuja esittelemiä tutkimuksia on yllättävän vähän ottaen huomioon, että niitä hyödyntäviä työkaluja on helposti löydettävissä.

2.2 Mobiilitestaus fyysisellä robotilla

Poistamalla ohjelmallinen yhteys testattavaa ohjelmaa suorittavan laitteen ja testiä suorittavan laitteen välillä olisi tietenkin äärimmäisin mustalaatikkotestauksen muoto ja siten se teoriassa pystyisi havaitsemaan kaikki viat, joita ihmiskäyttäjäkin. Robottitestaukses-

sa on kuitenkin useita ongelmia: Ratkaisu on ohjelmallista testausta työläämpi ja siten kalliimpi, joko työtunteina tai ostetun ratkaisun hintana.

Robottitestaukseen vaaditaan myös testattava fyysinen laite ja jokainen robotti pystyy testaamaan vain yhtä laitetta kerrallaan, joten usean laitteen samanaikaiseen testaukseen tarvitaan testattavien laitteiden lisäksi myös useita robotteja. Fyysinen laitteisto vaatii myös ylläpitoa ja voi vikaantua.

Haitat käytännössä tekevät testauksen fyysisellä manipuloinnilla epäkäytännölliseksi normaalissa mobiilisovelluskehityksessä. On kuitenkin joitain käyttötapauksia, joissa se olisi hyödyksi tai joissa ohjelmaa ajavaan laitteeseen ei voida tai ei haluta tehdä muutoksia.

Varsinaisen toteutuksen isoin ongelma on se, että jos testattavaan laitteeseen ei ole ohjelmallista yhteyttä, testin onnistumisen tai epäonnistumisen havaitseminen on merkittävästi vaikeampaa kuin ohjelmallisella yhteydellä. Kuten edellisessä luvussa mainittiin [21], yleisin käytetty tilan havainnointi tapahtuu testattavan ohjelman käyttöjärjestelmälle lähettämien signaalien pohjalta ja siten eivät olisi mahdollisia ilman ohjelmallista yhteyttä. Käytännössä vaihtoehdoksi jää vain laitteen näytön havainnointi.

Kokonaisratkaisuja tarjoavien yritysten löytäminen on yllättävän vaikeaa, mutta niitäkin on, kuten Optofidelity [15]. Näillä on kuitenkin yleensä tapana vain mainita tarjoavansa testauspalveluita, ei myydä tuotteita tai ainakaan esitellä tuotteita nettisivuillaan, joten niiden vertailu on haastavaa.

Kokonaisratkaisuiden lisäksi on tarjolla joitain kosketusnäyttötestaukseen tarkoitettuja robotteja, kuten Tactile Automationin taktouch-1000 [19]. Ne puolestaan vaikuttavat tarjoavan vain näytön manipuloinnin, jättäen varsinaisen kosketuslogiikan asiakkaan toteutettavaksi.

On kuitenkin joitain esimerkkejä ei-kaupallisesti projekteista, joista on saatavilla enemmän tietoa käytetyistä työkaluista ja rakenteesta. Varsinaisia perusteellisia raportteja ei kuitenkaan ole saatavilla niistäkään. Hyvin lähelle tämän työn tavoitetta vastaavia projekteja löytyi kaksi: TestDevLab esittelee blogissaan [7] ohjelmalliseen kuvankaappaukseen pohjaavan ratkaisun ja Cracinescu et al. [2] toteuttivat kameraan pohjaavan ratkaisun.

TestDevLabin projekti [7] päätyi alunperin käyttämään suorakulmaista robottia, mutta luopui siitä. Syiksi listattiin sen olevan haastava rakentaa, sen vaatiman pöytätilan olevan suuri sekä ongelmat sen kalibroinnissa eri testattaville laitteille. Projektissa päädyttiin sen sijaan käyttämään vapaan lähdekoodin Delta-robottiprojektia Tapsterbottia [20] ja rakentamaan se itse.

Ohjelmallisesti TestDevLabin projekti integroi robotin Appium -testikehykseen siten, että Appium käskyttää robottia testattavalta laitteelta otetun kuvankaappauksen mukaan. Tarkempaa kuvankäsittelyä ei eritellä, joten oletettavasti ollaan käytetty jotain Appiumille saatavaa valmista kuvantunnistuskirjastoa. Koska hyödynnetään kuvankaappausta, eikä kameralla otettua kuvaa laitteena lähteenä, näitä voidaankin käyttää suoraan ilman tarvetta ottaa huomioon kameran aiheuttamia virheitä.

Craciunescu et al. tutkimus [2] keskittyy ilmeisesti tuotteistamistarkoituksessa luotuun järjestelmään. He päätyivät käyttämään itse suunnittelemaansa ja rakentamaansa Delta-robottia ja ilmeisesti myös käyttämään omatekemäänsä ohjelmistoa ilman valmiin testikehyksen tarjoamia ominaisuuksia päätellen annetuista kuvantunnistusesimerkeistä. Robotti käytti kolmea osoitinkynää, joita voitiin liikuttaa toisiinsa nähden, testattavan laitteen manipulointiin käytettävänä työkaluna.

Craciunescu et al. käyttivät projektissaan suhteellisen halpaa kameraa, joka oli kiinnitetty robotin työkaluun. Raportti esittelee suhteellisen tarkasti käytetyn kuvantunnistusmekanismin (feature matching käyttäen SURF algoritmia) ja itse suunnitellun Delta-robotin rakentamisen, mutta ei mainitse testien suorituksesta mitään. Suorituskyvystä mainitaan vain se, että sitä on testattu perusteellisesti useilla laitteilla yli 2000 tuntia ja se on havaittu erittäin luotettavaksi. [2]

Näiden lisäksi löydettiin myös hieman eri tavalla asiaa lähestynyt tutkimus [13], jossa Mao et al. esittelevät tavan yhdistää täysin ilman ohjelmallista yhteyttä toimivaa testausta ohjelmallista yhteyttä hyödyntäviin menetelmiin käyttämällä kahta laitetta. Heidän järjestelmässään testi suoritetaan ensimmäiselle laitteelle ohjelmallisen yhteyden läpi. Suorituksen aikana tarkkaillaan laittelle välitettyjä käskyjä. Nämä käskyt tulkitaan sitten testin fyysisen suorituksen vaatimiksi kosketuksiksi ja kosketukset suoritetaan robotilla toiselle laitteelle, johon testausjärjestelmällä ei ole ohjelmallista yhteyttä. Näin toimimalla pystytään kameralla varmistamaan, että ohjelmallisesti ja fyysisesti ohjattujen laitteiden näytöt vastaavat toisiaan.

Menetelmällä saadaan siis sekä ohjelmallisen, että fyysisen testauksen tulokset ja voidaan käyttää ohjelmallisen testauksen keinoja testien määrittelyyn. Valitettavasti varsinainen toteutus esitellään tässäkin artikkelissa hyvin suurpiirteisesti ja esimerkiksi kuvantunnistuksesta mainitaan vain, että se toteutettiin hyödyntämällä OpenCV:tä.

2.3 Tutkimuskysymys ja prototyypin vaatimusmäärittely

Edellisen luvun kirjallisuuskatsauksen peristeella todetaan, että mobiililaittiden testaus fyysisesti on vähemmän käytetty ja tutkittu testaustapa verrattuna ohjelmalliseen yhteyteen perustuvaan testaukseen. Tämän työn aiheena on tutkia fyysisen testauksen toteutumismahdollisuuksia. Tarkoitus on rakentaa asian tutkimiseksi mobiililaitteen testausjärjestelmän prototyyppi. Laitteiston rakentamisen kustannukset olisi pidettävä mahdollisimman pienenä.

Tämän työn tutkimuskysymys on siis: Onko mahdollista tehdä mobiililaitteen fyysiseen manipulointiin perustuva testausrobotti kustannustehokkaasti?

Robotin päävaatimukseksi asetetaan kyky suoriutua yksinkertaisten ’paina kuvaketta, paina toista kuvaketta, tarkista näkyykö kolmas kuvake’-tason testien suorittamisesta esimerkkisovellukselle. Lisävaatimuksena ratkaisun on havainnoitava sovelluksen tilaa

ja vaikutettava testattavaan sovellukseen sitä ajavaa laitetta fyysisesti käsitellen.

Robotti, sekä sen vaatima ohjelmisto, toteutetaan vain niiltä osin kuin on tarpeellista toteutuskelpoisuuden arviointiin. Järjestelmä toteutetaan kuitenkin riittävän pitkälle, että sitä voidaan käyttää ohjelmallisen lähestymistavan toimivuuden arvointiin.

Edellämainittujen vaatimusten lisäksi robotin toimintaa testataan myös joissain vaativammissa käyttötapauksissa ja laitteelle on tarkoitus tehdä ainakin luotettavuustestejä. Näiden mittausten tuloksia ei kuitenkaan käytetä toteutuskelpoisuuden kriteereinä vaan ne esitetään vain suuntaa-antavina tuloksina ratkaisun luotettavuudesta. Perusteena tälle on se, että laitteen suorituskykyä tullaan parantamaan kunnes se on riittävällä tasolla tarkoitukseen, mutta vain siihen asti, joten mitattua tulisi 'tähän tarkoitukseen riittävä' suorituskyky, eikä se johon laitteisto voisi pidemmälle kehitettäessä kyetä.

Tutkimuskysymys vaatii siis ratkaisun kehittämisen oikean maailman ongelmaan. Täten tutkimusmetodiksi kannattaa valita niinsanottu konstruktiivinen menetelmä. Konstruktiivisessa menetelmässä tutkimuksessa tutkimusaiheeseen tutustutaan käytännön konstruktion avulla, eli kehitetään ratkaisu käytännön ongelmaan ja analysoidaan luotua konstruktiota sekä sen luomisprosessia. [11]

Konstruktiivinen menetelmä on yleiskäyttöinen, joten sitä on tarpeen hieman lyhentää ja mukauttaa tähän käyttötapaukseen. Lukan [11] listauksesta tässä yhteydessä olennaiset konstruktiivisen menetelmän prosessin vaiheet ovat:

1. Ongelman valinta
2. Ongelma-alueeseen tutustuminen
3. Ratkaisun innovointi, toteutus ja testaus
4. Prosessin teoreettisen kontribuution analysointi

Ensimmäisen ja toisen askeleen tavoite on hakea ongelma jota ei ole ratkaistu ja jonka ratkaisusta olisi tieteellistä hyötyä sekä tutustua ongelma-alueeseen siten, että pystytään käyttämään ongelman ratkaisuun oikeita työkaluja ja tunnistamaan tieteellinen hyöty. Näiden askeleiden jälkeen on oleellista, että ongelma on määritelty siten, että sen ratkaisu on kuvailtavissa konkreettisena tuotteena. Tässä DI-työssä tämä toteutetaan tekemällä kirjallisuuskatsaus mobiilitestauksen tutkimuksesta sekä saatavilla olevista työkaluista.

Kolmannen askeleen tavoite on lyhyesti ratkaista ongelma. Tässä työssä tämä tarkoittaa prototyypin luomista järjestelmästä joka toteuttaa vaaditun toiminnallisuuden ja toteutuksen sekä testaukseen ja arviointiin.

Neljännessä askeleessa analysoidaan luotua ratkaisua ja luomisprosessin aikana saatua kokemusta tavoitteena jalostaa tutkimuksessa kerätty tieto laajemmin hyödylliseen muotoon. Tämän työn tapauksessa tämän askeleen lopputulos on vastaus tutkimuskysymykseen sekä arviot kuinka yleistettävissä työn kokemukset ovat. Lisäksi tulokset olisi hyvä sitoa muihin vastaaviin tutkimuksiin. Tämä on kuitenkin tässä yhteydessä haastavaa, koska kuten edellisissä luvuissa esiteltiin, vastaavia tutkimuksia on saatavilla, mutta

niissä tehdyistä ratkaisuista on saatavilla vain vähän tietoa. Kunnollista vertausta ei siis kyetä tekemään vaan joudutaan vain esittämään tässä yhteydessä saadut tulokset.

Askeleen yksi pohjlta askeleessa kolme rakennettavalle prototyypille on nyt asetettava vaatimukset. Tutkimuskysymyksen pohjalta prototyypin päävaatimus on se, että järjestelmä pystyy testaamaan mobiililaitetta ilman ohjelmallista yhteyttä. Tämän vaatimuksen toteutuminen yleisellä tasolla validoidaan käyttämällä luotua järjestelmää jonkin testattavalle laitteelle saatavilla olevan ohjelman testaukseen.

Tutkimuskysymys on hieman ongelmallinen siksi, että vaikka tutkimuskysymykseen vastaus on suoraviivaista yleisellä tasolla, lähes yhtä oleellista olisi arvioida toteutuksen laatua. Tämä on kuitenkin vaikea tehdä objektiivisesti, koska vastaavista järjestelmistä ei ole saatavilla tarkempaa tietoa ja siten toteutusta ei voida verrata mihinkään vastaavaan. Se joudutaan siten tekemään yleisellä tasolla raportoiden.

Olisi myös hyödyllistä arvioida ratkaisun vaatimaa työmäärää ja siten kustannusarviota, mutta projektiteknisistä syistä tämä jätetään tekemättä: Prototyyppi toteutetaan osaamiskehittämisprojektina ja siten siihen käytetty aika on epäedustava verrattuna paremmin resurssoituun projektiin, jossa tiimin jäsenet pystyvät työskentelemään tehokkaasti. Tarve pitää kustannukset pienenä pidetään kuitenkin kriteerinä käytettäviä työkaluja arvioitaessa.

3 MOBIILILAITTEEN TESTAUSROBOTTI

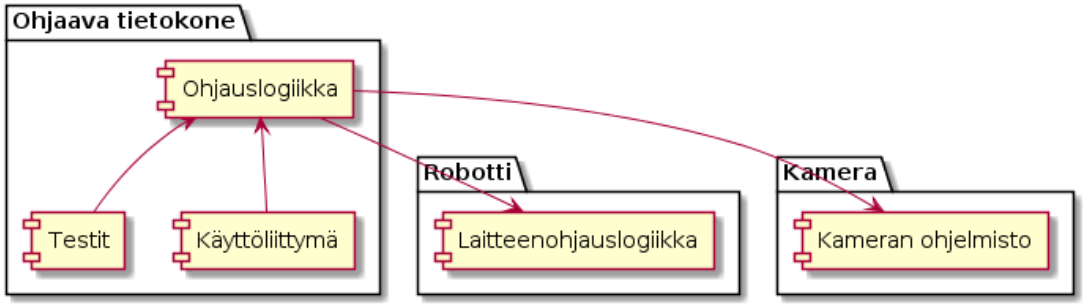
Tässä luvussa eritellään tekniikkoja, työkaluja ja menetelmiä mobiilitestausrobotin rakentamiseen. Mobiiliohjelmiston testausrobotilla tarkoitetaan tässä yhteydessä laitteistoa joka kykenee testaamaan erillistä mobiililaitetta, johon sillä ei ole ohjelmallista yhteyttä.

Mobiilitestausrobotti voidaan rakentaa useammalla tavalla, mutta jokaisen testausjärjestelmän on sisällettävä tietyt hyvin yleisen tason ydintoiminnallisuudet:

- Laitteiston on kyettävä havainnoimaan testattavan laitteen tilaa
- Laitteiston on kyettävä vaikuttamaan testattavan laitteen tilaan
- Laitteiston on kyettävä verifioimaan tilan oikeellisuus ja päättämään miten tilaan pitää vaikuttaa

Hyvin karkealla tasolla tämänlaisen perustoiminnallisuuden toteuttamiseen tarvitaan testattavan laitteen näyttöä manipuloimaan kykenevä robotti, kamera kohdelaitteen tilan havainnointiin sekä järjestelmää ohjaava logiikka. Tämä jako on esitetty kuvassa 3.1.

Tämä jako on vain tapa jakaa järjestelmää pienempiin, ainakin jossain määrin toisistaan riippumattomiin osiin, jotta niitä voidaan käsitellä pienempinä kokonaisuuksina, mutta vastaa todennäköisesti laitteiston fyysistä jakoa. Tässä luvussa keskitytään esittelemään vaihtoehtoja näiden kolmen toiminnallisuuden toteuttamiseen.



