Objektitunnistimen kehitys (Konenäkö)

Valtteri Koivunen

Haaga-Helia ammattikorkeakoulu

Amk-opinnäytetyö

2021

Tradenomin tutkinto

Tiivistelmä

Tekijä(t)
Valtteri Koivunen

Tutkinto
Tradenomi

Raportin/Opinnäytetyön nimi
Objektitunnistimen kehitys (Konenäkö)

Sivu- ja liitesivumäärä
37 + 3

Tässä opinnäytetyössä luodaan objektintunnistin, jonka toteutuksessa sovelletaan laajoja tietojoukkoja sekä koneoppiin perustuvia tekniikoita. Tarkoituksena on aikaansaada objektintunnistin, joka omaa matalan virhemarginaalin sekä kykenee havaitsemaan objekteja yli kymmenen metrin etäisyydeltä. Samalla tutkitaan kuinka koulutusparametrien muokkaaminen ja koulutusalgoritmin valinta vaikuttavat objektintunnistimen tarkkuuteen ja kantamaan.

Opinnäytetyön teoriaosuudessa perehdytään opinnäytetyön aiheeseen yleisesti, alaluvuissa selvitetään mitä konenäkö on. Perehtymisen yhteydessä tutustutaan lyhyesti konenäön historiaan tuomalla esille kuudenkymmenen vuoden aikana tapahtuneet merkitykselliset läpimurrot. Samalla selvitetään konenäön toimintaperiaate tutustumalla konenäön toiminnan kannalta oleellisiin pääkomponentteihin sekä objektintunnistus prosessiin. Edellä mainitun lisäksi teoriaosuudessa käsitellään konenäön ajankohtaisia käyttökohteita. Teoriaosuuden viimeisessä alaluvussa tutustutaan objektintunnistimen koulutukseen soveltuvien koulutusalgoritmien toimintaan.

Opinnäytetyön käytännönosiossa toteutettiin tutkimuksen toteutumisen kannalta oleelliset välivaiheet. Ensimmäisessä alaluvussa toteutettiin tietojoukon muodostukseen ja objektintunnistus luokittelijoiden visualisointiin soveltuvien ohjelmien vaatimusmäärittely. Vaatimusmäärittelyssä suunniteltiin ohjelmien päätoiminnallisuudet. Seuraavassa alaluvussa aloitettiin tietojoukon muodostukseen ja objektintunnistus luokittelijoiden visualisointiin soveltuvien ohjelmien toteutus. Ohjelmien koodauksessa hyödynnettiin python ohjelmointikieltä. Tässä asiayhteydessä pyrittiin implementoimaan mahdollisimman monta vaatimusmäärittelyssä ideoitua toiminnallisuutta. Tämän jälkeen aloitettiin tietojoukon muodostus soveltamalla tietojoukon muodostukseen luotua ohjelmaa. Lopuksi aikaansaatua tietojoukkoa käytettiin objektintunnistus luokittelijoiden koulutuksessa.

Opinnäytetyön lopussa vertaillaan eri koulutusalgoritmeilla ja vaihtelevilla koulutusparametreillä koulutettuja objektintunnistus luokittelijoita keskenään. Samalla analysoidaan kuinka koulutusalgoritminvalinta sekä koulutusparametrien vaihtelevat arvot vaikuttivat aikaansaannosten tarkkuuteen ja kantamaan. Kartutettua tietoa sovellettiin lopullisten objektintunnistus luokittelijoiden koulutuksessa. Tuotoksena syntyi kaksi objektintunnistinta, jotka olivat erittäin tarkkoja ja kykenivät havaitsemaan kohteen yli kymmenen metrin etäisyydeltä.

Asiasanat
Kasvojen tunnistus, Konenäkö, Koneoppiminen, Tekoäly, Algoritmit, Tietojoukko

Sisällys

1 Johdanto

Vuonna 2021 konenäkö markkinoiden arvoksi oli arvioitu 15.9 biljoonaa dollaria, arvioiden mukaan vuoteen 2026 mennessä konenäkö ratkaisuiden markkina-arvo tulee kasvamaan 51.3 biljoonaan dollariin. Primaarinen markkina-arvon kasvuun vaikuttava tekijä tulee olemaan konenäön ratkaisuiden soveltaminen viihde-elektroniikassa. Konenäön parissa tapahtuneet teknologiset innovaatiot ovat tuottaneet huomattavaa lisäarvoa yrityksille sekä tuotteiden loppukäyttäjille. Yrityksille konenäkö ratkaisut tuovat lisäarvoa automatisoimalla prosesseja, jotka perinteisesti vaativat ihmisen työpanosta toimiakseen tehokkaasti. Lisäarvoa tuotetaan ainakin neljällä eri tavalla: parantamalla tuotteiden ja palveluiden laatua, vähentämällä kustannuksia, vähentämällä häiriöitä aiheuttavia tekijöitä, yksinkertaistamalla monimutkaisia prosesseja. Kuluttajalle tarkoitetuissa tuotteissa konenäköä sovelletaan erityisesti älypuhelimissa, pöytäkoneissa sekä kannettavissa tietokoneissa. (Research and Markets 2021; Ajgaonkar 15.03.2021)

Objektintunnistus on konenäön keskeinen toiminnallisuus, joka pyrkii tunnistamaan tai paikantamaan visuaalisessa syötteessä ilmeneviä objekteja. Objektientunnistus voidaan toteuttaa kahdella tavalla, hyödyntämällä koneoppimiseen tai syväoppimiseen perustuvia tekniikoita. Koneoppimiseen perustuvissa tekniikoissa ominaisuuksia poimitaan ihmisen valvonnan alaisuudessa. Syväoppimiseen perustuvissa tekniikoissa ominaisuuksien poiminen toteutetaan ilman ihmisen valvontaa. Objektintunnistus teknologiat toimivat konenäkö painotteisten innovaatioiden pääkomponenttina. (FRITZ LABS INCORPORATED s.a.)