Kuva 3.1. *Järjestelmän korkean tason rakenne*

3.1 Järjestelmän toimintaperiaate

Ensin on hyödyllistä käsitellä järjestelmän yleisen tason toiminta, koska se käytännössä asettaa vaatimukset testattavan laitteen tilan havainnoinnille ja testattavan laitteen tilaan vaikuttamiselle. Tämä johtuu siitä, että vaatimukset järjestelmän kyvylle havainnoida testattavan laitteen tilaa ja vaikuttaa siihen riippuvat siitä mitä testataan ja miten. Näihin kysymyksiin vastataan yleisellä tasolla luvussa 2 määriteltäessä tutkimuskysymystä.

Tutkimuskysymys ei erikseen rajoita käyttämään manuaalisesti luotuja testejä. Luvussa 2.1 esitelty testien luonnin automatisointi ei kuitenkaan ole realistinen vaihtoehto. Koska testattavasta laitteesta saatava tieto on virheherkkää ja rajallista, olisi suora automatisointi haastavaa käyttämättä esimerkiksi luvussa 2.2 esiteltyä kahden laitteen ratkaisua [13]. Siinä testien suorittamiseen tarvitaan kuitenkin ohjelmallinen yhteys laitteeseen ja siten se ei täytä tutkimuskysymyksen asettamia ehtoja. On siis paras rajoittaa testauslogiikka suhteellisen yksinkertaiseen, manuaalisesti luotuihin testiaskeliin pohjaavaan ratkaisuun.

Testiaskelpohjainen testaus on niin yksinkertaista, että sen tekeminen itse ei olisi kovin työlästä. Testausjärjestelmän on kuitenkin toteutettava useita toiminnallisuuksia, kuten raportointi ja erilaiset tavat testien suorittamiseen, joiden kaikkien toteuttaminen riittävällä tasolla on työlästä. Lisäksi testien luonti on todella tärkeä osa testausjärjestelmän käyttäjäkokemusta ja vaikkei se olekaan testattavan laitteen käyttäjille näkyvää käyttäjäkokemusta, hyvä testausjärjestelmän käyttäjäkokemus auttaa testausinsinöörejä luomaan parempia testejä ja paikantamaan testien havaitsemat poikkeukset nopeammin, joten testausjärjestelmän käyttäjäkokemukseen kannattaa silti panostaa. Valmiin työkalun muokkaaminen tarkoitukseen on siis todennäköisesti nopeampaa kuin laadukkaan järjestelmän luominen tyhjästä.

Askelpohjaiseen testaukseen on tarjolla suuri määrä erilaisia työkaluja ja ratkaisut usein rakennetaan jotain valmista työkalua hyödyntäen. Esimerkiksi Kong et al. havaitsivat, että suuri osa heidän tutkimistaan testausratkaisuista oli rakennettu Robotium -testikehystä hyödyntäen [10]. Tässä käyttötapauksessa useista varsinaiseen mobiiliohjelmiston testaukseen tarkoitettujen työkalujen kehittyneemmistä ominaisuuksista ei ole hyötyä, koska työkalu ei käytä testattavaa mobiililaitetta suoraan. On todennäköisesti paras valita työkalu sen pohjalta mistä kehittäjillä on kokemusta. Erityistapaus on kuvapohjaiseen testaukseen tarkoitetut työkalut kuten Sikuli [18], koska niiden raportointi ja testien esitys tarjoaa paremman tuen kuville. Valmista työkalua käytettäessä rakennetaan työkalulle erillinen logiikka, joka esikäsittelee sille annettavat kuvat ja tulkitsee sen antamat käskyt ohjaukseksi robotille.

3.2 Testattavan laitteen tilan havainnointi

Tässä luvussa esitellään keinoja havainnoida testattavan laitteen tilaa.

Ohjelmistojen testauksessa ohjelman tilaa havainnoidaan yleensä ohjelmallisesti eli tulkitsemalla sen paljastamaa tietoa erilaisten rajapintojen läpi. Erityisesti käyttöliittymätestauksessa on kuitenkin hyödyllistä analysoida näkymää, koska silloin voidaan havaita suurempi osa ohjelman virheistä.

Visuaalisen näkymän käsittelyyn onkin tarjolla erilaisia palveluita ja työkaluja, kuten edellisessä luvussa esiteltiin. Saatavilla olevat työkalut ovat yleensä toiminnallisesti valmiita eli ne sisältävät kaiken ohjelman testaukseen vaadittavan toiminnallisuuden. Osaa näistä työkaluista voidaan käyttää myös mobiililaitteen testaukseen ja on myös erityisesti siihen tarkoitettuja työkaluja.

Nämä työkalut odottavat kuitenkin saavansa kuvankaappauksen eli ohjelmallisesti laitteelta tallennetun kuvan sen näytöstä. Kuvankaappaus on ohjelmallisesti kopioituna virheetön.

Ne eivät siis suoraan toimi kameran ottamalla kuvalla, koska kameran ottama kuva ei koskaan ole täysin virheetön vaan sisältää eri syistä johtuvia vääristymiä ja täysin samastakin testattavan laitteen näkymästä otettu kuva voi siten sisältää joitain eroja.

Kameran avulla toimivan mobiililaitteen testausrobotin on siis pystyttävä analysoimaan ovatko testattavan laitteen näytössä havaitut erot merkityksellisiä vai ovatko ne vain epätäydellisestä anturista johtuvia häiriöitä. Virheiden kompensointia voi ja kannattaakin lähestyä sekä käsittelemällä kuvaa niiden vähentämiseksi, että valitsemalla kuvan analysointitavat siten, että ne ovat virhesietoisia.

Varsinainen tieto, mitä testattavan laitteen tilasta halutaan, riippuu siitä mitä tarkalleen ottaen testataan, mutta jotta asiaa pystytään käsittelemään tarkemmin, tässä luvussa keskitytään pääasiassa tekniikoihin, joilla annettua kuvaa voidaan hakea testattavan laitteen näytöltä. Tähän kykenevä järjestelmä pystyy tarkistamaan yleisiä tilaan liittyviä asioita (’näyttääkö testattava sovellus halutun kuukauden’), tekemään yksinkertaisia rakennetarkistuksia (’onko otsikkopalkki näytön yläreunassa’) ja tekemään myös useimpia muita testaustyyppejä, esimerkiksi mittaamalla viiveitä nopeustesteihin.

Tämän perustoiminnallisuuden lisäksi olisi hyödyllistä myös kyetä tarkistamaan näytöllä näkyvän tekstin sisältö, jottei siinä tarvitsisi luottaa kuvavertaukseen. Lisäksi olisi myös hyödyllistä pystyä vertaamaan koko näkymää, mahdollisesti tietyin rajauksin. Nämä ovat kuitenkin toissijaisia verrattuna kuvan hakemiseen näkymästä.

Yksinkertaistaen tilan havainnointi voidaan jakaa kahteen osa-alueeseen: Mahdollisimman virheettömän kuvan saamiseen testattavan laitteen näytöstä ja saadun kuvan analysointiin. Nämä esitellään seuraavissa aliluvuissa.

3.2.1 Kuvan ottaminen

Kuten edellisessä luvussa mainittiin, kuvantunnistuksen prosessi alkaa kuvan ottamisesta. Kuvan ottamiseen käytettävällä kameralla onkin siten suuri merkitys järjestelmän suo-

rituskykyyn.

Sopivan kameran vaatimusten määrittely on kuitenkin hieman haastavaa lukuisien kameratyyppien ja ominaisuuksien takia: Krig listaa esimerkiksi sensorityypin, sensorimateriaalin, anturikoon, anturikaavan, väritarkkuuden ja lukuisia muita päätettäviä arvoja. [12, s. 2 - 7]

Näistä suuri osa vaikuttaa kuitenkin olevan oleellinen vain tietyissä vaativissa käyttötapauksissa. Esimerkiksi sulkimen tyyppi on merkittävä, jos kohde muuttuu nopeasti ja anturin koko on merkittävä heikossa valaistuissa olosuhteissa. Lisäongelma on, että vaatimukset riippuvat myös järjestelmän käytöstä, ja rakenteesta: Halutaanko järjestelmällä pystyä esimerkiksi havaitsemaan kuolleita pikseleitä tai mitata tarkasti animaatioita? Halutaanko kamera sormen viereen, jolloin sen koolla ja painolla on merkitystä?

Tältä pohjalta oletetaan siis, että jollei käyttötapaus sisällä mitään erityisiä vaatimuksia, kameran ainoa oleellinen vaatimus on riittävä tarkkuus. Riittävä tarkkuus on vähintään kaksinkertainen testattavan laitteen näytön resoluutioon verrattuna, jos testattavan laitteen näyttö täyttää kameran koko näkymän. Tämä vaatimus on johdettu siitä, että Nyquistin näytteenottoteoreemaa soveltamalla havaitaan, että kameran resoluution olisi oltava vähintään kaksinkertainen kohdelaitteen näytön resoluutioon verrattuna, jotta teoriassa pystyttäisiin saamaan siitä pikselintarkka kuva [12, s. 2]. Tämä ei kuitenkaan ole tiukka vaatimus, koska järjestelmän käsittelemät kohteet ovat yleensä käyttöliittymäelementtejä ja siten selkeästi pikseliä suurempia ja on tarkoitettu antamaan arvio riittävän tarkkuuden kokoluokasta. Jos kameralla on tarkoitus havaita kohdelaitteen pikselin kokoisia virheitä, esimerkiksi hakemaan näytön virheitä, kuten kuolleita pikseleitä, tarkkuuden olisi oltava tätäkin parempi [12, s. 2].

Kameralla otettu kuva ei koskaan vastaa täysin testattavan näytön esittämää kuvaa edes ideaalitilanteessa. Mobiililaitteen testaus on käyttötapauksena suhteellisen helppo, koska kuvattava kohde on lähellä, paikallaan ja olosuhteet, kuten valaistus, voidaan valita optimaalisesti. Riippuen valittavasta robottityypistä, kamera saatetaan myös pystyä asettamaan suoraan kulmaan testattavan laitteen näyttöön verrattuna.

Kameralla otetun kuvan voidaan siis olettaa sisältävän erilaisia virheitä, kuten kohinaa, väriväristymiä, kuolleita pikseleitä sekä geometrisiä vääristymiä. Kohina, väriväristymät sekä kuolleet pikselit aiheutuvat siitä, että kameran sensori koostuu suuresta määrästä pieniä valoherkkiä soluja, joiden mittaustuloksista koostetaan paras arvio kameran edessä olevasta näkymästä. Virheitä aiheuttaa sekä solujen epätäydellisyydet että ongelmat yhtenäisen kuvan muodostamisessa yksittäisistä datapisteistä. [12, s. 2]

Geometriset vääristymät johtuvat pääasiassa kameran linssistä ja kameran asennosta ja etäisyydestä kohdelaitteeseen. Näiden vääristymien korjaukseen on lukuisia prosesseja, mutta niiden kaikkien käyttäminen ei välttämättä paranna kaikkien kuvantunnistusmenetelmien toimintaa. Siten Krig suositteleekin, että esikäsittelytoimenpiteet kannattaa valita käyttötarkoituksen mukaan. [12, s. 41]

On kuitenkin joitain hyvin todennäköisesti tarpeen olevia toimenpiteitä, joita kannattaa

tehdä riippumatta käyttötarkoituksesta. Krig listaa valaistuksen, tarkennuksen ja kohinanpoistamisen yleisesti hyödyllisiksi toimenpiteiksi ja ainakin halvalla (ja siten pienilinssisellä) kameralla on todennäköisesti hyödyllistä lisätä listaan geomentriakorjaukset. [12, s. 41]

Näiden yleisempien toimenpiteiden lisäksi on yksi erityisesti tässä käyttötapauksessa hyödyllinen kuvan esikäsittelytoimenpide: Jos kamera asetetaan siten, että koko testattavan laitteen näyttö mahtuu sen näkökenttään, on todella hyödyllistä rajata käsiteltävä kuva sisältämään vain testattavan laitteen näytön ja kääntää se pystyasentoon. Näin poistetaan mahdollisesta laitteen alustasta aiheutuvat kuvankäsittelyhäiriöt, parannetaan geometristen vääristymien kompensointitarkkuutta ja mahdollistetaan testejä luodessa viittaaminen testattavan näytön koordinaatteihin. Lisäksi näin toimimalla mahdollistetaan myös valmiiden testikehysten käyttö jos niitä halutaan käyttää. Jos näin pystytään toimimaan, se on hyvä tehdä yhteisesti kaikille kuville, koska se muuttaa tarvittavia koordinaattikonversioita. Siten se kannattaa tehdä kootusti, jotta vältetään tarve käsitellä useita koordinaatistoja muissa ohjelman osissa.

3.2.2 Kuvan analysoinnin perustoiminta

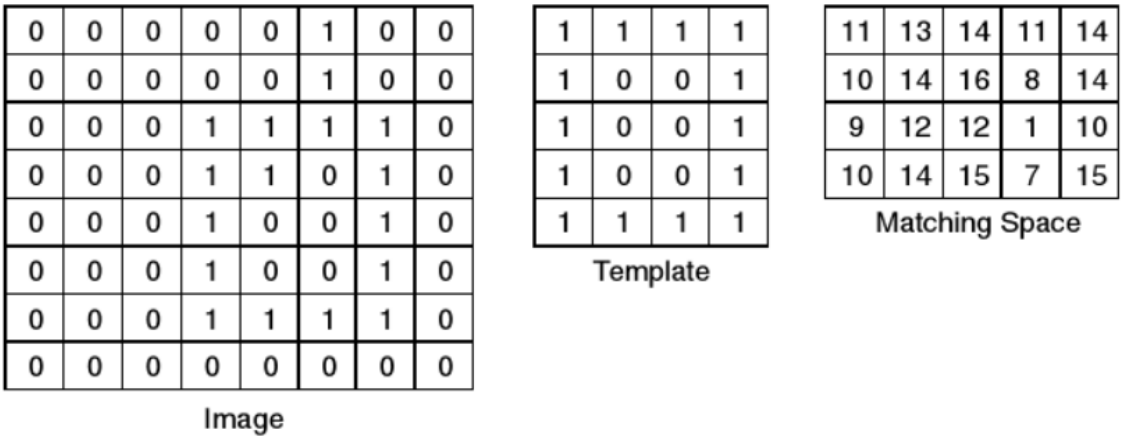
Kuvan analysoinnissa olennaisin toiminnallisuus on kyetä hakemaan testattavan laitteen näytöltä haluttua kuvaa. Se on yksinkertaisin tapa testata. Se on oleellinen toiminnallisuus myös silloin, jos varsinainen tilan analysointi tehdään jollain eri periaatteella, koska ilman sitä testattavan laitteen manipulointi on tehtävä ottamalla muistiin painallusten koordinaatit.

Tämä toiminnallisuus voitaisiin toteuttaa yksinkertaisesti vertaamalla haettavaa kuvaa jokaiseen näkymän pisteeseen. Jos jokainen haettavan kuvan pikseli vastaa näkymän sen kohdan pikseliä, voidaan todeta haettavan kuvan olevan siinä pisteessä näkymää. Tämä on myös ainoa tapa, jolla on niin sanotusti todistusarvoa eli jos tällä tekniikalla löydetään vastaavat pikselit, tiedetään täysin varmasti haettavan kuvan olevan kohdekuvassa.

Kuten edellisessä luvussa esiteltiin, kameran ottamassa kuvassa on kuitenkin aina vääristymiä. Siten tämä yksinkertainen ratkaisu olisi todennäköisesti epäluotettava.

Lisäongelmana suorassa pikselivertauksessa on se, että vaikka periaatteessa olisikin arvokasta käyttää menetelmää, joka pystyy osoittamaan näkymän oikeellisuuden pikselintarkasti, käytännössä tämän ominaisuuden arvo on hyvin käyttötapausriippuvainen: Esimerkiksi testattaessa kuvan näyttämistä tai näytön ominaisuuksia olisi pikselintarkka vertaus tarpeen, mutta testattaessa esimerkiksi mobiililaitteille luotavaa ohjelmaa eri laitteilla, pikselierojen havaitseminen voi usein olla jopa haitallista. Tämä johtuu siitä, että eri laitteet ja eri alustaversiot aiheuttavat lieviä eroja näkymään, jotka haluttaisiin jättää huomiotta jottei jokaiselle testattavalle laitteelle tarvitse luoda omia testejä.

Näiden vääristymien aiheuttamia ongelmia voidaan lähestyä kahdella tavalla: Ensinnä-



Kuva 3.2. *Template matchingin toiminta[3, s. 134]*

kin kameran ottamassa kuvassa olevia vääristymiä voidaan kompensoida edellisessä luvussa esitellyin menetelmin. Toisekseen vertaukseen voidaan käyttää suoraa pikselivertausta kehittyneempää ratkaisua, joiden esittelyyn keskitytään tässä luvussa. Menetelmät eivät ole toisiaan poissulkevia, vaan vääristymien kompensointia kannattaa tehdä joka tapauksessa.

Pienemmän kuvan hakemista isommasta kuvasta pikselivertauksella nimitetään Template Matchingiksi. Siinä haettavaa kuvaa yksinkertaisesti verrataan pikseli pikseliltä jokaiseen sen kuvan kohtaan, josta sitä haetaan. Jokaiselle kohdalle lasketaan kuinka paljon sen kohdan kohdekuvan pikselit eroavat haettavan kuvan vastaavan kohdan pikselistä. Jos löytyy paikka, jossa haettavan kuvan pikselit vastaavat täydellisesti kohdekuvan pikseleitä, tiedetään, että siinä kohdassa kohdekuvaa on täydellinen kopio haettavasta kuvasta. Menetelmä onkin erinomainen kun verrataan täysin ohjelmallisesti saatavia kuvia, esimerkiksi haettaessa ikonia tietokoneen näytöltä ohjelmallisesti.[3, s. 134]

Toiminta on esitelty hyvin yksinkertaisessa tilanteessa kuvassa 3.2. Vasemmanpuoleinen taulukko (Image) esittää 64 pikselin kuvaa, josta haettavaa kuvaa haetaan ja keskimmäinen taulukko (Template) esittää 20 pikselin haettavaa kuvaa. Esimerkkitilannetta on yksinkertaistettu myös käyttämällä vain kahta väriä sisältäviä kuvia. Viimeisessä kuvassa on näille kuville tehdyn template matchingin tulostaulukko (Matching Space). Tulostaulukko on muodostettu laskemalla jokaisen paikan, jossa keskimmäinen taulukko mahtuu kokonaan vasemmanpuoleiseen taulukkoon, eroavien pikseleiden erojen neliöiden summa. Menetelmän paras arvio haettavan kuvan paikasta isommasta kuvasta on pienin summa eli tässä tapauksessa tulos olisi siis oikealla alhaalla oleva 1.

Tilanteissa, joissa kohdekuva on vääristynyt, template matching selviää suhteellisen hyvin koko kohdekuvaan samalla tavalla vaikuttavissa eroissa, kuten lievistä värieroista tai isoistakin eroista hyvin pienessä osassa kohdekuvassa olevaa haettavaa kuvaa, kuten vaikka haettavan kuvan nurkan peittyminen. Menetelmä selviää myös hyvin valaistuseroista. [3, s. 134]

Menetelmässä on kuitenkin merkittäviä ongelmia tässä käyttötapauksessa: Template matc-

hing vertaa vastinpikseleitä suoraan, joten jos kohdekuva on lievästikkään vääristynyt esimerkiksi siten, että kohdekuvassa haettava kuva on joitain yksittäisiä pikseleitä leveämpi, template matchingin vastepikselivertaus voi olla virheellinen lähes koko kuvassa. Ongelmaa pahentaa vielä se, että jos kameralla otetun kuvan resoluutio ei ole riittävä tarkoitukseen, kuvan elementit voivat helposti siirtyä yksittäisiä pikseleitä. Samasta syystä pienikin kohdekuvan kierto voi aiheuttaa virheellisiä tuloksia.[3, s. 134]

Menetelmän vaatimus täydelliseen koko- ja muotovastaavuuteen on ongelmallinen myös täysin vääristymättömän kuvan tilanteessa siksi, että jos testattavat laitteet eivät ole täysin toisiaan vastaavia näytöltään, kuvakkeet voivat hyvin olla lievästi erikokoisia tai erimuotoisia järjestelmän kameran kuvassa. Testit, tai ainakin testien käyttämät kuvat olisi siis tehtävä jokaiselle testattavalle laitteelle erikseen. Tämä on menetelmän merkittävimpiä haittoja, koska testattaessa mobiililaitteiden ohjelmia testausautomaation suurimpia hyötyjä on ohjelman testaus useilla alustoilla.

Verrattava kuva olisi myös käytännössä pakko ottaa testattavan ohjelman näytöltä käyttäen järjestelmän kameraa. Ei olisi mahdollista esimerkiksi käyttää testattavan ohjelman käyttämiä ikonitiedostoja, mikä tarkoittaa sitä, että testejä ei olisi mahdollista luoda ilman fyysistä laitetta.

Näiden ongelmien ratkaisuun on joitain tekniikoita. Yksinkertaisin on tehdä haku haettavasta kuvasta useita kertoja varioiden hieman käännettyjä versioita tai hieman isompia tai pienempiä versioita haettavasta kuvasta. Tämän tavan ongelma on sen vaatima prosessointiaika: Hakemalla myös hieman käännettyjä versioita haettavasta kuvasta joudutaan koko prosessi suorittamaan jokaiselle käännetylle versiolle. Lisäämällä myös koon variointia, joudutaan prosessi suorittamaan jokaisen koon jokaiselle asennolle. Menetelmää voidaan toki optimoida esimerkiksi tekemällä esihaku pienemmällä resoluutiolla ohjaamaan kuvan prosessointijärjestystä ja lopettamalla haku välittömästi riittävän hyvän osuman löytyessä. Nämä optimoinnit ja erityisesti isommat asennon ja koon varioinnit heikentävät kuitenkin menetelmän luotettavuutta erityisesti pieniä ja yksinkertaisia kuvia haettaessa.

Jos näitä tekniikoita joudutaan käyttämään runsaasti tai tunnistuksen luotettavuus ei ole niistä huolimattakaan riittävä, on todennäköisesti parempi testata muita tekniikoita, koska käytännössä vertaus siirtyy joka tapauksessa kauemmaksi suorasta pikselivertauksesta ja siten vertauksen todistusarvo vähenee.

Näistä syistä onkin todennäköistä, että perinteisistä mentelmistä varteenotettavimpana vaihtoehtona voidaan tunnistaa haettavasta kuvasta erotettavia pisteitä ja sitten hakea samoja, samassa asennossa toisiinsa nähden olevien erotettavien pisteiden joukkoa kohdekuvasta. Tätä tekniikkaa kutsutaan feature matchingiksi.

Feature matchingillä tehtynä kuvan hakemisprosessi aloitetaan hakemalla sekä haettavasta kuvasta, että kohdekuvasta avainpisteitä. Mikä tarkalleen ottaen voi olla avainpiste, vaihtelee riippuen käytetystä algoritmista, mutta yleistäen ja yksinkertaistaen algoritmit haluavat löytää kuvasta kulmat. Tämä johtuu siitä, että kulma on suhteellisen helposti

löydettävissä näkymästä, se on yleinen ja sen paikka on helppo määritellä yksiselitteisesti. [3]

Käytännössä tämä tapahtuu yksinkertaisimmillaan hakemalla niin sanottuja Harrissin kulmia eli pisteitä joissa pisteiden kirkkaus vaihtuu nopeasti kummallakin akselilla. Näitä kulmia voi käyttää suoraan ja on myös muita algoritmeja, jotka käytännössä vain tekevät saman asian eri tavalla. Esimerkiksi FAST (Features from Accelerated Segments Test) vertaa vierekkäisiä pikseleitä eikä laske muutosnopeuksia. [9, s. 527-]

Kehittyneemmät algoritmit lisäksi suodattavat löytyneistä avainpisteistä parhaat ja lisäävät niihin myös lisätietoa. Esimerkkinä SIFT (Scale Invariant Feature Transform), suodattaa löytämiään avainpisteitä esimerkiksi etsimällä kulmat useista skaaloista sekä hylkäämällä pienen kontrastin kulmat. SIFTi lisää avainpisteisiin tiedon suunnasta ja kulmien ympäristöstä mahdollistaen tarkemman kulmien erottamisen toisistaan. [3, s. 126]

Varsinainen hakeminen tehdään nyt suorittamalla tämä prosessi sekä näkymälle josta kuvaa haetaan sekä haettavalle kuvalle ja hakemalla vastaavat avainpisteet kummastakin kuvasta. Jos haettavan kuvan avainpisteet löytyvät näkymästä tietystä kohdasta, oletetaan haettavan kuvan olevan siinä kohdassa näkymää. Näkymän tai kuvan ollessa vääristynyt avainpisteet eroavat hieman toisistaan jolloin voidaan sallia hieman eroavaisuutta. Yksinkertaisin tapa tehdä tämä on verrata pisteitä yksi kerrallaan ja laskemalla eron etäisyys. Vertaamiseen on myös mahdollista käyttää kehittyneempiä menetelmiä, mutta niiden päähyöty on tilanteissa joissa haettavia kuvia on useita ja siten suora pistevertaus on raskas. [9, s. 573-]

Feature matchingin vahvuus verrattuna template matchingiin on se, että lievä vääristyminen joka siirtää kaikkia pikseleitä ja voi siten antaa täysin kelvottoman tuloksen template matchingissä, siirtää feature matchingissa vain hieman joitain avainpisteitä ja siten tilanteet, joissa jokainen pikseli on hieman siirtynyt vaikuttaa tulokseen hyvin vähän. Ongelmana on, että vertaus tehdään avainpisteille, ei koko kuvalle ja siten menetetään todistusarvoa, että oikeasti löydettiin haluttu kuva. Lisäksi esimerkiksi väri jätetään kokonaan huomiotta.

3.2.3 Kuvan analysoinnin vaihtoehdot

Edellisessä luvussa esitellyt tekniikat ovat todennäköisimmin paras lähtökohta kuvantunnistuksen toteutukselle, mutta niille on muutamia parannusalueita:

- Tekstin tunnistus
- Näkymän rakenteen testaus
- Koneoppimiseen perustuvat keinot edellisten ratkaisuun

Ehkä tärkeimpänä erikoistapauksena testatessa on usein hyödyllistä pystyä lukemaan tekstiä, koska toisin kuin kuvaa hakiessa, tekstiä hakiessa ollaan kiinnostuneita nimenomaan tekstin sisällöstä. Esimerkiksi rivityserot eivät välttämättä ole oleellisia. Lisäksi

Kuva 3.3. *Kuvan jakaminen kahteen väriin [3, s. 50]*

erot, kuten puuttuvat välimerkit, voivat olla hyvin pieni osa haettavaa kuvaa ja siten vaikeasti havaittavissa kuvahaulla. Siten olisi parempi tarjota erikseen tekstin tunnistukseen tarkoitettu toiminnallisuus.

Tekstin tunnistukseen käytetään menetelmää nimeltä OCR (Optical Character Recognition) tai suomeksi optinen merkintunnistus. Yleisesti suositeltu OCR ratkaisu on vapaan lähdekoodin kirjasto nimeltä Tesseract. Tesseract on vanha, mutta edelleen kehitettävä OCR kirjasto. Projektissa ei myöskään oleteta olevan tarvetta tulkita käsin kirjoitettua tekstiä, joka vaikutti yleisimmältä syyltä suositella jotain muuta kirjastoa.

Tekstintunnistuksen prosessi koostuu hyvin yleisellä tasolla kuvan prosessoinnista käytetyn tekstintunnistuskirjaston vaatimaan muotoon, kuvan prosessoinnista merkeiksi ja lopulta kirjainten tunnistamisesta.

Tesseractia käytettäessä esikäsittelyssä kuva jaetaan täysin mustavalkoiseksi siten, että jokainen piste on täysin musta tai täysin valkoinen (tresholding tai binarization). Kuvassa 3.3 on värit tai harmaan sävyt eivät sisällä tekstin ymmärtämiseen oleellista informaatiota, joten niiden poisto kuvasta yksinkertaistaa prosessia. Kuva myös käännetään ja sen vääristymiä korjataan siten, että rivit ovat mahdollisimman suorassa vaakatasossa. Näiden käytännössä pakollisten askelten lisäksi tunnistustarkkuutta voi parantaa myös poistamalla kohinaa, ylimääräisiä ei tekstiä sisältäviä kuvan osia ja varmistaa tekstin olevan riittävän suurta merkkien muodon tunnistamiseksi. [14]

Tesseract tekee kuvalle joitain näistä esikäsittelytoimenpiteistä normaalissa toiminnassaan: Se esimerkiksi tekee mustavalkokonversion käyttäen Otsun metodia kuvan prosessoinnin aluksi ja poistaa tekstiä sisältämättömät alueet tunnistaessaan tekstin rakennetta [14]. Nämä ovat kuitenkin tehty yleiskäyttöisiksi ja siten tunnistustarkkuutta saattaa pystyä parantamaan käyttämällä juuri kohdekäyttötapaukseen sopivaa prosessia [8].

Seuraavassa vaiheessa tekstin rakennetta prosessoidaan kunnes voidaan erottaa merkit. Tesseractin tapa suorittaa tämä askel alkaa siten, että haetaan kuvasta alueet joissa on kuvia tai viivoja, ja poistetaan ne kuvasta. Seuraavaksi kuvasta haetaan tekstialueiden

reunat. Näistä reunoista yhdistellään vastaavat reunat ja lopuksi näiden reunojen ja niiden tekstin rivivälin pohjalta kuvan teksteistä muodostetaan tekstialueita. [14]

Seuraavaksi löydetyistä tekstialueista haetaan yhtenäiset osat. Nämä yhtenäiset osat ovat merkkejä ja merkkien osia joten näitä osia käyttäen voidaan tunnistaa rivit aloittamalla tekstialueen vasemmasta reunasta ja asettamalla jokainen osa sitä lähimmälle riville. Osat prosessoidaan vasemmalta oikealle eikä rivi kerrallaan, jotta lievät rivin epäsuoruudet eivät haittaa prosessia. Löydetyt rivit jaetaan sanoihin merkkien välien avulla. [14]

Lopuksi sanat jaetaan merkkeihin ja tunnistetaan ne. Tesseract käyttää tähän monimutkaista, monivaiheista iteratiivista prosessia tavoitteenaan hyödyntää jokaisen prosessointitason tietoa tunnistuksen muodostamisessa. Vahvasti yksinkertaistaen yksinkertaisin versio tästä prosessista jakaa edellisen vaiheen yhtenäiset osat kirjasinkoon mukaan tunnistettaviksi merkeiksi ja tunnistaa nämä merkit. Tesseractin lisäominaisuuksiin kuuluu esimerkiksi se, että sanojen ja yhtenäisten osien jakamista merkkeihin iteroidaan vertaamalla tunnistettuja sanoja sanakirjaan ja prosessi toistetaan hyödyntäen ensimmäisellä yrityksellä kerättyä tietoa. Näillä ominaisuuksilla ei kuitenkaan ole merkitystä kirjaston käytön kannalta tässä käyttötapauksessa.[14]

Näkymän rakenteen testauksella tarkoitetaan tässä yhteydessä sitä, että sen sijaan, että testissä laitteen tilaa arvioidaan määrittämällä tietyt käyttöliittymäelementit ja niiden paikat, verrataan koko näkymää aiemmin oikeaksi todettuun versioon siitä. Tällöin testin luominen tehtäisiin tallentamalla kuva halutusta näkymästä. Varsinaisessa testauksessa verrattaisiin sitten koko näkymää.

Tässä käytössä edellisessä luvussa esitellyissä tekniikoissa on se ongelma, että verratessa koko kuvaa, joudutaan hyväksymään suhteellisen paljon eroja, koska vertauksessa joudutaan hyväksymään koko näkymän vääristymät. Tästä seuraa esimerkiksi tilanne jossa testattavan käyttöliittymän tekstit puuttuvat, mutta ovat pinta-alaltaan vain 3% näkymästä ja siten sopivat virhemarginaaliin.

Tämän toiminnallisuuden toteuttaminen suoralla pikselivertauksella on todennäköisesti ongelmallinen edellisessä luvussa eritellyistä syistä. Edellisen luvun feature matching -tekniikka voisi kuitenkin toimia ratkaisuna myös tässä yhteydessä. Avainpisteiden vertaus olisi kuitenkin tarpeen tehdä siten, että virheet eivät pystyisi kasautumaan johonkin näkymän alueeseen, joko jakamalla kuva pienempiin osiin tai luomalla sen estävä vertailulogiikka.