Opinnäytetyön avulla pyrin selvittämään kuinka objektintunnistusalgoritmin valinta sekä parametrien säätäminen vaikuttaa objektintunnistimen tarkkuuteen sekä kantamaan. Algoritmien toiminnasta löytyy runsaasti teoreettista tietoa, mutta hyvin vähän käytännön esimerkkejä. Internetistä löytyy hyvin vähän objektintunnistukseen soveltuvia luokittelijoita, jotka on toteutettu hyödyntämällä eri algoritmi vaihtoehtoja sekä laajoja tietojoukkoja. Lähes kaikki saatavilla olevat luokittelijat on toteutettu hyödyntämällä pieniä tietojoukkoja, jonka seurauksesta luokittelijat ovat usein heikkolaatuisia sekä epätarkkoja. Ilmaisten tietojoukkojen löytäminen on myös hyvin haastavaa, koska laajat tietojoukot ovat maksullisia ja sisällöstä riippuen saattavat vaatia turvallisuusselvityksen. Opiskelijan näkökulmasta helpoin ja halvin tapa päästä käsiksi laajoihin tietojoukkoihin on tietojoukon luominen omatoimisesti.

Opinnäytetyöni toimii keskitettynä tiedonlähteenä kaikille aiheesta kiinnostuneille tahoille, jotka ovat kiinnostuneita oman objektitunnistimen kehittämisestä. Tyypillisesti objektitunnistimen toteuttaminen on pitkäkestoinen "trial and error" prosessi. Toivon mukaan

opinnäytetyöni auttaa muita aiheesta kiinnostuneita välttämään tyypilliset sudenkuopat, sekä säästämään aikaa ja vaivaa.

1.1 Opinnäytetyön tehtävä

Opinnäytetyössäni pyrin aikaansaamaan korkealaatuisen luokittelijan, jonka toteutuksessa hyödynnetään suuria tietojoukkoja. Aikaansaadulla luokittelijalla tulee olla matala virhemarginaali sekä kyky havaita objekteja yli kymmenen metrin etäisyydeltä.

1.2 Opinnäytetyön tavoite ja rajaus

Opinnäytetyölläni on neljä tavoitetta, ensimmäinen tavoite on luoda ohjelman, joka kykenee luomaan suuren tietojoukon. Tietojoukko koostuu valokuvista, jotka luodaan/tallennetaan hyödyntämällä ip-kameraa tai kannettavan tietokoneen web-kameraa. Toinen tavoite on luoda objektintunnistukseen soveltuva luokittelija hyödyntämällä edellä mainitun ohjelman muodostamaa tietojoukkoa. Luokittelijan koulutus toteutetaan hyödyntämällä koneoppiin perustuvia algoritmeja esimerkiksi: HAAR, LBP, HOG. Kolmas tavoite on luoda ohjelma, joka kykenee suorittamaan objektintunnistuksen reaaliaikaisesti hyödyntämällä aikaansaatua luokittelijaa. Neljäs tavoite on tutkia, kuinka koulutusparametrit sekä koulutusalgoritmin valinta vaikuttavat luokittelijan tarkkuuteen ja kantamaan.

Opinnäytetyössä ei luoda erillistä ohjelmaa, joka suorittaa objektintunnistus luokittelijan koulutuksen. Luokittelijan koulutus toteutetaan kolmannen osapuolen ohjelmalla, joka ladataan tietokoneelle. Lisäksi luokittelijan koulutuksessa ei hyödynnetä syväoppiin perustuvia algoritmeja. Syväoppimiseen perustuvien luokittelijoiden luominen on liian pitkäkestoinen prosessi ollakseen toteutettavissa realistisessa aikataulussa.

1.3 Keskeisiä käsitteitä

Konenäkö = Konenäkö tarkoittaa tietokoneella tapahtuvaa kuvatietojen käsittelyä. (Fisher ym. 2014, 162)

Kolmiulotteinen = Kolmiulotteinen tarkoittaa tilaa, joka kuvaillaan kolmella ortogonaalisella vektorilla. (Fisher ym. 2014, 3)

3D-malli = 3D-malli tarkoittaa 3D-objektin muodon kuvausta. (Fisher ym. 2014, 3–4)

Objekti = Objekti tarkoittaa rakennetta, joka koostuu lukuisista ominaisuuksista. (Fisher ym. 2014, 192)

Topologia = Topologia tarkoittaa pistejoukkojen ominaisuuksia, jotka pysyvät muuttumattomina riippumatta tilan uudelleenparametroinnista. (Fisher ym. 2014, 294)

Projektiivinen invariantti = Projektiivisella invariantilla tarkoitetaan ominaisuutta, johon projektiivisella muutoksella ei ole vaikutusta. (Fisher ym. 2014, 222)

Kaavionleikkaus = Kaavionleikkaus tarkoittaa, että pisteiden muodostama osio leikataan kahteen erilliseen joukkoon. (Fisher ym. 2014, 115)

Koneoppiminen = Koneoppiminen tarkoittaa menetelmiä, joita käytetään tietorakenteen automaattisessa analysoinnissa. (Fisher ym. 2014, 162)

Neuroverkko = Neuroverkko tarkoittaa verkostoa, joka koostuu lukuisista verkostoon yhdistetyistä yksiköistä. Neuroverkot suorittavat kuvioiden tunnistustehtäviä. (Fisher ym. 2014, 186)

Stereonäkö = Stereonäkö tarkoittaa ihmisen kykyä määritellä kolmiulotteisia rakenteita. (Fisher ym. 2014, 275)

Kuvaominaisuus = Kuvaominaisuus tarkoittaa kohtausrakenteessa ilmeneviä kuvarakenteita, kuten reunoja ja viivoja. (Fisher ym. 2014, 130)

Stereo = Stereo tarkoittaa ongelmaluokkaa, jossa kuvia hyödyntämällä pyritään aikaansaamaan 3D-ominaisuuksia. (Fisher ym. 2014, 274)

6D-Vision = 6D-Vision tarkoittaa periaatetta, jossa ominaisuuksien 3D-sijainti ja 3D-liike arvioidaan samanaikaisesti. (López ym. 2017, 40)

Syväoppiminen = Syväoppiminen on koneoppimisen muoto, jossa tietokoneet oppivat tietojoukkojen ja neuroverkkojen avulla. (Schramm 2017, 49–50)

Autonominen kulkuneuvo = Autonominen kulkuneuvo on tietokoneen ohjaama mobiilirobotti, jossa ihmisen tekemä päätöksenteko on vähäistä. (Fisher ym. 2014, 23)

2 Tutkimusmenetelmä

Opinnäytetyössä käytetään kvalitatiivisia (laadullisia) sekä kvantitatiivisia (määrällisiä) tutkimusmenetelmiä. Tutkimus toteutetaan luomalla vertailukelpoinen tutkimusaineisto omatoimisesti. Tutkimuksen aineistonhankintamenetelmänä käytetään tutkimuksen aikana toteutettuja kokeita. Tutkimusaineiston keruu toteutetaan porrastetusti kolmessa vaiheessa. Ensimmäisessä vaiheessa luodaan ohjelma, joka kykenee tallentamaan valokuvia automatisoidusti. Toisessa vaiheessa edellä mainittua ohjelmaa hyödynnetään valokuva kokoelman luomisessa. Kolmannessa vaiheessa valokuva kokoelmia hyödynnetään objektintunnistus luokittelijan koulutuksessa. Luokittelijat koulutetaan hyödyntäen eri koulutusalgoritmi vaihtoehtoja sekä vaihtoehtoisia koulutus parametrejä. Kyseiset luokittelijat toimivat tutkimuksen tutkimusaineistona.

Tutkimuksen analyysimenetelmänä käytetään taulukointia. Taulukkoon kirjataan kokeissa käytetyt koulutusalgoritmit, valokuvien lukumäärät, koulutusparametrit, kokeen tila onnistunut/epäonnistunut, koulutuksen kesto sekä kantama. Taulukoinnilla pyritään aikaansaamaan kokonaiskuva luokittelijan tarkkuuteen ja tunnistusetäisyyteen vaikuttavista tekijöistä. Johtopäätösten tekemisessä hyödynnetään taulukkoon kirjattua tietoa sekä visuaalisia havaintoja.

3 Teoriatausta

3.1 Konenäkö

3.1.1 Mikä on konenäkö?

Konenäkö tarkoittaa näköaistin ja päättelykyvyn mekaanista jäljittämistä. Ihmiset hahmottavat ympäröivän maailman rakenteet kolmiulotteisesti. Katsomalla valokuvaa kykenemme laskemaan kuvassa esiintyvien henkilöiden lukumäärän, arvaamaan sukupuolen sekä tunnistamaan henkilön tunnetilan ilmeiden perusteella. Kolmiulotteisen muodon hahmottaminen konenäön avulla perustuu matemaattisiin tekniikoihin eli algoritmeihin. Tarkkojen kohtaus painotteisten 3D-mallien luomisessa, hyödynnetään tuhansia osittain päällekkäisiä valokuvia. Kuvien avulla pyritään rekonstruoimaan objektin ominaisuuksia kuten muotoa tai valaistusta. Szeliskin mukaan konenäkö on haastava tietotekniikan osa-alue, koska aihepiirissä törmätään käänteiseen ongelmaan. Käänteisellä ongelmalla tarkoitetaan täsmällisen ratkaisun aikaansaamista tuntemattomilla sekä puutteellisilla tiedoilla. (Szeliski 2021, luku 1.1)

3.2 Konenäön historia

3.2.1 1970-luku

Viidenkymmenen vuoden aikana konenäkö on kehittynyt nopeaan tahtiin. Konenäön kehitys alkoi 1970-luvun alkupuolella, tutkijoiden tavoitteena oli luoda ohjelma, joka kykenee jäljittelemään ihmisen älykkyyttä. Konenäön kehitystä motivoi halu luoda ohjelma, joka poikkeaa klassisesta kuvaprosessoinnista erityisesti yhden ominaisuuden ansiosta. Valokuvia hyödyntämällä ohjelma kykenisi jäljittelemään maailman kolmiulotteisia rakenteita. Varhaisessa vaiheessa konenäkö päätteli objektin kolmiulotteisen rakenteen kulmien sekä 2D viivojen topologisen rakenteen perusteella. (Szeliski 2021, luku 1.2)

3.2.2 1980-luku

1980-luvulla huomio kiinnitettiin matemaattisiin tekniikoihin. Matemaattisia tekniikoita hyödynnettiin kvantitatiivisessa kuva ja ympäristö analyysissä. Vuosikymmenen aikana toteutettiin lukuisia tutkimuksia, jotka käsittelivät kolmiulotteisten rakenteiden hankkimista, yhdistämistä, mallintamista ja tunnistamista. Tutkimusten avulla pyrittiin selvittämään, kuinka reunoja sekä ääriviivoja voidaan havaita paremmin. Tutkimusten yhteydessä käyttöön otettiin dynaamisesti kehittyvä ääriviivojen seuranta. (Szeliski 2021, luku 1.2)

3.2.3 1990-luku

1990-luvulla panostettiin erityisesti algoritmien kehitykseen. Objekteja pyrittiin tunnistamaan hyödyntämällä projektiivisia invariantteja. Projektiivisilla invarianteilla pyrittiin ratkaisemaan objektin rakenteeseen kohdistuvia liike ongelmia. Vuosikymmenen edetessä edistyksellistä kehitystä tapahtui erityisesti stereoalgoritmien parissa. Vuosikymmenen aikana globaalin optimoinnin ja kaavion leikkaustekniikan yhdistelmää pidettiin merkittävänä läpimurtona. Aktiivisia tutkimuskohteita olivat myös moninäkymäiset stereoalgoritmit, moninäkymäiset stereoalgoritmit, jotka kykenivät tuottamaan kokonaisia 3D-pintoja. Seuranta-algoritmien parissa tapahtui huomattavaa edistystä, tutkijat kykenivät seuraamaan objektin sijaintia aktiivisten ääriviivojen avulla. Varhaisten kasvojentunnistusmenetelmien toiminta perustui pääkomponentin ominaispiirteiden analyysiin, analyysi toteutettiin hyödyntämällä tilastollisia oppimistekniikoita. Ihmisen ja tietokonegrafiikan välillä tapahtuvaa vuorovaikutusta pidettiin vuosikymmenen merkittävimpänä edistyksenä. Vuosikymmenen loppupuolella käyttöön otettiin kuviin perustuvia mallinnustekniikoita, tekniikoita hyödyntämällä kuvakokoelmista pystyttiin luomaan realistisia 3D-malleja. (Szeliski 2021, luku 1.2)