Viimeisenä vaihtoehtona on hyödyntää koneoppimiseen perustuvaa versiota jostain aiemmin mainitusta vaihtoehdosta. Erityisesti neuroverkot ovat viime vuosina vallanneet alaa kuvankäsittelyssä, koska on ongelmatyyppejä, jotka ovat haastavia ratkaista algoritmipohjaisilla menetelmillä, mutta ovat yksinkertaisia ratkaista neuroverkoilla. Niin sanotussa luokittelutehtävässä, kun esimerkiksi yritetään tunnistaa onko kuvassa pyörä vai lentokone on tyyppiesimerkki tehtävästä, jossa neuroverkot ovat vahvoja.

Koneoppimismenetelmissä on kuitenkin tässä käyttötapauksessa ongelmana se, että

niissä käytettävä tekniikka alustetaan sopivaan rakenteeseen ja sitten se koulutetaan antamalla sille valtava määrä syötteitä ja siltä odotettavia tulosteita. Tässä käyttötapauksessa tämä tarkoittaisi tarvetta suurelle määrälle järjestelmän kameralla otettuja kuvia, joissa haettavaa kohdetta ei ole näkyvissä sekä sellaisia, joissa haettava kohde on näkyvissä.

Tämän ongelman ratkaisuun on joitain toimintatapoja. Koulutusdataa voidaan luoda esimerkiksi näyttämällä kohdelaitteen näytöllä sekä muokkaamatonta kohdekuvaa, että jonkin verran ohjelmallisesti muokattua kohdekuvaa. Koska näin tiedettäisiin kohdekuvan paikka, pystyttäisiin koulutuskuvat ottamaan automatisoidusti ja ainakin joitain häiriölähteitä, kuten näytön kirkkautta pystyttäisiin myös muuttelemaan ohjelmallisesti. Ongelmaksi tulee kuitenkin se, että koko koulutus pitäisi toistaa kaikille haettaville kuville, joita on helposti paljon jo suhteellisen yksinkertaisessakin ohjelmassa.

3.3 Testattavan laitteen manipulointi

Koska testattavaan laitteeseen ei luoda ohjelmallista yhteyttä, sen kosketusnäyttöä sekä fyysisiä painikkeita on manipuloitava fyysisesti. Käytännössä tähän on luotava yksi tai useampi mekaaninen 'sormi'. Kosketusnäyttöjä olisi todennäköisesti mahdollista manipuloida myös hyödyntämällä tietoa niiden käyttämästä anturitekniikasta, mutta tätä vaihtoehtoa ei tutkita, koska tällainen ratkaisu olisi kosketusnäyttötekniikkariippuvainen, eikä pystyisi manipuloimaan laitteen painikkeita ja mekaaninen sormi simuloi paremmin ihmiskäyttäjää.

Tässä luvussa käydään ensin läpi kosketusnäyttötekniikoita, sitten esitellään vaihtoehdot niitä manipuloimaan kykenevistä mekaanisista sormista ja lopuksi esitellään vaihtoehtoja robotiksi, joka liikuttaa sormen koskemaan haluttuun kohtaan testattavaa laitetta.

3.3.1 Kosketusnäyttö

Kunnollista ajantasaista tutkimusta käytetyistä kosketusnäyttötekniikoista on haastava löytää, mutta vaikuttaa siltä, että mobiililaitteissa yleisimmin käytetyt tekniikat pohjaavat näytön kosketuksen aiheuttamiin muutoksiin näytön sähköisissä ominaisuuksissa, yleisimpänä kapasitanssiin pohjaavat ratkaisut. Esimerkkinä Walker [1, s. 28-31] listaa 18 tekniikkaa, mutta yleisyyden puolesta oleellisia olevat vain resistiiviset ja kapasitiiviset tekniikat. Walkerin käyttämä tutkimus ei tosin sisällä vuoden 2012 jälkeistä tietoa, mutta ainakaan ei-tieteelliset lähteet eivät anna aihetta olettaa, että tilanne on oleellisesti muuttunut.

Kapasitiivisella periaatteella toimivissa näytöissä kosketuksen havaitseminen perustuu siihen, että näytön rakenteeseen tai sen pintaan on upotettu läpinäkyvää johdinta. Koskettamalla tätä johdinta sähköä johtavalla esineellä, kuten esimerkiksi ihmissormella voi-

daan havaita muutos sähkökentässä ja muutoksesta voidaan laskea kosketuspaikka. Menetelmä on suhteellisen hyvä mobiililaitteiden yleisessä käytössä: Se on responsiivinen, suhteellisen tarkka, sallii useiden kosketuksen samanaikaisen tunnistuksen, kestävä ja sietää suhteellisen hyvin pinnan epäpuhtauksia. Menetelmän vakavin ongelma käytännön tilanteissa on sen perustuminen sähkönjohtavuuteen ja siten kyvyttömyys havaita sähköä johtamattomien esineiden kosketusta, kuten laitteen käyttöä hansikkaat kädessä. [1, s. 35-]

Walker esittelee kapasitiivisesta periaatteesta useita variaatioita, mutta ne eivät ole merkityksellisiä näytön manipuloinnin kannalta. Näiden variaatioiden lisäksi ainoastaan resistiivinen menetelmä on enemmän käytetty mobiililaitteissa.

Resistiivisen kosketusnäytön näyttöön rakennetaan kaksi kerrosta toisistaan erotettuja sähköä johtavia kalvoja ja toiseen kytketään jännite. Kosketuksessa nämä kalvot painautuvat toisiaan vasten painokohdassa joka voidaan havaita ja kosketuspaikka laskea. Menetelmän suurimmat edut ovat hinta ja toiminta sähköä johtamattomilla esineillä.[1, s. 52-]

Muut menetelmät perustuvat kosketuksen havaitsemiseen esimerkiksi ultraäänen tai infrapunan avulla. Ne ovat kuitenkin harvinaisempia mobiililaitteissa ja siten ne kannattaa ottaa huomioon vain jos tiedetään, että järjestelmällä on tarkoitus testata niitä hyödyntäviä laitteita.

3.3.2 Kosketusnäytön manipulointi

Kosketusnäytön manipulointiin kykenevän työkalun valintaan on kaksi vaihtoehtoa: Yksinkertaisinta on käyttää osoitinkynää, joka on siihen tarkoitettu ja siten tiedetään toimivaksi. Toinen vaihtoehto on rakentaa työkalu itse. Edellisen luvun perusteella havaitaan, että yleisimmin käytössä olevissa eli kapasitiivisissa näytöissä robotin työkalun on johdettava sähköä, mutta sen ei ole tarpeen painaa pintaa. Resistiivisissä näytöissä työkalun on oltava riittävän jäykkä että sillä voi painaa näyttöä.

Työkalun vaatimukset ovat siis jäykkyys sekä sähkönjohtuvuus. Nämä vaatimukset on helppo täyttää esimerkiksi käyttämällä metallista keppiä. Metallikepillä voitaisiin itse asiassa manipuloida myös useimpien muiden Walkerin listaamien tekniikoiden [1] näyttöjä.

Mobiililaitteiden käyttöliittymät hyödyntävät yksinkertaisen kosketuksen lisäksi erilaisia kosketustapoja, joita kutsutaan eleiksi. Useimmat eleistä eivät aiheuta ylimääräisiä vaatimuksia kosketuksen lisäksi, koska esimerkiksi raahaus eli sormen asetus näytön pinnalle ja liikuttaminen johonkin suuntaan vaatii vain hieman erilaisen liikeradan robotille. On kuitenkin joitain eleitä, jotka vaativat useamman sormen käyttöä samanaikaisesti. Näiden eleiden toistamiseen vaaditaan jokin ratkaisu.

Periaatteessa ongelman täydellinen ratkaisu vaatii useamman robotin, mutta se on kallis ratkaisu ja vaatisi monimutkaisempaa ohjauslogiikkaa robottien synkronointiin. Lisäksi se

rajoittaisi myös robotin valinnan sellaisiin robotteihin jotka pystyvät toimimaan samalla työalueella.

Yleisimmät eleet on kuitenkin tarkoitettu tehtäväksi yhdellä kädella ja siten kosketuspisteiden ei tarvitse olla kovin kaukana toisistaan tai liikkua toisiinsa nähden monimutkaisesti. Tyyppiesimerkkinä kahden sormen siirtäminen toisiaan päin näyttöä koskettaen, ei kahden näytön kahden vastakkaisen reunan samanaikainen koskettaminen.

Tämä tarkoittaa sitä, että eleet voidaan toteuttaa käyttämällä yhtä robottia jonka työkalussa on useampi sormi siten, että sormia voidaan liikuttaa toisiinsa nähden. Tämä on merkittävästi halvempaa ja yksinkertaisempaa ja siten paras ratkaisu, jollei käyttötapaus vaadi eleitä johon se ei kykene.

3.3.3 Robotti

Edellisessä luvussa määritelty työkalu on pystyttävä liikuttamaan haluttuun kohtaan testattavan laitteen näyttöä. Tämä onnistuu parhaiten hankkimalla siihen sopiva robotti.

Kosketusnäyttötestaukseen on tarjolla juuri siihen tarkoitukseen rakennettuja robotteja, kuten aiemmin mainittu Taktouch-1000 [19], mutta käyttötapaus ei ole kovin vaativa, joten esimerkiksi pienet Delta-robotit ovat myös hyvä ratkaisu.

Erityisen kiinnostava vaihtoehto on kuitenkin Tapsterbot [20]. Tapsterbot on mobiililaitteen testaukseen tarkoitettu, avoimen lähdekoodin Delta-robotti. Tarjolla on siis sekä sen 3D-tulostamiseen tarvittavat tiedostot, että sen ohjauslogiikka. Se on myös mahdollista tilata valmiiksi koottuna suhteellisen halvalla.

4 PROTOTYYPIN TOTEUTUS

Tämän DI-työn tavoitteena on tutkia mobiililaitteen testausta ilman ohjelmallista yhteyttä käyttäen edullisia komponentteja. Kuten luvussa 2.3 esiteltiin, tätä tutkittiin rakentamalla ratkaisusta prototyyppi. Prototyypin vaatimukset esiteltiin yleisellä tasolla samassa luvussa.

Tässä luvussa esitellään luotu prototyyppi. Mobiililaitteen testausrobottiin vaadittavia toiminnallisuuksia ja sen toteutukseen tarjolla olevia työkaluja esiteltiin luvussa 3.

Laitteisto toteutettiin työnantajani Wapice Oy:n osaamisenkehittämisprojektissa silloin kun tiimiläisillä oli mahdollisuus käyttää aikaa varsinaisten työprojektiensa ulkopuolella. Toteutus kärsi siten resurssipulasta, tauoista ja pienestä budjetista. Tämä näkyy siinä, että toteutuksessa poikettiin joissain kohdissa edellisessä luvussa esitellyistä vaihtoehdoista ja joitain osia jätettiin toteuttamatta tai toteutettiin tavalla joka ei olisi riittävä tuotantokäytössä. Suurin osa näistä puutteista ei kuitenkaan ole tässä yhteydessä merkittäviä, koska työn tavoitteena ei ollut tehdä täysin valmista, tuotantokäyttöön sopivaa järjestelmää, vaan toteuttaa järjestelmä riittävän pitkälle, jotta sen toteutuskelpoisuutta voidaan arvioida alustavasti. Puutteet mainitaan niistä moduuleista kertovissa luvuissa.

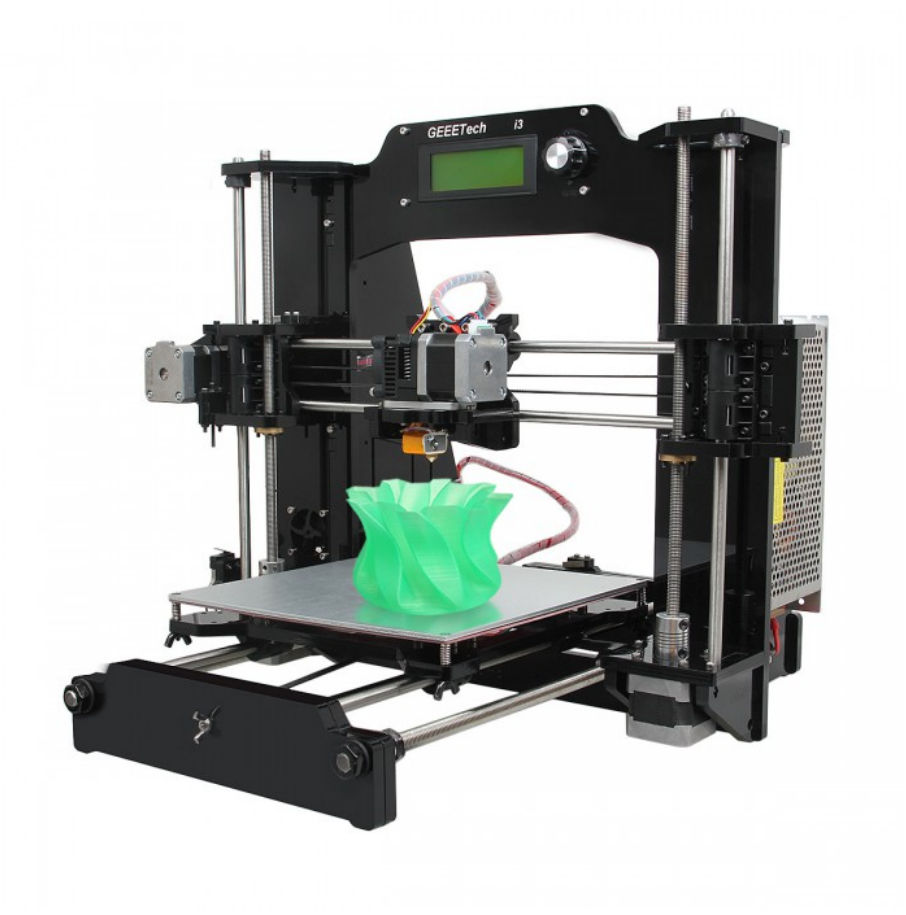
4.1 Laitteisto

Protoyypin laitteisto koostuu:

- Ohjelmiston ajoon käytetystä tietokoneesta
- Testattavan laitteen manipulointiin käytetystä robotista
- Testattavan laitteen tilan havainnointiin käytetystä kamerasta

Ohjelmiston ajoon harkittiin aluksi erillisen laitteen, esimerkiksi Raspberry Pi:n ostamista, mutta ainakin näin prototyyppikäytössä katsottiin, ettei erillisestä laitteesta ole riittävästi hyötyä ottaen huomioon budjettirajoitteet sekä se, että siitä saattaisi aiheutua suorituskykyongelmia. Ohjausohjelmiston ajoon päätettiin siis käyttää ohjelmiston kehitykseen käytettyjä tietokoneita.

Laitteiston muut osat esitellään seuraavissa aliluvuissa.



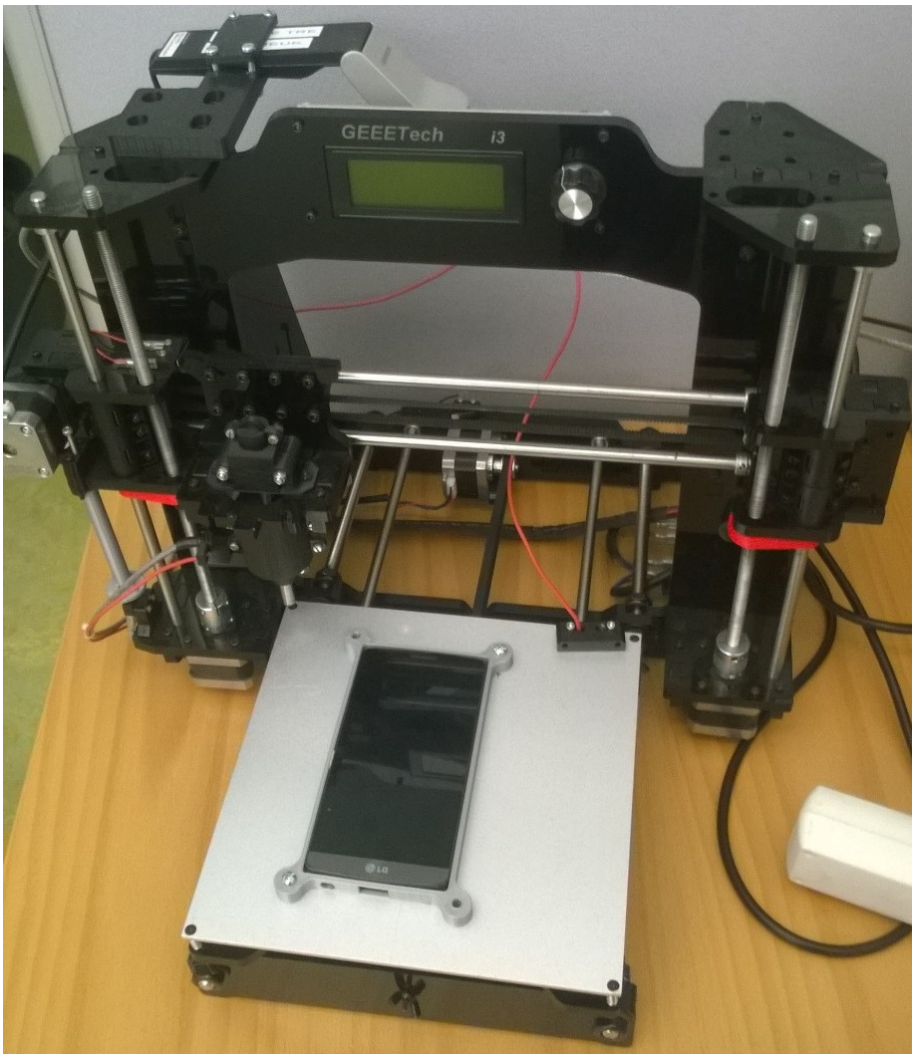
Kuva 4.1. *Geeetech i3 Reprap Prusa [6]*

4.1.1 Robotti

Testattavan laitteen manipulointiin tarvittavaksi robotiksi harkittiin kolmea päävaihtoehtoa: Teollista robottia, itse tehtyä robottia ja 3D-tulostinta.