3.2.4 2000-luku

2000-luvulla datapohjaisia oppimistekniikoita pidettiin konenäön ydinkomponentteina. Vuosikymmenen aikana kehitettiin kuvien yhdistämistekniikoita, tekniikoiden avulla valaistuksellisesti poikkeavia valokuvia pystyttiin yhdistelemään keskenään. Objektin tunnistimien kehityksessä hyödynnettiin ominaisuusperusteisia tekniikoita. Ominaisuusperusteisten tekniikoiden avulla onnistuttiin luomaan tunnistimia, jotka kykenivät tunnistamaan ympäristön sekä geologisen sijainnin. Vuosikymmenen puolessavälissä tutkijat panostivat tehokkaampien algoritmien kehitykseen. Edistystä tapahtui erityisesti viestinvälitysalgoritmien parissa, välitysalgoritmeja olivat esimerkiksi LBP-algoritmit. Vuosikymmenen lopussa konenäköön kohdistuvia ongelmia ratkaistiin hyödyntämällä kehittyneitä koneoppimistekniikoita. (Szeliski 2021, luku 1.2)

3.2.5 2010-luku

2010-luvulla koneoppimiseen perustuvia algoritmeja kehitettiin laajojen tietojoukkojen avulla. Koneoppimiseen perustuvia algoritmeja suosittiin erityisesti tunnistusalgoritmien jatko kehityksessä. Kartutettua tietoa sovellettiin koneoppimiseen perustuvien kokonaisratkaisujen kehityksessä. Käytettävissä olevaa laskentatehoa pystyttiin kasvattamaan kehittämällä algoritmeja graafisissa prosessointiyksiköissä. Vuosikymmenen aikana konvoluutiomallista tuli tunnistus- ja segmentointitehtävien edelläkävijä. Konvoluutiomallin menestystä edistäviä tekijöitä olivat: uudistuneet arkkitehtuurimallit, syväoppiminen sekä

neuroverkot. Vuosikymmenen lopussa ihmiskehon 3D-malli pystyttiin päättelemään kuvassa esiintyvien eleiden ja ilmeiden perusteella. (Szeliski 2021, luku 1.2)

3.2.6 2020-luku

2020-luvulla koneoppiin perustuvat algoritmit päihittivät Mooren lakiin perustuvat tekniikat. Mooren laki on teoria, jonka mukaa mikrosirujen prosessointiteho kaksinkertaistuu kahden vuoden välein. Tekoälyyn perustuvat algoritmit noudattivat Mooren lakia vuoteen 2012 saakka. Vuodesta 2012 lähtien algoritmien tehokkuus on kaksinkertaistunut 3.4 kuukauden välein. Vuonna 2017 objektitunnistimen koulutus kesti kolme tuntia, saman tunnistimen koulutus vuonna 2019 kesti 88 sekuntia. Mooren lain heikkeneminen vaikuttaa erityisesti tekoälyä hyödyntävien ratkaisuiden hinnastoon. (Puiu 16.12.2019; Dorrier 17.05.2020)

3.3 Konenäön toiminta

3.3.1 Järjestelmän komponentit

Konenäön toiminta perustuu kahden komponentin väliseen vuorovaikutukseen. Komponentteja ovat: anturi ja tulkkauslaite. Edellä mainittuja komponentteja voidaan verrata ihmisen elimiin. Komponentit jäljittelevät biologisten elinten toimintaa mekaanisesti. Anturi jäljittelee silmän toimintaa ja tulkintalaite jäljittelee aivojen toimintaa. Anturin avulla tietokone kerää visuaalista tietoa ympäristöstä, tyypillisesti anturi on kamera. Tulkkauslaitteen avulla tietokone tulkitsee ja luokittelee anturin tekemiä havaintoja, tulkkauslaite on tyypillisesti konenäkö algoritmi. (Mohamed 2020, luku 1.1.2)

3.3.2 Objektin tunnistus prosessi

Konenäkö käsittelee visuaalista tietoa neljässä vaiheessa, kokonaisuudesta käytetään nimitystä ”computer vision pipeline”. Ensimmäisessä vaiheessa tietokone vastaanottaa visuaalista syötettä kuvantamislaitteesta. Toisessa vaiheessa visuaalinen syöte esikäsitellään. Esikäsittelyllä visuaalisen syötteen väri, koko, asetelma pyritään yhdenmukaistamaan. Kuvien yhdenmukaisuus helpottaa tiedon jatkokäsittelyä. Kolmannessa vaiheessa esikäsiteltyjä kuvia analysoidaan, analyysin yhteydessä kuvista poimitaan tunnistusta helpottavia ominaisuuksia. Poimitut ominaisuudet sisältävät mitattavaa tietoa. Lopputuloksena syntyy ominaisuusvektori, joka mahdollistaa objektin tunnistuksen. Neljännessä vaiheessa ominaisuusvektorit syötetään luokitusmalliin, malli luokittelee kuvassa esiintyvän objektin ennusteiden perusteella. Ennusteiden prosentuaalista tarkkuutta parannetaan neljällä eri keinolla. Ensimmäinen keino on visuaalisen syötteen lisääminen. Toinen keino on esikäsittely prosessin pitkittäminen. Kolmas keino on laadukkaiden ominaisuuksien poimiminen. Neljäs keino on luokittelu algoritmin vaihtaminen. (Mohamed 2020, luku 1.3)

3.4 Konenäön käyttökohteita

3.4.1 Miehitetty kulkuneuvo

Nykyisin konenäköä käytetään kulkuneuvojen järjestelmissä. Konenäköä hyödyntäviä kulkuneuvoja ovat esimerkiksi: henkilöautot, kuorma-autot, lentokoneet ja mars-mönkijät. Miehitetyissä kulkuneuvoissa konenäkö pyrkii parantamaan kuljettajan turvallisuutta tai mukavuutta. Autonomisesti toimivissa kulkuneuvoissa konenäkö pyrkii parantamaan liikkeen sekä toiminnan hallittavuutta. Konenäön avulla voidaan ratkaista liikenneturvallisuuteen kohdistuvia ongelmia, kuten liikenneonnettomuuksia. Liikenneonnettomuuksia voidaan vähentää hallitsemalla liikenteen kulkua kulkuneuvon sisäänrakennetuilla tai ulkoisilla kameroilla. Konenäkö on parantanut nykyaikaisten kulkuneuvojen ajomukavuutta. Ajomukavuutta parantavia tekijöitä ovat esimerkiksi: sokeapisteiden tarkkailu, automatisoitu etäisyyden hallinta, tien epätasaisuuden kompensaatio. Konenäön avulla pyritään aikaansaamaan automatisoituja ratkaisuja, jotka parantavat kulkuneuvojen turvallista navigaatiota monimutkaisessa ajoympäristössä. Turvallisuutta lisää myös konenäön kyky toimia jatkuvasti toisin kuin ihminen, joka tarvitsee riittävän määrän lepoa toimiakseen tehokkaasti. (López, Imiya, Pajdla & Álvarez 2017, 1–4)