Teollinen robotti, esimerkiksi karteesinen- tai Delta-robotti arvioitiin todennäköisesti paremman suorituskyvyn omaaviksi kuin vaihtoehdot, mutta halvimmat löydetyt robotit olivat kaikki yli 1000 euroa, joka olisi vaikea saada mahtumaan tiimin budjettin. Epäilimme myös halvimpien vaihtoehtojen mahdollisia tarkkuus tai luotettavuusongelmia. Lisäksi koska kenelläkään tiimistä ei ollut kokemusta robotin ohjauksesta, ohjauslogiikan toteutuksen vaativa työmäärä olisi ollut ylimääräinen riskitekijä.

Itse tehty karteesinen robotti olisi tietenkin ehdottomasti halvin tarvittavien komponenttien hinnan puolesta ja koska kyseessä oli prototyyppi, mahdolliset hitaus tai luotettavuusongelmat eivät olisi merkittäviä. Prototyypin tarpeisiin valittavan robotin olisi kuitenkin oltava suhteellisen tarkka ja tarkkuusongelmat olisivat todennäköisiä. Ongelmat eivät välttämättä olisi ylitsepääsemättömiä erityisesti jos kamera asetettaisiin robottiin sormen viereen, koska silloin voitaisiin ainakin lieventää mahdollisia tarkkuusongelmia tekemällä ohjaus takaisinkytkevästi. Ratkaisu olisi myös ollut projektin päätavoitteesta eli osaamisen kehittämisnäkökulmasta paras. Ratkaisussa oli kuitenkin se suuri ongelma, että toteutuksen

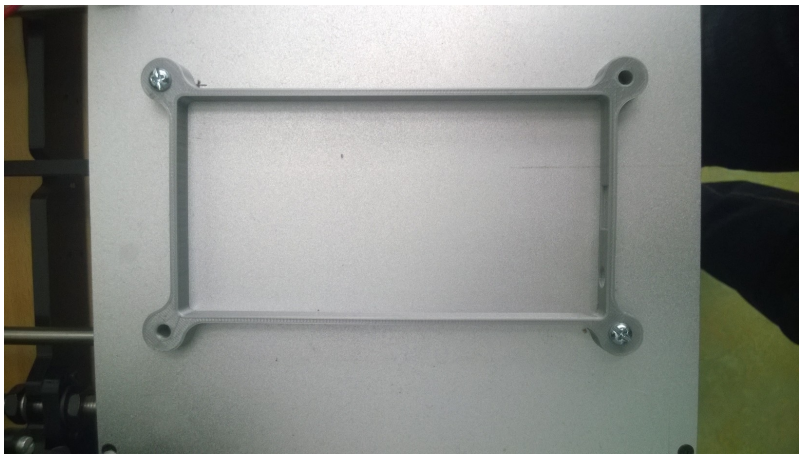


Kuva 4.2. *Tulostin muutosten jälkeen*

työmäärä arvioitiin tiimin resursseihin nähden suureksi, erityisesti huomioon ottaen sen sisältämät riskit.

Lopulta tiimi päätyi muokkaamaan tarkoitukseen 3D-tulostimen. 3D-tulostimen saa muutamalla sadalla eurolla eli se toteutti budjettitavoitteen. Lisäksi niiden ohjauksen tarkkuus tiedetään tähän tarkoitukseen riittäväksi, koska tulostaminen vaatii suurempaa tarkkuutta. Mahdollinen laitteen hitaus ei ole merkittävä ongelma prototyyppiä tehtäessä. Konversio olisi myös todennäköisesti helppo. Lisäksi yhdellä tiimiläisellä oli kokemusta 3D-tulostimista ja siten pystyimme olemaan suhteellisen varmoja ettei ohjaus tuottaisi ongelmia.

3D-tulostimeksi valikoitui pääasiassa hintansa takia Geeetech i3 Reprap Prusa [6]. Kokoamisen jälkeen sen tulostinpää korvattiin kosketusnäytön manipulointiin sopivalla sormella, kamera kiinnitettiin siihen, sen alustaan kiinnitettiin kehikko testattavan laitteen kiinnitystä varten ja sen turvakytkimiin tehtiin hieman muutoksia, jotta ne saatiin toimimaan oikein sormen kanssa. Näistä muutoksista enemmän seuraavissa kappaleissa. Valmistajan kuva käytetystä tulostimesta kuvassa 4.1.



Kuva 4.3. *Testattavan laitteen kiinnitys*

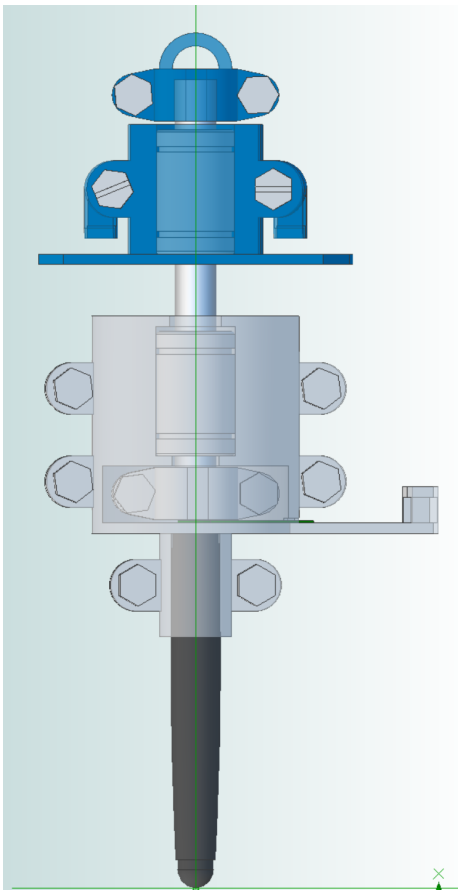
Alkuperäiseen suunnitelmaan kuului tulostimen firmwareen tehtävä paineenmittaus toiminnallisuus. Muutosten tekemistä hidastivat ongelmat lähdekoodin saatavuudessa: Koodi oli kyllä vapaasti saatavilla, mutta varsinaisesti käytössä olevan koodiversion ja asetusten löytäminen oli haastavaa. Koodiversio-ongelmien ja luvussa 4.1.2 esiteltyjen sormen rakenteen ongelmien takia kehitystä jatkettiin pitkään ilman mahdollisuutta havaita kosketuksen voimakkuutta asettamalla näytön korkeus ohjelmaan kalibroinnin yhteydessä ja tämä ratkaisu havaittiin ainakin näin testilaitteessa riittäväksi.

Turvatoiminto näytön liian voimakkaan painamisen estämiseksi katsottiin samoin tarpeettomaksi sormen kiinnityksen muokkauksen takia: Tulostimen automaattisen kalibroinnin takia jouduttiin sen korkeussuuntaista turvakytkintä muokkaamaan siten, että kalibrointi ei yritä ajaa sormea pöydän läpi. Tämä yhdistettynä viimeiseksi jääneen version sormen kiinnityksen joustavuus käytännössä estää robottia painamasta testattavan laitteen näyttöä liian kovaa. Pidimme myös todennäköisenä, että sormi yksinkertaisesti irtoaisi kiinnikkeestään jos sitä painetaan (esimerkiksi sivusuunnasta) testattavaa laitetta päin, joten testipuhelimen omistaja katsoi turvallisuuden riittäväksi valvotuissa olosuhteissa.

Kosketuksen voimakkuuden havainnointia testattiin kiinnittämällä paineanturi lämpötila anturiksi, joten sitä pystyttiin lukemaan lämpötilan luku käskyillä. Tätä toiminnallisuutta ei kuitenkaan lopulta käytetä luvussa 4.1.2 esiteltyjen ongelmien takia. Kuvassa 4.2 on tulostin muutosten jälkeen.

Koska valittu toteutus ei hyödynnä takaisinkytkevää ohjausta, on kameran ja printterin koordinaatistot kalibroitava. Olisi periaatteessa mahdollista tehdä kalibrointi alustaan, mutta jatkokehityksessä oli tarkoitus kalibroida suoraan testattavan laitteen näyttöön ja siten laite oli tarpeen kiinnittää alustaan. Kiinnitys on tarpeen myös esimerkiksi raahauseleitä tehdessä tai testattavan laitteen fyysisiä nappeja painaessa.

Kiinnitykseen harkittiin joustavaa tai säädettävää ratkaisua, johon pystyttäisiin kiinnittämään useita mobiililaitetyyppejä, mutta se jätettiin toteuttamatta, koska kehityksessä käytettyjä testattavia laitteita oli vain yksi. Kiinnitykseen 3D-tulostettiin yksinkertainen kehys joka kiinnitettiin printterin alustaan kuvan 4.3 mukaisesti.



Kuva 4.4. *Työkalun kiinnityksen kaaviokuva*

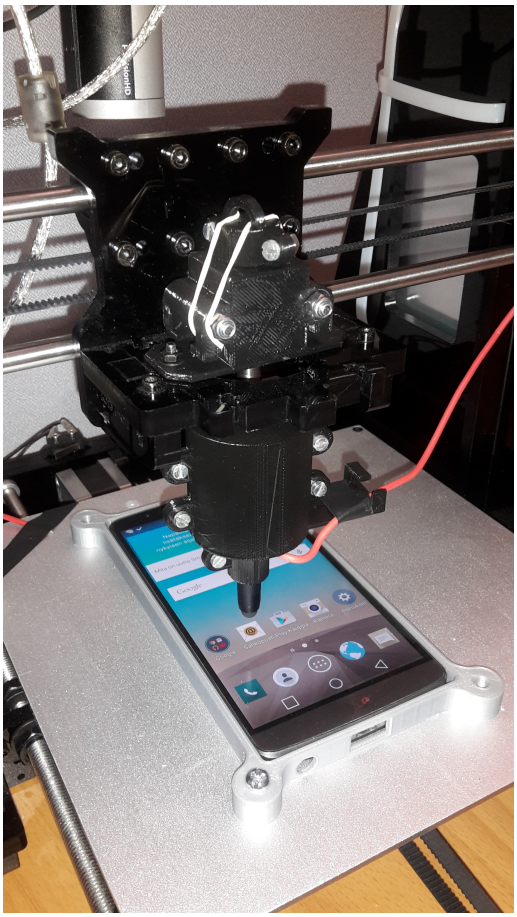
4.1.2 Näytön manipulointityökalu

Tarve toimivan työkalun tekemiselle yritettiin aluksi välttää hankkimalla osoitinkynä ja kiinnittämällä se tulostimen tulostinpään tilalle. Tätä varten suunniteltiin ja 3D-tulostettiin kiinnike.

Kiinnike vaati useamman version, koska sen piti kiinnittää sormi siten, että kiinnikkeeseen piti pystyä lisäämään paineanturi, jolla pystytään havaitsemaan kuinka suurella voimalla työkalu painaa mitattavan laitteen näyttöä. Tämä oli vaikea saavuttaa luotettavasti, koska valittu anturi ei saanut olla pysyvästi paineessa, joten sitä ei voitu yksinkertaisesti laittaa jousen päähän. Lopullinen versio on esitetty kuvassa 4.4.

Paineen mittauksen lisäksi ongelmia aiheutti se, että kehityksessä testauslaitteena käytetyn Android-puhelimen käyttöliittymä tulkitsi jo suhteellisen nopean kosketuksen raahauseleen aloitukseksi. Tulostimen suhteellisen hidas korkeussuuntainen liikenopeus yhdistettynä osoitinkynän joustoon aiheutti hyvin tarkan kalibrointitarpeen, jopa niin tarkan että hieman epätasaisesti alustalla olevan puhelimen ruudun eri kohdissa oltaisiin vaadittu eri kosketuskorkeus.

Tästä syystä vaihdettiin käyttämään yksinkertaista metallikeppiä, jolloin kosketuksen aiheuttama paine työkaluun pystyttäisiin havaitsemaan suoraan. Ohjaamalla tulostinta pai-



Kuva 4.5. *Valokuva työkalun kiinnityksestä.*

neanturitiedon mukaan saatiin toimimaan tällä yhdistelmällä, mutta se aiheutti ylimääräistä viivettä ja paineanturin kiinnitys oli myös hyvin herkkä irtoamaan oikeasta paikasta ja siten epäluotettava.

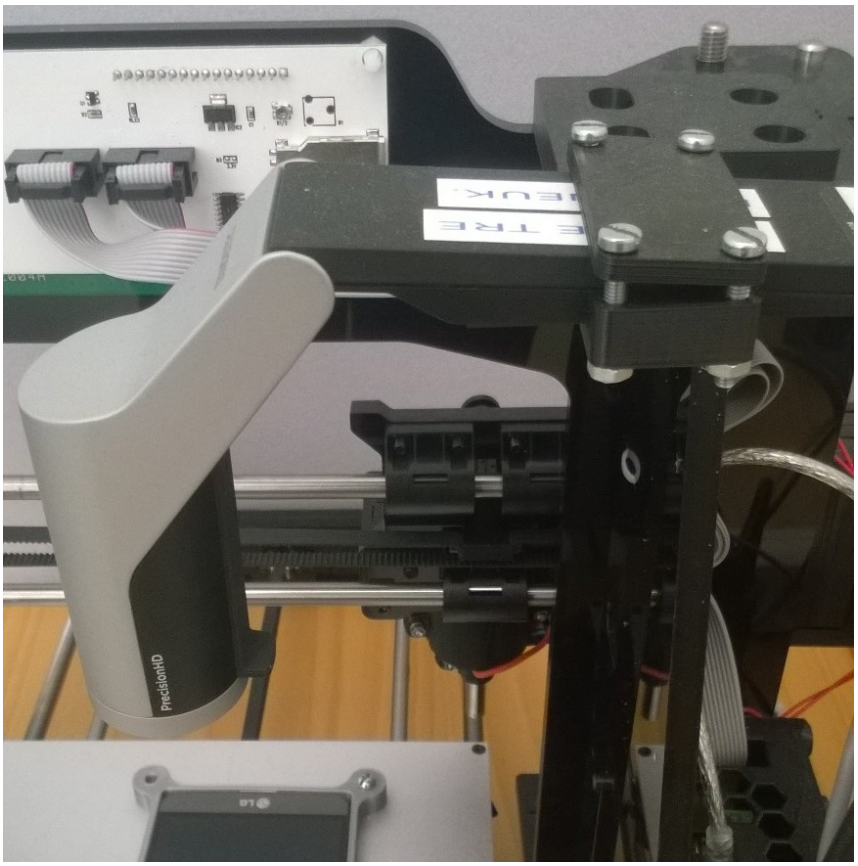
Viimeiseksi jäänyt versio käytti metallikeppiä ilman anturia ja yksinkertaisesti asettamalla kosketuskorkeus kalibrointivaiheessa kosketus saatiin suhteellisen luotettavaksi. Keppiä olisi tosin pitänyt jonkin verran pyöristää - tasaisella päällä yleensä vain yksi sen reunoista koskee näyttöön, joka aiheuttaa ongelmia kosketuksen rekisteröinnissä ja kosketuksen paikassa. Ratkaisusta valokuva kuvassa 4.5.

4.1.3 Kamera

Laitteistoon tarvittavalle kameralle ei ollut erityisiä vaatimuksia kuvankäsittelyn suunnittelun aikana ja tiimistä ei löytynyt osaamista niiden määrittelyyn. Siten kameran valinnan kriteereiksi tulivat riittävä tarkkuus ja halpa hinta. Kameraksi valikoitui Logitech C170.

Valinta osoittautui virheeksi heti ensimmäisissä testeissä: Sen tarkkuus ei ollut läheskään riittävä erityisesti pienempien kohteiden tunnistamiseen.

Koska uuden kameran ostamiseen ei haluttu käyttää rahaa eikä aikaa, päätettin vaihtaa



Kuva 4.6. *Kameran kiinnitys*

kamera ensimmäisissä testeissä käytettyyn ja toimivaksi havaittuun Tandberg Precision HD konferenssipuhelukameraan. Kamera oli hieman haastava kiinnittää, koska sitä ei haluttu vaurioittaa esimerkiksi poraamalla siihen kiinnitysreikiä, mutta ratkaisu oli muuten toimiva: Kamera oli riittävän tarkka ja käytännössä ilmainen.

Myös tämä valinta havaittiin myöhemmin ongelmalliseksi: Kameraa ei oltu tarkoitettu tähän tarkoitukseen ja siten sen tarkennus ja valotuksen säätö olivat automaattisia. Tarkennuksessa ei varsinaisesti ollut ongelmia, joita ei olisi todennäköisesti ollut myös paremmin käyttöön sopivassa kamerassa, mutta valotuksen vaihtelun havaittiin aiheuttavan selkeää epäluotettavuutta testeissä. Näistä ongelmista lisää luvussa 4.2.3.

Kamera kannattaa sijoittaa tällaisessa järjestelmässä sormen viereen siten, että sormi on juuri ja juuri kameran näkökentässä, koska siinä paikassa kameran kuvasta pystytään havainnoimaan laitteen tilan lisäksi myös sormen sijainti laitteeseen verrattuna. Kuten kuvasta 4.6 näkyy, järjestelmän kamera sijoitettiin kuitenkin korkealle tulostimen liikkuvien osien yläpuolelle.

Pääasiallinen syy sijoitukseen on myös havaittavissa kuvasta: Koska jouduimme käyttämään konferenssikameraa, sen sijoitus sormen viereen olisi käytännössä vaatinut sen purkamista tai vähintään kookasta rakennelmaa. Lisäksi kameran nostaminen riittävän korkealle kuvaamaan koko testattavan mobiililaitteen näyttö kerralla ei todennäköisesti olisi onnistunut tai olisi vähintään ollut todella hidasta johtuen tulostimen hitaasta korkeus-

liikkeestä. Tämäkin sijoitus vaatii tulostimen siirtoa tiettyyn asentoon kuvan ottoa varten, mutta siirto on sivusuuntaista, joka on käytetyllä tulostimella huomattavasti nopeampaa.

Tällä sijoituksella menetetään mahdollisuus havainnoida sormen paikkaa kamerasta. Sormen paikkatiedon menetys tarkoittaa tarvetta synkronoida järjestelmä. Lisäksi menetetään mahdollisuus takaisinkytkevän ohjauksen käyttöön. Takaisinkytkevässä ohjauksessa ohjauskomennoissa voidaan hakea haluttua kohdetta kuvassa ja sitten havainnoida mihin suuntaan sormen on liikuttava jotta se osuu haluttuun kohteeseen ja liikuttaa sormea iteratiivisesti oikeaan suuntaan. Tällä iteratiivisella metodilla saavutettaisiin hyvä virhesietoisuus ja mahdollisuus olla sitomatta testattavan laitteen asemaa tiukasti, koska ohjaus tapahtuisi suhteellisen, ei absoluuttisen aseman pohjalta.

Näin tehty ohjaus vaatii kuitenkin kuvantunnistukselta kykyä tunnistaa etsittävä kohde huolimatta etäisyydestä ja kulmasta. Se on selkeästi vaikeampi ja työläämpi ongelma ratkaistavaksi kuin testattavan laitteen aseman kalibrointi. Pidettiin myös hyvin todennäköisenä, että menetelmän virheet olisivat suurempaa kokoluokkaa kuin tulostimen liikkeiden virheet.

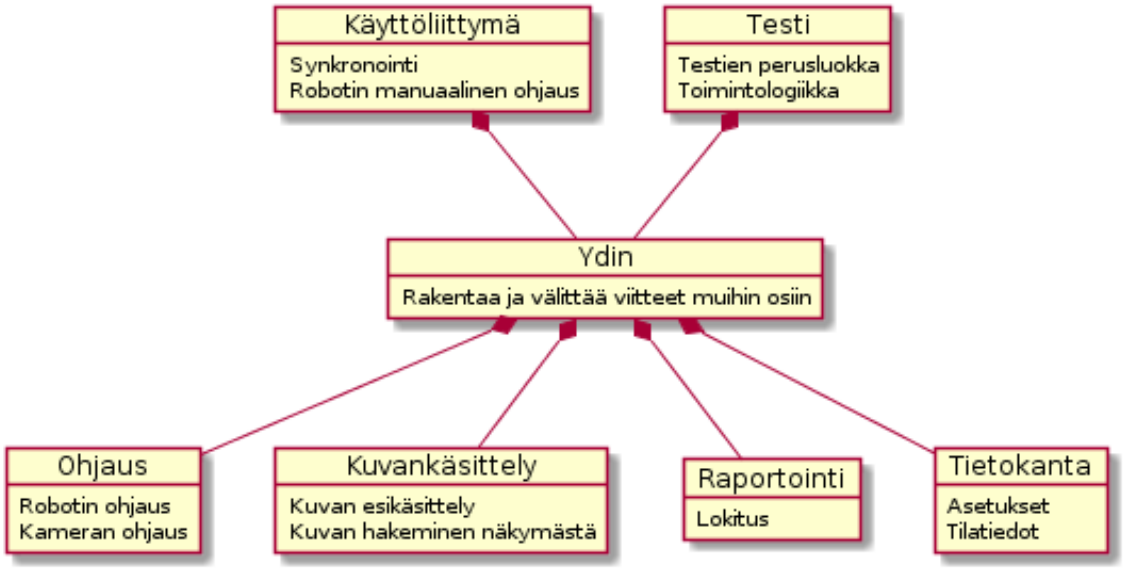
4.2 Ohjelmisto

Prototyypin ohjelmistolle havaitaan aiempien lukujen (erityisesti luvun 3) pohjalta kolme yleisen tason vaatimusta:

- Ohjelmiston on kyettävä tulkitsemaan testattavan laitteen tilaa laitteistoon kuuluvan kameran kuvasta.
- Ohjelmiston on kyettävä ohjaamaan laitteistoon kuuluvaa robottia vaikuttaakseen testattavan laitteen tilaan.
- Ohjelmiston on sisällettävä testauslogiikka edellisen kahden vaatimuksen käyttöön ja työkalut testauslogiikan määrittämiseen.
- Ohjelmiston on tarjottava käyttöliittymä sen käyttöön.

Prototyypin tapauksessa käyttöliittymä ei ole oleellinen ja voi siten olla suppea, mutta on käytännön pakko tarjota joitain perustoiminnallisuuksia, jotta prototyyppiä pystyy testaamaan. Käytännössä tämä on hyvin tiiviisti kytköksissä testauslogiikan toteutukseen, joten ohjelman toteutettaviksi osiksi saadaan:

- Kuvantunnistus
- Laiteohjaus
- Testauslogiikka



Kuva 4.7. *Ohjelman rakenne.*

4.2.1 Ohjelman rakenne

Kappaleen 3.1 suositusten vastaisesti prototyypissä päädyttiin jättämään käyttämättä testikehyksiä. Tämä johtuu siitä, että kappale 3.1 käsittelee tilannetta, jossa järjestelmä rakennettaisiin oikeaan käyttöön ja siten sille tehtäisiin suuri määrä testejä. Täten tehokkaampien työkalujen säästämä aika testien luonnissa kertautuisi lukuisia kertoja ja siten työkaluihin kannattaa käyttää aikaa. Tämä ei tietenkään ole totta prototyypin kohdalla, jolle tehtäisiin vain joitain yksittäisiä testejä.

Työpöytäsovelluksen toteutuskieleksi valikoitui Python 3 [17]. Pääsyynä oli se, että käytettävät kirjastot olivat helposti saatavilla vain Pythonille ja C++:lle. Näistä Python sopii korkeamman tason ohjelmointikielenä paremmin testien kirjoituskieleksi, joten se valittiin koko työpöytäsovelluksen toteutuskieleksi.

Alkuperäisessä suunnitelmassa ohjelma koostui ydinkomponentista, joka omisti ja rakensi muut komponentit, välittäen niille tarvittaessa viitteet toisiinsa. Ydinkomponentti oli myös vastuussa ohjelman tilan ylläpidosta ja eri käyttötapojen käynnistyksestä.

Peruskäytössä ohjelman perussekvenssi, eli testin suoritus lähtisi siis joko ytimestä tai graafisesta käyttöliittymästä. Tämä aktivoisi testin ajomoduulin, joka sitten käyttäisi laiteohjausta, kuvantunnistusta ja lokitusta testin suorittamiseen ja raportointiin.

Muussa käytössä käyttösekvenssit lähtisivät pääasiassa käyttöliittymästä, joka välittäisi käskyt ydinkomponentilta saamilleen muille komponenteille. Tässäkin tapauksessa ydinkomponentti rakentaisi kaikki komponentit.

Ohjelman lopulliseksi jäänyt versio on kuitenkin pikemminkin kokoelma osittain integroituja osia, kuin yhtenäinen looginen kokonaisuus.

Ensimmäisenä syynä heikkoon integraatioon on se, että nopeasti sen jälkeen kun ohjel-

man runko oli valmis ja testien ajon logiikkaa alettiin toteuttaa, päädyttiin käyttämään Pythonin yksikkötestikehystä. Näin saatiin valmiina suuri osa testien perustoiminnallisuudesta, kuten testien perusosat, keinot ajaa testejä ja raportoida niiden onnistumisesta sekä automatiikkaa testien hakemiseen sekä ajoon. Jopa näitä tärkeämpi syy valintaan oli kuitenkin sen mukanaan tuomat integraatiot: Yksikkötestauskirjastoon on saatavilla erilaisia valmiita lisäosia ja ennen kaikkea sille löytyy suora tuki useista kehitysympäristöistä.

Valinta ei kuitenkaan ollut yhteensopiva alkuperäisen arkkitehtuurin kanssa, koska kirjaston ajomekanismi aloittaa suorittamisen suoraan testitiedostosta ja siten alkuperäisen suunnitelman ydinkomponenttia ei suoraan ajettaisi.

Ongelma ratkaistiin määrittelemällä jokaisen järjestelmän testin olevan peritty testien perusluokasta, joka rakentaa ytimen ja pyytää siltä sitten tarvitsemansa komponentit. Tämä muutos tarkoitti myös sitä, että varsinaista ohjelman sisäistä toimintoa testien suorittamiseen ei tehty vaan ainoa tapa suorittaa testejä on kirjaston tarjoamat integraatiot ja komentorivikäskyt.

Toinen syy heikkoon integraatioon on projektin suuri tekninen velka: Osia ohjelmasta tehtiin jo ennen kuin varsinaisen ohjelman runko oli valmis ja näiden osien integrointi katsottiin aina vähemmän tärkeäksi kuin jonkin uuden toiminnallisuuden toteutus.

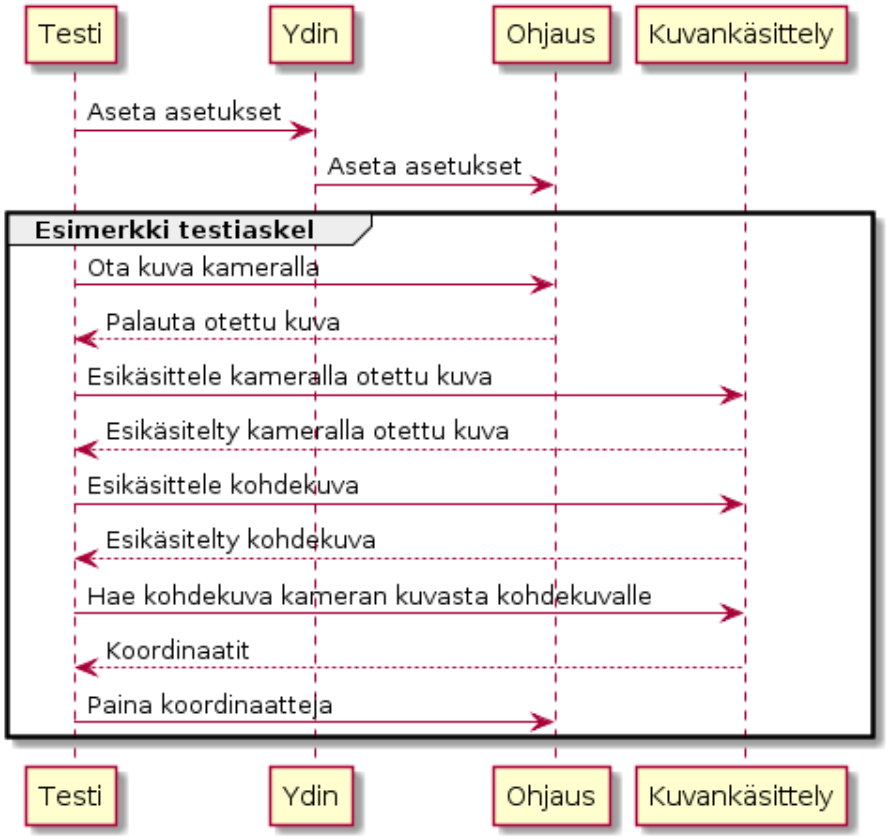
Isoin heikosti integroitu komponentti on käyttöliittymä: Testaukseen rakennettiin nopeasti yksinkertainen käyttöliittymä ennen kuin ohjelman runko oli valmis ja sen sijaan, että oltaisiin tehty arkkitehtuuriin paremmin sopiva käyttöliittymä, kehityksen aikana laajennettiin tätä alun yksinkertaista versiota.

Hahmotelma ohjelman lopullisesta rakenteesta on kuvassa 4.7. Kaavio esittää ohjelman yleistä rakennetta eli se ei suoraan esitä luokka- tai moduulirakennetta, vaan siitä on selkeyden vuoksi jätetty pois vähemmän oleellisia luokkia ja moduuleita ja assosiaatioita luokkien välillä.

Ohjelman pääsekvenssi eli testin suoritus on esitetty sekvenssikaaviona kuvassa 4.8. Sekvenssikaaviossa esitetään toiminta samaan tapaan kuin rakennekuvassakin eli selvyyden vuoksi yleisellä tasolla yksityiskohdat huomiotta jättäen.

4.2.2 Laiteohjaus

Tulostinta käskytetään G-koodi komennoilla sarjaportin kautta. G-koodi on yleisesti työstökoneiden ohjaukseen käytetty ohjelmointikieli. G-koodin komennot ovat kirjain- ja numeroyhdistelmiä, esimerkiksi "G50". Perusidealtaan koodi on vain lista joko erityyppisiä liikekomentoja tai yleensä ohjelman suoritusajan voimassa olevia asetuskomentoja, mutta G-koodiin on tarjolla myös erilaisia suorituksenohjauskomentoja, kuten ehdollista suoritusta tai silmukoita. G-koodin käyttö ja tuetut komennot vaihtelevat kuitenkin laitteiden välillä, sillä siitä on useita standardeja ja valmistajat usein tekevät standardeihin omat muutoksensa. Käytetty tulostin käyttää NIST RS274NGC -standardiin pohjautuvaa ver-



Kuva 4.8. *Testin suoritus*

siota komennoista. [5]

Koska tulostimen ohjelmisto on vastuussa kaikesta varsinaisesta fyysisen laitteen ohjauksesta, pääohjelmassa riittää, että tarjotaan funktiot, jotka lähettävä oikeat ohjaussekvenssit printterille. Näiden funktioiden ei siis ole tarpeen valvoa suoritusta mitenkään vaan riittää, että esimerkiksi näytön painamisfunktio käskyttää tulostinta siirtämään sormen ensin halutun painamiskohdan yläpuolelle, sitten näytön pinnalle ja sitten taas painamiskohdan yläpuolelle.