3.4.2 Autonominen kulkuneuvo

Autonomisten kulkuneuvojen toiminta pohjautuu konenäköön. Konenäköä käytetään kulkuneuvon sijainnin paikannuksessa, kuva-analyysissä, objektin tunnistuksessa. Kulkuneuvon sijainnin paikannuksessa hyödynnetään kehittyneitä karttoja ja maamerkkejä. Karttojen avulla autonominen kulkuneuvo kykenee laskelmoimaan optimaalisen kulkureitin liikennetilanteen perusteella. Maamerkkejä hyödyntävä sijainninpaikannus perustuu ennalta opittuun tietoon, tiedon avulla konenäkö kykenee määrittelemään kulkuneuvon kolmiulotteisen sijainnin. (López ym. 2017, 33–34)

Autonomisessa kulkuneuvossa kuva-analyysi suoritetaan käsittelemällä stereonäköön perustuvaa tietoa. Tietoa voidaan käsitellä neljässä vaiheessa. Ensimmäisessä vaiheessa arvioidaan visuaalisen tiedon syvyyttä. Toisessa vaiheessa laskennallista taakkaa pyritään vähentämään pakkaamalla visuaalinen tieto kompaktiin muotoon, aikaansaannos ilmaistaan stixeleinä. Kolmannessa vaiheessa stixeleiden tarkkaa liiketilaa arvioidaan hyödyntämällä ”6D-Vision” periaatetta. Neljännessä vaiheessa stixelit luokitellaan tapahtuneen liikkeen perusteella. (López ym. 2017, 36–40)

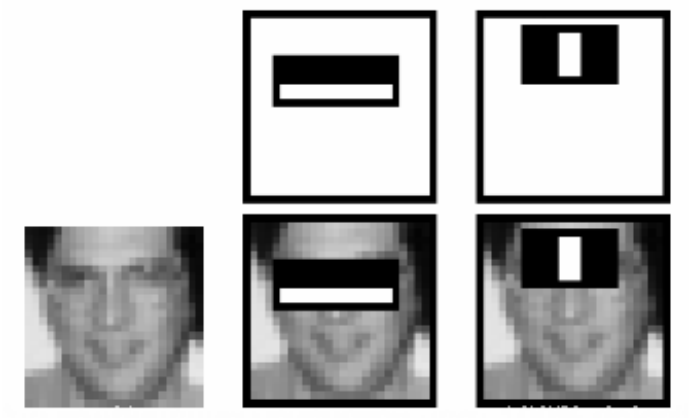
Autonomisessa kulkuneuvossa konenäköön perustuvalla objektin tunnistuksella on kolme päätavoitetta. Ensimmäinen tavoite on jalankulkijoiden ja pyöräilijöiden turvallisuuden

takaaminen. Toinen tavoite on muiden kulkuneuvojen havaitseminen. Kolmas tavoite on liikennevalojen reaaliaikainen tarkkailu. Objektintunnistus järjestelmä koostuu kahdesta päämoduulista. Ensimmäinen moduuli on kiinnostava-alue eli ”ROI”. Kiinnostavat alueet selvitetään kamerakuvasta, kohdistamalla havaittuun kuvaan lukuisia hypoteeseja. Hypoteesit helpottavat kiinnostavan objektin talteen ottamista. Toinen moduuli on objektien luokittelu. Autonomisen kulkuneuvon toiminnan kannalta merkitykselliset objektit ryhmitellään kolmeen luokkaan. Luokkia voivat olla esimerkiksi: jalankulkijat, kulkuneuvot, liikennevalot. Edellä mainittuja luokkia pyritään tunnistamaan läheltä ja kaukaa. Lähietäisyydeltä jalankulkijat tunnistetaan kuvaominaisuuksien ja -muotojen perusteella. Edellä mainittua menetelmää voidaan myös soveltaa lähietäisyydellä olevien kulkuneuvojen havaitsemisessa. Kaukana olevia kulkuneuvoja havaitaan hyödyntämällä ”Viola-Jones” tunnistinta. Liikennevalojen tunnistuksessa hyödynnetään neuroverkkoja, neuroverkot parantavat tapahtuneen tunnistuksen luotettavuutta. Tunnistusta mahdollisesti hankaloittavia tekijöitä ovat: liikennevalojen matala kirkkaus ja suuntaava rakenne. (López ym. 2017, 43–48)

3.5 Konenäkö algoritmit

3.5.1 Haar cascade

Haar cascade on ”objektintunnistusalgoritmi, jota käytetään kuvassa olevien kasvojen tunnistamiseen” (Behera 24.12.2020). Algoritmin kehitystä inspiroi matemaattinen konsepti, joka tunnetaan nimellä ”Haar wavelets”. Algoritmi havaitsee kuvissa ilmenevän objektin luontaisen rakenteen sekä yhtäläisyydet. Hyödyntämällä kuvissa ilmeneviä ominaisuuksia algoritmi kykenee luomaan objektista yleisen tunnistusmallin. Haar cascade luokittelija luodaan kahdessa vaiheessa. Ensimmäisessä vaiheessa algoritmille syötetään positiivisia sekä negatiivisia kuvia. Positiiviset kuvat sisältävät objektin, jota pyritään havaitsemaan. Negatiiviset kuvat eivät sisällä havaittavaa objektia. Toisessa vaiheessa kuvista poimitaan haar-ominaisuuksia, hyödyntämällä konvoluutioytimiä. Positiiviset kuvat peitetään konvoluutioytimillä, ytimien avulla kuvassa ilmeneville ominaisuuksille lasketaan ominaisuusarvot. Algoritmin tuottamia ominaisuusarvoja verrataan käyttäjän määrittelemään ominaisuusarvo kynnykseen. Kynnyksen ylittyessä ominaisuus täyttää käyttäjän määrittelemän laatu vaatimuksen. Ominaisuusarvojen laskennassa hyödynnetään integraalikuvia, kuvat vähentävät haar-ominaisuuksien laskennallista taakkaa. (Kapur & Thakkar 2015, luku 4)