Varsinainen komentojen välitys suoritetaan käyttämällä Pyserial-kirjastoa, joka tarjoaa mahdollisuuden lähettää sarjaportin läpi tekstikäskyjä ja lukea siihen välitettyjä käskyjä. Tilan luku- ja erilaiset asetuskomennot välitettiin suoraan ja siirtymiskäskyihin käytettiin Mecode-kirjastoa.

Tulostimen ohjauksen on kuitenkin toteutettava kaksi toiminnallisuutta, jotta se kykenee toteuttamaan ohjausmoduulin vaatiman toiminnallisuuden: Ensinnäkin ohjauksen on huolehdittava siitä, että tulostin ei aja sormea laitteen reunoihin, yritä painaa sormea näytön sisään tai raahaa sormea näytön pinnalla muuta kuin raahaus-elettä tehdessään. Tämä toteutettiin yksinkertaisesti lisäämällä "nosto"ja "lasku"siirtokäskyihin ja kalibroimalla käytön aluksi näytön taso ohjauksen käyttämäksi nollapisteeksi.

Ratkaisu on normaaleissa tilanteissa toimiva, mutta ratkaisussa tarvitsisi silti toimivan paineanturin sormeen, koska ratkaisu joutuu luottamaan tulostimen sisäiseen paikkatietoon,

joka oli joitain kertoja virheellinen korkeusakselin liikkumisongelmien takia. Toisaalta akseleiden liikkumisongelmien voidaan katsoa olevan laitteiston virhe, joka olisi korjattava laitteistossa eikä sen tarkastus kuuluisi ohjelmistolle.

Toinen vaatimus on koordinaattikonversiot. Muu ohjelma toimii kuvakoordinaatistossa, koska se käsittelee kameralta saatua kuvaa. Tästä syystä oli hyödyllistä lisätä kameran ohjaus samaan ohjelmistokomponenttiin kuin tulostimen ohjaus, jolloin koordinaattikonversiot pystyttiin eristämään siihen komponenttiin.

4.2.3 Kuvantunnistus

Kuvantunnistuksessa lähdettiin suoraan hakemaan toimivaa ratkaisua tarkoituksena lisätä esikäsittelyä ja kehittyneempiä menetelmiä vain jos yksinkertainen ratkaisu ei olisi riittävä. Kuvantunnistus toteutettiin aluksi käyttämällä feature matching -pohjaista ratkaisua suoraan otettuun kuvaan ilman minkäänlaista esikäsittelyä. Ratkaisu käytti SIFT-algoritmia kulmien hakemiseen ja SURF-algoritmia niiden yhdistämiseen.

Ratkaisu toimi suhteellisen hyvin, joten tiimi keskittyi toteuttamaan muita ohjelman osia. Tämä ei tarkoita, että ratkaisu oli luotettava, mutta koska se toimi 'suhteellisen hyvin', kunnollisen esikäsittelyn tai tunnistuksen parantaminen ei koskaan saanut kehitysaikaa, vaan sitä korjailtiin toteutuksen aikana riittävästi että pystyttiin tarkistamaan sen hetkisen kehityksen kohteena olevan toiminnallisuuden toimintaa.

Tehtyjä parannuksia on kuvien muuttaminen mustavalkoiseksi, eri algoritmien sekä asetusten testaus ja vaihtelu. Feature matching -toteutuksemme tyyppiongelma koko kehityksen ajan oli sen herkkyys löytää suurin osa avainpisteistä oikein, mutta sitten löytää yksi tai kaksi avainpistettä jostain hyvin kaukaa ja siten vääristää oletettu haettavan kuvan paikka näytöllä. Virheitä pystyttiin kompensoimaan käyttämällä etsityn kuvan paikaksi löydettyjen avainpisteiden keskipistettä, eikä prosessin usein voimakkaasti vääristyneeksi olettaman kuvan keskipistettä.

Hieman yllättäen havaittiin template matchingin olevan merkittävästi tarkempi. Se löysi haettavan kuvan lähes virheettömästi, kunhan haettava kuva rajattiin sisältämään mahdollisimman vähän taustaa. Oletettavasti muuten lievät sävyerot aiheuttivat haetun kuvan virheellisen löytymisen tyhjästä pelkkää taustaväriä sisältävästä paikasta, jossa oletettavasti valaistuksen erot tarkoittivat taustavärin olevan lähempänä haettavaa kuvaa.

Template matchingin parempi toimintavarmuus käytännössä esikäsittelemättömän, silminnähtävästi vääristyneen kuvan tapauksessa ei vastaa luvussa 3.2.2 läpikäytyä teoriaa, mutta oletettavasti ero johtuu siitä, että laite on kiinnitetty kiinteästi samassa paikassa olevaan kehikkoon ja kamera ottaa siten kuvan aina samasta asennosta. Tämä yhdistettynä siihen, että kuvantunnistuksessa käytetään laitteistolla otetuista kuvista rajattuja haettavia kuvia, vääristymät ovat samat sekä haetussa kuvassa että kameran ottamassa kohdekuvassa. Tällöin feature matchingin vahvuudet koon ja asennon vaihtelun kans-

sa eivät ole hyödyllisiä ja template matchingin vahvuus myös silminnähden vaihtelevan valaistuksen kanssa tekevät template matchingistä toimivamman ratkaisun.

Lopulliseksi jäänyt versio käyttää tosin kumpaakin tekniikkaa rinnan, koska template matchingille ei onnistuttu löytämään yleisesti pätevää rajaa onnistuneesti löydetylle kuvalle, joten kuvan löytymisen tarkistamiseen käytetään edelleen merkittävästi siinä tarkoituksessa toimintavarmempaa feature matchingiä.

Koska kuvantunnistus oli näin toteutettu nopeasti ja eri vaiheissa työmäärää säästellen, sen tekeminen suunnitelman mukaan olisi vaatinut suhteellisen suurta työmäärää ei vain kuvantunnistuksessa vaan myös muissa ohjelmiston osissa. Koska muutoksia ei oltaisi myöskään pystytty tekemään yhteensopivaksi aiemman toteutuksen kanssa, muutoksen tekeminen olisi vaatinut laitteiston muun kehityksen lopettamista muutoksen ajaksi. Tästä syystä muutoksen tekeminen jätettiin lopulta tekemättä. Valitettavasti tämä tarkoittaa sitä, että muiden kuvantunnistusominaisuuksien toteutus ei käytännössä ollut mahdollista, koska ne olisivat vaatineet useita osia esikäsittelystä.

4.2.4 Testauslogiikka ja järjestelmän käyttö

Kuten luvussa 4.2.1 mainittiin, testauslogiikan pohjaksi valittiin Pythonin yksikkötestauskirjasto. Näin pystyttiin hyvin vähällä työllä luomaan yksinkertainen logiikka testien ajamiselle hyödyntäen kirjaston tarjoamia testinajotapoja, sekä sille tarjolla olevaa kehitysympäristötukea. Käytännössä kirjaston testien perusluokalle luotiin funktio parametrinä annetun kuvan hakemiselle sekä painamiselle. Näin pystyttiin luomaan testejä jotka tarkistavat onko esimerkiksi ikoni näkyvissä ja painaa ikonia.

Kirjastosta johtuen yleinen testauskäyttö toimii siis rakenteellisesti pitkälle samalla logiikalla kuin yksikkötestaus. Varsinainen testien luomis- ja suorittamistyötapa eroaa kuitenkin kahdella tavalla normaalista yksikkötestauksesta: Testejä luodessa on tarpeen saada kuvia haettavista käyttöliittymäelementeistä ja testejä ajettaessa on tarpeen pystyä ohjaamaan robottia. Valitusta kuvantunnistuslogiikasta johtuen järjestelmä on myös tarpeen pystyä kalibroimaan.

Tarvittiin siis käyttöliittymä. Käyttöliittymä toteutettiin PyQt:lla [16]. Käyttöliittymä tarjoaa kuvan järjestelmän kamerasta, ohjauspainikkeet, kuvankaappauspainikkeen testien luomiseen sekä kalibrointitoiminnallisuuden.

Kalibroinnissa robotti ajetaan kahteen pisteeseen testausalueella ja sitten samat pisteet merkitään kameran kuvassa. Näin saadaan kaksi pistettä, joiden sijainti tiedetään sekä kameran ottaman kuvan koordinaatistossa, että robotin koordinaatistossa ja näistä pisteistä voidaan laskea muut pisteet.

Varsinaisessa käytössä tämä todettiin suhteellisen toimivaksi, mutta oikeassa käytössä tarve olla fyysisesti testattavan laitteen ja robotin vieressä olisi tietenkin epäkäytännöllistä ja kuvankaappausten ottaminen olisi tarpeen saada nopeammaksi, esimerkiksi Sikulin ta-

paan tarjoamalla mahdollisuus merkitä näkymästä haluttu alue. Ongelman voisi tietenkin myös ratkaista paremmalla kuvankäsittelyllä, jolloin voitaisiin käyttää suoraan esimerkiksi ohjelman itsensä käyttämiä ikonitiedostoja.

Testiraportointi oli alunperin suunniteltu jättää vain yksikkötestauskirjaston ja ajoraporttien varaan. Toteutuksen aikana tuli kuitenkin selväksi, että lokituksen puutteellisuus oli merkittävä ongelma testejä luodessa sekä järjestelmän testauksessa. Erityisesti erilaiset ei-toistuvat virheet, jossa järjestelmä painoi testattavan laitteen väärää kohtaa olivat usein vaikeasti tutkittavia.

Ongelmaa pystyttiin osittain vähentämään hyödyntämällä laajemmin yksikkötestauskirjaston tarjoamia ominaisuuksia sekä yksinkertaisesti kieleen itseensä liittyviä tulostusmahdollisuuksia, mutta lopulta katsottiin tarpeelliseksi luoda yksinkertainen lokitusmahdollisuus.

Tähän oli kaksi vaihtoehtoehtoa: Yksikkötestauskirjaston raportointikyvyn laajentaminen, erityisesti siten että sen tuottamat raportit pystyisivät sisältämään kuvia ja erillisen toteutuksen luominen. Tuotantokäytössä raporttien laajentaminen olisi ehdottomasti oikea ratkaisu, koska virheiden sitominen tiettyyn testiin olisi olennaista.

Yksikkötestikirjaston laajentaminen olisi kuitenkin ollut hieman työlästä integroinneista johtuen. Erillinen lokitus katsottiin merkittävästi nopeammaksi toteuttaa ja koska järjestelmää testattaessa kehityksen aikana joka tapauksessa ajetaan testejä yksi kerrallaan, se katsottiin riittäväksi.

Erillinen toteutus katsottiin siis prototyypin tapauksessa paremmaksi vaihtoehdoksi ja se toteutettin kahdessa osassa: Ensinnäkin järjestelmään lisättiin ajonaikainen lokitus käyttäen Pythonin standardikirjastoihin kuuluvaa Logger-kirjastoa.

Tärkeämpi osa oli kuitenkin lisätä mahdollisuus tallentaa kuvantunnistustoimenpiteitä tehtäessä haettava kuva, näkymä sekä hakutulos. Nämä tallennetut kuvat oli tarkoituksena myös sitoa ajoraportteihin vähintään merkkaamalla tallennetun kuvan nimet, mutta se jäi tekemättä, koska havaittiin pelkän kuvan riittävän kehitystarkoitukseen.

5 TULOKSET JA TULOSTEN ARVIOINTI

Tämän työn tutkimusaihe on mobiililaitteen fyysisen testauksen viabiliteetti halvoilla fyysisillä komponenteilla sekä avoimen lähdekoodin ohjelmistokomponenteilla. Luvussa 2.3 tätä päätettiin tutkia käyttäen konstruktiivista menetelmää. Prosessissa luotiin tutkimuksen pohjaksi fyysinen prototyyppi.

Edellisessä luvussa esiteltiin luotu prototyyppi ja tässä luvussa esitellään prosessin tulokset. Tässä tapauksessa tulokset saadaan testaamalla prototyyppiä sekä analysoimalla sen rakentamista.

Luvussa 2.3 prototyypille jouduttiin tyytymään suhteellisen yleisiin vaatimuksiin eli käytännössä vain vaatimukseen kyetä testaamaan mobiililaitetta. Tässä luvussa arvioidaan siis toteutetun prototyypin tarjoamia työkaluja mobiililaitteen testaukseen, työmäärää ja valittujen ratkaisuiden toimivuutta. Lisäksi järjestelmälle tehdään yksinkertainen luotettavuustesti.

5.1 Toteutuksen arviointi

Prototyyppiin toteutettiin vaadittava ydintoiminnallisuus. Toiminnallisesti sen voidaan siis katsoa täyttävän minimivaatimukset.

Koska projekti lopetettiin ennen kuin useat osat toiminnallisuudesta olivat valmiit, ydintoiminnallisuus jäi kuitenkin hyvin epäluotettavaksi. Tämä johtui suurimmaksi osaksi toteuttamatta jääneestä kameran kuvan esikäsittelystä. Tämä on siksi erityisen harmillista, että näin ei voida myöskään tehdä oletuksia kuvan laadun riittävyydestä valmiiden työkalujen käyttöön. Mahdollisuus valmiin, kuvaan perustuvan testikehyksen käyttöön vähentäisi oikeaan käyttöön luotavan järjestelmän vaatimaa työmäärää ja tarkoittaisi myös merkittävästi parempaa käyttökokemusta testien luontiin sekä ajoon. Tällöin ainoaksi itse toteutettavaksi osaksi jäisi robotin ohjaus ja mahdollisesti kosketettavan pisteen hakeminen ruudulta, jollei testikehys tarjoaisi siihen mahdollisuutta.

On kuitenkin todennäköistä, että esikäsittelyä ei saataisi riittävän tarkaksi ainakaan investoimatta laadukkaampaan kameraan. Tekstin tunnistuksen alustavat, järjestelmästä erikseen tehdyt testit olivat lupaavia, joten paremmalla esikäsittelyllä tunnistusprosessi todennäköisesti saataisiin toimimaan vain integroimalla tehty toteutus järjestelmään.

Rakenteen tunnistus jäisi siis ohjelmiston suurimmaksi epävarmuustekijäksi, koska ra-

kennepohjainen ratkaisu olisi työläs ja koko näytön vertaus vaatisi kuvan esikäsittelyltä lähes virheetöntä tulosta. Käyttäjäkokemus jäisi todennäköisesti heikoksi, mutta sen parantaminen vain yrityksen omaan käyttöön tulevassa järjestelmässä ei olisi kovin iso ongelma ainakaan tälle yksinkertaiselle versiolle.

Fyysisen ratkaisun voidaan katsoa olevan tarkoitukseen riittävä: Järjestelmä pystyy fyysisesti kaikkeen muuhun kuin testattavan laitteen kyljessä tai alla olevien kytkinten käsittelyyn. Testattavan laitteen kyljissä olevien kytkinten käsittely vaatisi vain laitetta kiinni pitävän kehyksen muokkauksen.

Robotiksi muunnettu 3D-tulostin on tarkoitukseen riittävän tarkka ja perustoiminnallisuus toimii suhteellisen hyvin. Tulostin on kuitenkin tähän tarkoitukseen hidas ja hieman epäluotettava. On kuitenkin huomioitava, että näiden ongelmien ratkaisua yritettiin vain todella pintapuolisesti ja ne olisivat todennäköisesti ainakin lievennettävissä suhteellisen pienellä työmäärällä. Delta-robotti, kuten Tapsterbot [20] olisi kuitenkin todennäköisesti parempi valinta.

Näytön manipulointi on myös viimeiseksi jääneessä versiossa hieman epäluotettavaa, mutta senkin luotettavuutta olisi todennäköisesti suhteellisen yksinkertaista parantaa pyöristämällä sormena käytettyä metallikeppiä.

Kamera ei ole tarkoitukseen sopiva, mutta kamera on helppo vaihtaa, joten sillä ei ole merkitystä kokonaisuuden toteutuskelpoisuuden arvioimisessa. Kameran vaihtaminen myös todennäköisesti vähentäisi ohjelmallisen puolen ongelmia.

Järjestelmään toteutettiin haluttu minimitoiminnallisuus, joten se voidaan katsoa toteutuskelpoiseksi. Valitettavasti toteuttamatta jäänyt näytön rakenteen verifiointi tarkoittaa, ettei tätä voida pitää osoituksena viabiliteetistä.