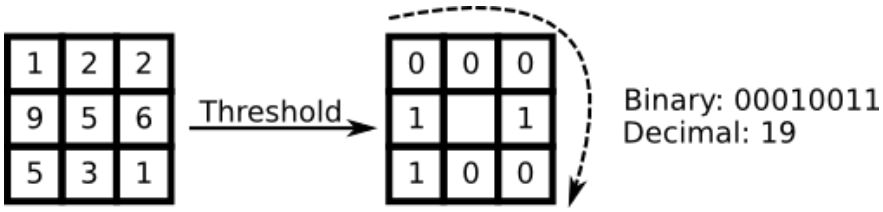


Kuva 1. Face Detection using Haar Cascades (OpenCV s.a.)

3.5.2 LBP (Local binary pattern)

LBP-algoritmin toiminta perustuu binäärisiin lukuihin, lukujen laskennassa hyödynnetään tilastollisia sekä rakenteellisia menetelmiä. Algoritmin toiminnan kannalta binäärisillä luvuilla on kaksi päätehtävää. Binäärilukujen ensimmäinen päätehtävä on kuvaelementtien ympäristön kuvailu. Toinen päätehtävä on kuvassa ilmenevien yksittäisten ominaisuuksien tunnistus. Ominaisuuksien tunnistuksessa algoritmi keskittyy havaittujen ominaisuuksien mikrorakenteisiin, kuten kulmiin tai viivoihin. (Harmouch 19.06.2020)

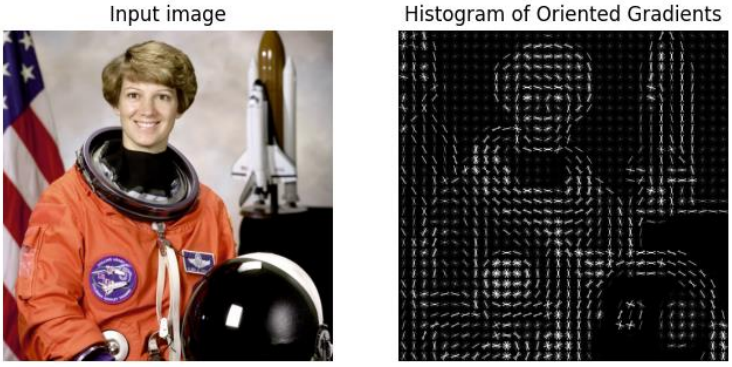
Algoritmi jakaa kuvat pieniin lohkoihin, lopputuloksena syntyy pikseliarvo matriisi. Matriisin koko vaihtelee, mutta esimerkiksi 3x3 matriisi sisältää yhdeksän lohkoa. Matriisin keskimmäistä pikseliä kutsutaan paikalliseksi pikseliksi. Pikseliarvo matriisin perusteella luodaan binäärinen matriisi, lohkojen binäärinen arvo saadaan vertailemalla paikallisen pikselin pikseliarvoa ympäröivien lohkojen pikseliarvoihin. Kahdeksan bitin pikseliarvo on dynaaminen kokonaisluku, joka liikkuu 0 ja 255 välillä. Mikäli paikallisen pikselin pikseliarvo on suurempi kuin naapuri lohkon pikseliarvo binääriseen matriisin kirjataan 0, mikäli paikallinen pikseliarvo on pienempi matriisiin, kirjataan 1. Binäärisen numerojonon perusteella paikalliselle pikselille lasketaan desimaaliarvo. Lopputuloksena syntyy LBP-ominaisuusvektori, joka mahdollistaa objektin tunnistuksen. (Escrivá & Laganiere 2019, luku 14; Singh 16.03.2021)



Kuva 2. Local Binary Patterns (JuliaImages s.a.)

3.5.3 HOG (Histogram of oriented gradients)

HOG-algoritmin toiminta perustuu pikselitasolla tapahtuvien suunnanmuutosten tarkkailuun, algoritmi kiinnittää huomiota erityisesti objektin rakenteeseen sekä muotoon. Algoritmi pilkkoo kuvat pienempiin osiin, joita kutsutaan soluiksi. Jokaiselle solulle lasketaan kaltevuusjoukko, jota kutsutaan gradientiksi. Gradienttien avulla pikselitasolla tapahtunutta liikemuutosta pystytään ilmaisemaan visuaalisesti, kyseistä kuvaajaa kutsutaan histogrammi kaavioksi. Soluissa gradientit ilmenevät värillisinä läikkinä, läikät sisältävät erimittaisia viivoja, viivojen pituus ilmaisee gradienttien vahvuutta. Algoritmi pyrkii luomaan korkeatasoisen kuvaajan käsittelemällä solujen histogrammeja. Histogrammeja voidaan käsitellä esimerkiksi yhdistelemällä yksittäisiä soluja, muodostunutta kokonaisuutta kutsutaan lohkoksi. Lohkoille luodaan vektorit, jotka helpottavat lohkoissa tapahtuvien muutosten havaitsemista. (Howse & Minichino 2020, luku 7; Tyagi 04.07.2021)



Kuva 3. Histogram of Oriented Gradients (scikit-image s.a.)

4 Tulokset

4.1 Vaatimusmäärittely

4.1.1 Tietojoukon luomiseen soveltuva ohjelma (dataset-collector.py)

1. Ohjelman tulee luoda yhteys ip-kameraan, ip-kamera helpottaa negatiivisen tietojoukon luomista. Kamerassa on sisäänrakennettu panorointi, joka mahdollistaa kuvien eheyden eli vähäinen määrä duplikaatteja.

2. Ohjelman täytyy sisältää konsoli kirjautuminen, konsoli kirjautuminen on erityisen tärkeä turvallisuuden kannalta. Konsoli kirjautumisen avulla voidaan välttää kameran yhdistykseen vaadittavan käyttäjätunnuksen ja salasanan puskemista julkiselle verkkosivulle.

3. Ohjelman käyttäjällä tulee olla mahdollisuus valita kuinka monta kuvaa ohjelma nappaa. Kuvien lukumäärä kirjoitetaan käynnistyksen yhteydessä avautuvaan konsoli näkymään.

4. Ohjelman käynnistyksen yhteydessä ip-kameran yhteys validoidaan, jos ip-kameran yhdistyksessä ilmenee ongelmia, ohjelma tarkistaa löytyykö laitteistosta korvaavaa laitetta, kuten sisäänrakennettua web-kameraa.

5. Käynnistyksen yhteydessä luodaan pääkansio, pääkansio helpottaa tietojoukon organisointia, käsittelyä. Pääkansion nimi voidaan kova koodata, nimi voi olla esimerkiksi images.

6. Käyttäjällä tulee olla mahdollisuus nimetä alakansiot, alakansion nimi syötetään konsoliin, ohjelma tuottaa kuvat käyttäjän määrittelemään alakansioon. Käynnistyksen yhteydessä luodaan aina uusi alakansio. Kaikki alakansiot luodaan kovakoodattuun pääkansioon.

7. Ohjelman nappaamat kuvat tulee nimetä seuraavasti 0,1,2,3,4,5,6,7… jne.

8. Ohjelman täytyy seurata alakansioissa olevien kuvien lukumäärää dynaamisesti. Lukumäärän avulla voidaan välttää kuvien nimeämiseen liittyvät konfliktit, eli kuvien nimet pysyvät uniikkina.

9. Ohjelma tarkistaa onko alakansio olemassa, jos esimerkiksi test niminen alakansio on olemassa, ohjelma uudelleennimeää alakansion nimellä test1, eli nimeen lisätään lisäysnumero.

10. Ohjelman tulee kyetä yhtenäistämään kameran nappaamat kuvat, eli jokaisella kuvalla täytyy olla identtiset mitat esimerkiksi 1024x1024. Kuvien vakio väritys tulee olla grayscale, eli kuvat ovat harmaan värisiä.

11. Käyttäjällä tulee olla mahdollisuus valita minkä värisiä kuvia ohjelma nappaa, jos käyttäjä haluaa, että ohjelma nappaa punaisen, vihreän tai sinisen värisiä kuvia konsoliin syötetään R, G tai B. Värikanavien avulla kuvia voidaan tummentaa keinotekoisesti, kuvien tummennus helpottaa pimeässä tapahtuvaa kasvojen tunnistusta.

4.1.2 Kaskadin lukemiseen soveltuva ohjelma

1. Ohjelman täytyy luoda yhteys ip-kameraan, kameraa hyödynnetään objektin tunnistuksessa.

2. Ohjelman täytyy kyetä lukemaan objektin tunnistukseen soveltuvaa tiedostoa, kuten cascade.xml.

3. Ohjelma luo tunnistetun objektin ympärille rajauslaatikon, rajauslaatikkoa voidaan muokata visuaalisesti miellyttävämmän näköiseksi, esimerkiksi crosshair.

4.2 Toteutus (dataset-collector.py)

Tässä osiossa luodaan ohjelma, joka kykenee tuottamaan positiivisista tai negatiivisista kuvista koostuvan laajan tietojoukon.

4.2.1 Ip-kameran yhdistys ja konsoli kirjautuminen

Kuvassa 4 ilmenee koodipätkä, joka muodostaa yhteyden ip-kameraan, yhteyden muodostus vaatii käyttäjätunnuksen, salasanan sekä ip osoitteen. Käyttäjätunnus ja salasana luodaan ip-kameran mukana tulleessa ohjelmistossa. Rivillä 1 projektiin tuodaan cv2 kirjasto, kirjasto sisältää lukuisia konenäkö menetelmiä, jotka mahdollistavat projektin toteutuksen. Riveillä 3–4 luodaan kaksi muuttujaa username sekä password, konsoliin kirjoitettu syöte asetetaan muuttujien arvoksi. Rivillä 7 määritellään visuaalisen syötteen tulolähde, yhteyden muodostuksen vaatimuksena on, että käyttäjän syöttämät muuttuja arvot ovat olemassa ja ovat oikeat. Rivillä 10–11 luodaan while-loop, joka pitää visuaalisen syötteen tulolähteen auki. Rivillä 13 luodaan erillinen ikkuna, joka visualisoi kameran syötettä. Ikkunalle annetaan nimi sekä määritellään visuaalisen syötteen tulolähde. Riveillä 15–19 luodaan if-lauseke, joka mahdollistaa ikkunan sulkemisen, eli jos käyttäjä painaa näppäimistön q painiketta rivillä 10–11 määritelty while loop katkeaa.

```
1  import cv2
2
3  username = input("Enter ip-camera username: ")
4  password = input("Enter ip-camera password: ")
5  cameraip = "192.168.10.39:88"
6  #yhteys ip kameraan
7  cap = cv2.VideoCapture("rtsp://" + username + ":" + password + "@" + cameraip + "/videoSub")
8
9  #luodaan while loop, joka jatkuu ikuisesti, eli kamera yhteys pysyy auki
10 while True:
11     ret, frame = cap.read()
12     #nimetään kamera ikkuna, määritellään visuaalisen datan tulolähde
13     cv2.imshow("ipkameranäkymä", frame)
14     #käyttäjä voi katkaista while loopin painamalla näppäimistön q painiketta
15     if cv2.waitKey(1) & 0xFF == ord('q'):
16         break
17 # jos while loop katkaistaan kamera näkymä sammutetaan
18 cap.release()
19 cv2.destroyAllWindows()
```