5.2 Luotettavuustestaus

Järjestelmän arviointia varten luotiin järjestelmälle testi Androidin kellosovellukselle. Se tehtiin käyttäen järjestelmälle määriteltyä ydintoiminnallisuutta eli painettavan käyttöliittymäelementin tunnistus, sen painaminen ja komennon onnistumisen tarkistus tarkistamalla toisen käyttöliittymäelementin näkyminen ruudulla. Testissä oli useita, mutta yksittäin suhteellisen helppoja askeleita. Askeleet koskivat siis selkeitä, suhteellisen kookkaita käyttöliittymäelementtejä eikä esimerkiksi paljon toisiaan muistuttavia, hyvin pieniä tai pienen kontrastin elementtejä ollut mukana.

Testissä testattiin yksi toiminnallisuus joka näkymästä. Testauksessa käytetty testi on yksinkertaistettuna ohjelmassa 5.1. Poistetut osat ovat erilaisten virhetilanteiden käsittelyä ja ohjelman rakenteeseen liittyviä komentoja, kuten laitteen palauttamista perustilaan testien välissä. Paina_kuvaa ohjaa robotin painamaan nimettyä kuvaa ja kuva_nakyvissa tarkistaa onko nimetty kuva näkyvissä. Kummankin funktion epäonnistuminen merkitsee testin epäonnistuneeksi.

```
# Ohjalma kayntiin
poista_lukitus()
paina_kuvaa("kellon ikoni.jpg")
kuva_nakyvissa("kellon otsikko.jpg")

# Halytys paalle ja pois
paina_kuvaa("halytyksen aktivoimis ikoni.jpg")
paina_kuvaa("halytyksen poisto ikoni.jpg")

# Ajastin paalle ja pois
paina_kuvaa("ajastin sivun teksti.jpg")
paina_kuvaa("ajastimen kaynnistys ikoni.jpg")
paina_kuvaa("ajastimen keskeytys ikoni.jpg")
kuva_nakyvissa("keskeytetyn ajastimen uudelleenkaynnistysikoni.jpg")

# Sekunttikellon kaynnistys ja pysaytys
paina_kuvaa("sekunttikello sivun teksti.jpg")
paina_kuvaa("sekunttikellon kaynnistys ikoni.jpg")
paina_kuvaa("sekunttikellon pysaytys ikoni.jpg")
kuva_nakyvissa("sekunttikellon kaynnistys ikoni.jpg")
```

Ohjelma 5.1. *Yksinkertaistettu versio luotettavuustestauksessa käytetystä kellon testikoodista pseudokoodina.*

Testi ajettiin 15 kertaa, joista se epäonnistui 6 kertaa. Testin ajoa valvottiin silmämääräisesti ohjelmallisen tarkistuksen lisäksi. Valvontaa ei voi pitää täysin luotettavana, joten on mahdollista, että noin puolentoista tunnin testiajon aikana valvojalta jäi huomaamatta joitain tilanteita joissa tapahtunutta virhettä ei havaittu ohjelmallisesti. Oikea luotettavuus voi siis olla jopa alle saadun 60%.

Tulos oli kehityksen aikaisen kokemuksen pohjalta arvioitua heikompi. Oletettavasti syynä on se, että testissä yleensä epäonnistui vain yksi testiaskel ja joskus jopa siten, ettei sitä välttämättä olisi huomannut laitetta tarkkailematta. Virheet olivat siis usein sellaisia, että niitä ei olisi välttämättä havaittu kehityksen aikana laitetta ajaessa.

Jos ottaa huomioon, että käytetyssä testissä oli yhteensä 24 laitteen tilan havainnointia vaativaa komentoa, järjestelmän askelkohtainen luotettavuus on suhteellisen korkea. Tästä mittauksesta havaitaan, että askelkohtaisen luotettavuuden on oltava todella korkea, jotta järjestelmä on käyttökelpoinen.

Edustavamman kuvan laitteiston luotettavuudesta saisi, jos laadittaisiin monipuolisempi testi, jossa testattaisiin myös sitä, että havaitaanko kaikkien askeleiden epäonnistuminen. Se vaatisi kuitenkin joko erillisen testiohjelman testatulle mobiililaitteelle, tai testin kirjoitusta siten, että testattavan ohjelman tilaan voidaan vaikuttaa testiajojen välissä siten, että testi epäonnistuu. Testiin olisi myös lisättävä vaikeaksi arvioituja käyttöliittymäelementtejä. Perusteellisempien testien hyödyllisyys katsottiin kuitenkin tässä yhteydessä kyseenalaiseksi, koska laitteiston luotettavuus todettiin jo tässä yksinkertaisessa testissä riittämättömäksi.

Tässä testiajossa kaikki havaitut virheet johtuivat kuvantunnistuksen virheistä, mutta kehityksen aikasen kokemuksen pohjalta on myös mainittava, että kosketuksen rekisteröinti sekä printterin toiminta eivät ole kumpikaan virheettömiä, vaikkei näitä ongelmia havaittu varsinaisen mitatun testiajon aikana.

5.3 Projektin arviointia

Lukujen 5.1 ja 5.2 perusteella joudutaan toteamaan, ettei tämän työn tutkimuskysymykseen pystytä vastaamaan luotettavasti toteutetun järjestelmän pohjalta. Laitteisto kykenee sille asetettuun ydinvaatimuksiin, mutta on liian epäluotettava ollakseen käytettävä.

Luotettavuusongelman lisäksi on myös hieman kyseenalaista pystytäänkö toteutetun järjestelmän katsoa olevan edullinen: Järjestelmän laitteisto oli edullinen, mutta työkäyttöön tulevassa projektissa olennaista on projektin vaatima työaika. Projektiin käytetty työmäärä on todennäköisestä epäedustava projektin harrasteluonteisuudesta ja pitkästä aikajänteestä johtuvasta 'mieleenpalautuksesta' ja epätehokkaasta työjaosta johtuen, mutta tähän pisteeseen pääseminen on vienyt tiimiltä useita satoja työtunteja, joka olisi merkittävä kuluerä.

Katsomalla projektia yleisemmällä tasolla voidaan tehdä joitain huomioita: Mobiililaitteen koskemiseen kykenevän robotin rakentaminen on suhteellisen suoraviivaista ja laitteita on saatavilla halvalla. Testauslogiikka oli myös suhteellisen nopea toteuttaa. Valtaosa käytetystä ajasta käytettiin kuvantunnistukseen tai sen integrointiin muuhun järjestelmään.

Kuvantunnistus oli siis tämän projektin selkeästi ongelmallisin ja työläin osuus. Kameralta saadun kuvan esikäsittely ja tunnistus on haastavaa tehdä siten, että se havaitsee pienetkin virheet vääristymistä huolimatta. Sen lisäksi, että ongelman ratkaisuun kuluu työaikaa, tässä on havaittavissa myös se ongelma, että jos tehtävässä onnistutaan, estetään samalla testiä toimimasta usealla laitteella, testattavista laitteista johtuvien erojen takia. Kuvantunnistuksen vaatimukset ovat siis ristiriitaiset: Testauksessa yleisesti olisi hyödyllistä havaita kaikki muutokset laitteen toiminnassa. Eli esimerkiksi jos käyttöliittymäkomponentti on yhden pikselin leveämpi, se olisi hyvä huomata. Tämä on kuitenkin ristiriitainen vaatimus sen kanssa, että automaattitestit haluttaisiin ajaa useille laitetyypeille.

Testien luontiin useille laitteille voi tarjota hyvät työkalut, mutta ehkä parempi ratkaisu olisi harkita mitä yleensä kannattaa testata ja miten. Tässä projektissa järjestelmä tehtiin osaamisenkehittämistarkoituksessa ja siten yritettiin rakentaa 'geneerinen' testaustyökalu, mutta käytännössä saattaisi olla parempi käyttää useita työkaluja ja hyväksyä ettei tällä työkalulla voida havaita pieniä käyttöliittymävirheitä ja käyttää niiden havaitsemiseen jotain muuta työkalua.

Tämä lähestymistapa vähentäisi kuvantunnistuksen tarkkuusvaatimuksia ja siten sen toteuttamisen vaatimaa työmäärää. Näin saataisiin myös monikäyttöisemmät testit ja lisäk-

si esimerkiksi koneoppimiseen perustuvien menetelmien hyödyntäminen olisi yksinkertaisempaa.

6 YHTEENVETO

Tämän DI-työn tavoitteena oli tutkia mobiililaitteen fyysisen testauksen toteutuskelpoisuutta käyttämättä kaupallisia valmiita työkaluja. Tavoitteena oli siis tutkia, onko mahdollista suorittaa fyysistä testausta rajallisin resurssein vaihtoehtona valmiin ratkaisun ostamiselle.

Mobiililaitteen fyysisellä testauksella tarkoitetaan tässä yhteydessä laitteen testausta ilman ohjelmallista yhteyttä. Testattavan laitteen tilaa havainnoidaan lukemalla kameralla sen näyttöä ja siihen vaikutetaan vain laitteen kosketusnäyttöä mekaanisesti manipuloiden. Tämä on vastakohta ohjelmallista yhteyttä hyödyntävään testaukseen, joka on alalla yleistä.

Toteutuskelpoisuutta tutkittiin konstruktiivisella menetelmällä eli määrittelemällä ensin järjestelmä, joka osoittaisi sen toteutuskelpoisuuden. Sitten pyrittiin rakentamaan prototyyppi järjestelmästä riittävän pitkälle, että voidaan osoittaa ratkaisun suoriutuvan järjestelmälle asetetuista vaatimuksista.

Näiden vaatimusten asettamista vaikeutti kuitenkin se, että vastaavia kaupallisia järjestelmiä ei ollut saatavilla vertailtavaksi ja projekti tehtiin osaamisenkehitystarkoituksessa, ei työkäyttöön, joten sitä ei voitu testata varsinaisessa käytössä. Lopulta vaatimuksiksi asetettiin kyky suorittaa joitain yksinkertaisia testejä mobiilisovellukselle. Näin vaatimuksia ei voida mitata, mutta toisin kuin esimerkiksi tarkkuusvaatimukset, ne ovat kuitenkin hyvin kohdennettuja.

Testausrobotin prototyyppi rakennettiin muokkaamalla testattavan laitteen manipulointiin 3D-tulostin ja kiinnittämällä tulostimeen kamera. Näitä ohjaamaan luotiin ohjelmisto hyödyntäen vapaan lähdekoodin kirjastoja.

Prototyypin laitteisto oli yleisellä tasolla suoraviivainen: 3D-tulostin tarjosi valmiin ohjausrajapinnan ja vaati vain pieniä muutoksia, suurimpana tulostinpään vaihto testattavaa laitetta manipuloivana sormena käytetettyyn metallitappiin. Prototyypin manipulointikyvyn voidaan katsoa olevan tarkoitukseen riittävä, mutta käytetty kamera ei todennäköisesti ollut riittävä.

Ohjelmallisella puolella prototyypin suorituskyky jäi puutteelliseksi. Tämä johtuu pääasiassa siitä, että projekti lopetettiin lähes heti kun ydintoiminnallisuus, eli kyky suorittaa yksinkertaisia ’tarkista näkyykö käyttöliittymäelementti 1, paina käyttöliittymäelementtiä 2’-tyylisiä testejä ilman ohjelmallista yhteyttä, saatiin toimimaan.

Varsinaiset puuttuvat osat, kuten tekstin tunnistus, eivät kuitenkaan ole merkityksellisiä tässä yhteydessä, koska testaus on mahdollista pelkällä ydintoiminnallisuudella. Prototyypin toteutuksessa on kuitenkin se vakava ongelma, että järjestelmän luotettavuus on täysin riittämätön oikeaan testaukseen. Luotettavuutta voitaisiin parantaa jatkamalla esikäsittelyn työstämistä, mutta ei voida olla varmoja, että prototyyppi saataisiin riittävän luotettavaksi.

Prototyypin rakentamisen kustannustehokkuus on myös hieman kyseenalainen: Laitteiston rakentamiskustannukset olivat pienet - vain muutama sata euroa - , mutta kehityksen vaatima työpanos suhteellisen suuri, useita satoja tunteja. Prosessi toteutettiin osaamisenkehittämisprojektina, joten työskentely oli tehotonta taukojen ja niukkojen resurssien takia, mutta myös ammattimaisesti tehdyssä projektissa työmäärä olisi merkittävä.

Tämän DI-työn tulokseksi jääkin siten se, että alustavat tulokset ovat lupaavat, mutta toteutuskelpoisuuden arviointia ei käytännössä voida tehdä tämän prototyypin pohjalta.

LÄHDELUETTELO

[1] A. K. Bhowmik. *Interactive Displays: Natural Human-Interface Technologies*. English. 1. painos. Vol. 9781118631379. GB: Wiley, 2015. ISBN: 9781118631379.

[2] M. Craciunescu, S. Mocanu, C. Dobre ja R. Dobrescu. Robot Based Automated Testing Procedure Dedicated to Mobile Devices. *2018 25th International Conference on Systems, Signals and Image Processing (IWSSIP)*. Kesäkuu 2018, 1–4. DOI: 10.1109/IWSSIP.2018.8439614.

[3] K. Dawson-Howe. *A Practical Introduction to Computer Vision with OpenCV*. Englanti. Boca Raton, Florida, USA: John Wiley ja Sons, Incorporated, 2014.

[4] V. Garousi ja M. V. Mäntylä. A systematic literature review of literature reviews in software testing. English. *Information and Software Technology* 80 (2016), 195–216.

[5] *G-Code*. Englanti. RepRap, verkkosivu. 2019. URL: https://reprap.org/wiki/G-code (viitattu 21. 03. 2019).

[6] *Geetech*. Englanti. Shenzhen Getech Technology Co.,Ltd, verkkosivu. 2019. URL: https://www.geeetech.com/ (viitattu 24. 03. 2019).

[7] *How We Built a Robot for Automated Manual Mobile Testing*. Englanti. TestDevLab, verkkosivu. 2017. URL: https://www.testdevlab.com/blog/2017/07/how-we-built-a-robot-for-automated-manual-mobile-testing/ (viitattu 02. 03. 2019).

[8] *Improving the quality of the output*. Englanti. Github, verkkosivu. 2018. URL: https://github.com/tesseract-ocr/tesseract/wiki/ImproveQuality (viitattu 09. 09. 2018).

[9] A. Kaehler ja G. Bradski. *Learning OpenCV 3 : Computer Vision in C++ with the OpenCV Library*. English. Sebastopol: O'Reilly Media, Incorporated, 2017. ISBN: 1491937998.

[10] P. Kong, L. Li, J. Gao, K. Liu, T. F. Bissyande ja J. Klein. Automated Testing of Android Apps: A Systematic Literature Review. English. *IEEE Transactions on Reliability* (2018), 1–22.

[11] *Konstruktiivinen tutkimusote*. Englanti. Metodix, verkkosivu. 2001. URL: https://metodix.fi/2014/05/19/lukka-konstruktiivinen-tutkimusote/ (viitattu 21. 03. 2019).

[12] S. Krig. *Computer Vision Metrics*. Englanti. Apress, 2014.

[13] K. Mao, M. Harman ja Y. Jia. Robotic Testing of Mobile Apps for Truly Black-Box Automation. English. *IEEE Software* 34.2 (2017), 11–16.

[14] A. Nair. *Overview of Tesseract OCR engine*. Tekninen raportti. Joulukuu 2016.

[15] *Optofidelity*. Englanti. Optofidelity, verkkosivu. 2018. URL: https://www.optofidelity.com/ (viitattu 10. 11. 2018).

[16] *PyQt*. Englanti. Python wiki, verkkosivu. 2019. URL: https://wiki.python.org/moin/PyQt (viitattu 21. 03. 2019).

[17] *Python*. Englanti. Python Software Foundation, verkkosivu. 2019. URL: https://www.python.org/ (viitattu 21. 03. 2019).

[18] *SikuliX by RaiMan*. Englanti. Sikuli, verkkosivu. 2018. URL: http://www.sikulix.com/ (viitattu 03. 03. 2019).

[19] *TakTouch 1000, Robotic Touch Tester*. Englanti. Tactile Automation, verkkosivu. 2018. URL: https://www.tactileautomation.com/products/robotic-touch-tester/taktouch-1000 (viitattu 10. 11. 2018).

[20] *Tapsterbot*. Englanti. Tapster Robotics, verkkosivu. 2015. URL: http://tapster.io (viitattu 11. 11. 2018).

[21] P. Tramontana, D. Amalfitano, N. Amatucci ja A. R. Fasolino. Automated functional testing of mobile applications: a systematic mapping study. English. *Software Quality Journal* (2018), 1–53.