Kuva 4. Ip-kameran yhdistys + konsoli kirjautuminen

4.2.2 Ip-kamerayhteyden validointi

Kuvassa 5 ilmenee koodipätkä, jolla ip-kamera yhteys validoidaan. Rivillä 9 muodostetaan if-lauseke, joka tarkistaa onko visuaalisen syötteen tulolähde olemassa tai onko tulolähde suljettu. Mikäli visuaalisessa tulolähteessä ilmenee ongelmia rivillä 10 määritellään visuaaliselle syötteelle uusi tulolähde, uusi tulolähde on tietokoneen sisäänrakennettu web-kamera. Rivillä 11 konsoliin kirjoitetaan viesti, jossa käyttäjälle ilmoitetaan, että yhteyden muodostus on epäonnistunut. Rivillä 12 määritellään mitä tapahtuu mikäli, visuaalisen syötteen tulolähteessä ei ilmene ongelmia. Rivillä 13 konsoliin kirjoitetaan viesti, jossa käyttäjälle ilmoitetaan, että yhteys ip-kameraan on muodostettu onnistuneesti. Rivillä 9–11 oleva koodipätkä laukeaa, mikäli ip-kamera ei ole kiinnitetty pistorasiaan, käyttäjä kirjoittaa väärän käyttäjätunnuksen tai salasanan, internet yhteydessä ilmenee ongelmia.

```
8   #jos primaarinen kamera ei ole saatavilla yhdistetään sekundääriseen kameraan
9   if cap is None or not cap.isOpened():
10      cap = cv2.VideoCapture(0)
11      print("Ip-kamera ei ole saatavilla tarkista käyttäjätunnus ja salasana, yhdistetään web-kameraan")
12  else:
13      print("Ip-kamera saatavilla, yhdistetään ip-kameraan")
14
```

Kuva 5. Ip-kamera yhteyden validointi

4.2.3 Pääkansion luominen

Kuvassa 6 ilmenee koodipätkä, jolla luodaan pääkansio. Pääkansion tarkoitus on helpottaa napattujen kuvien organisointia. Rivillä 2 projektiin tuodaan os kirjasto, kirjaston metodit helpottavat kansioiden luomista/käsittelyä/hallintaa. Rivillä 6 luodaan muuttuja nimeltä paakansio, muuttujan arvoksi määritellään kovakoodattu merkkijono ’Kuvat’. Rivillä 9 luodaan if-lauseke, joka tarkistaa onko kyseisessä polussa kansiota nimeltä ’Kuvat’, koska erillistä polkua ei ole määritelty ’Kuvat’ kansio sijaitsee aina projektin root-kansiossa. Mikäli kansiota ei ole olemassa rivillä 10–11 konsoliin tulostetaan viesti, jossa käyttäjälle ilmoitetaan, että kyseistä kansiota ei ole olemassa, kansion luominen aloitetaan. Rivillä 12 luodaan kovakoodattu pääkansio nimeltä ’Kuvat’, kansio luodaan sovelluksen käynnistyksen yhteydessä. Mikäli käyttäjä on aiemmin käynnistänyt sovelluksen rivillä 13 määritelty else-lauseke aktivoituu, rivillä 14 konsoliin tulostetaan viesti, jossa käyttäjälle ilmoitetaan, että kyseinen pääkansio on jo olemassa.

```
2   import os
3   username = input("Enter ip-camera username: ")
4   password = input("Enter ip-camera password: ")
5   cameraip = "192.168.10.39:88"
6   paakansio = "Kuvat"
7
8   #pääkansion luominen
9   if not os.path.exists(paakansio):
10      print("Kansio nimeltä: " + paakansio + " ei ole olemassa...")
11      print("Luodaan kansiota nimeltä: " + paakansio)
12      os.makedirs(paakansio)
13  else:
14      print("Kansio nimeltä: " + paakansio + " on jo olemassa")
15
```

Kuva 6. Pääkansion luominen

4.2.4 Alakansioiden luominen sekä uudelleen nimeäminen

Kuvassa 7 ilmenevä koodipätkä luo alakansion käyttäjän syötteen perusteella. Rivillä 7 luodaan muuttaja nimeltä alakansio, muuttujan arvoksi määritellään käyttäjän konsoliin kirjoittama syöte. Käyttäjän kirjoittama syöte asetetaan alakansion nimeksi. Suuria tietojoukkoja käsiteltäessä alakansiolla pyritään helpottamaan haluttujen kuvien löytämistä, esimerkiksi alakansio nimeltä ’olohuone’ sisältää pelkästään olohuoneesta napattuja kuvia. Rivillä 8 luodaan muuttuja nimeltä alakansioninkramentti, muuttujan arvoksi määritellään numeerinen arvo 1, eli kansioiden inkramentointi aloitetaan luvusta 1. Rivillä 11 luodaan if-lauseke, joka tarkistaa löytyykö kyseistä alakansiota määritellystä polusta. Mikäli alakansiota ei löydy, alakansion luominen aloitetaan rivillä 12, luomisen yhteydessä alakansiolle määritellään polku. Rivillä 13 konsoliin tulostetaan viesti, joka kertoo käyttäjälle, että alakansion luominen on aloitettu. Rivillä 11–13 oleva koodipätkä aktivoituu vain, jos käyttäjä luo alakansiota ensimmäistä kertaa. Rivillä 14 luodaan else-lauseke, eli määritellään mitä ohjelmassa tapahtuu, jos käyttäjän nimeämä alakansio on jo olemassa. Rivillä 15 luodaan while-loop, joka tarkistaa löytyykö pääkansiosta saman nimisiä alakansioita. Mikäli käyttäjän syötteen perusteella luotu alakansio löytyy pääkansiosta, rivillä 16 luodaan alakansiolle inkramentoitu numeerinen arvo. Inkramentoidun arvon avulla alakansioiden nimet pysyvät uniikkeina, vanhat kuvat eivät vahingossakaan tule korvatuksi. Rivillä 17 luodaan inkramentoitu alakansio, alakansion luomisessa hyödynnetään os kirjaston menetelmiä. Rivillä 18 konsoliin tulostetaan viesti, jossa käyttäjälle ilmoitetaan, että käyttäjän syötteen perusteella luotu alakansio on jo olemassa. Samalla käyttäjälle ilmoitetaan, että ohjelma uudelleen nimeää alakansion.

```
alakansio = input("Syötä alakansion nimi: ")
alakansioninkramentti = 1

#alakansion luominen, uudellen nimeäminen
if not os.path.exists(paakansio+"/"+alakansio):
    os.makedirs(paakansio + "/" + alakansio)
    print("Luodaan alakansiota nimeltä: " + alakansio + "...")
else:
    while os.path.exists(paakansio + "/" + alakansio + (str(alakansioninkramentti))):
        alakansioninkramentti += 1
    os.makedirs(paakansio+"/"+alakansio + (str(alakansioninkramentti)))
    print("Alakansio: " + alakansio + " on olemassa, uudelleen nimetään kansio nimellä: " + alakansio + (str(alakansioninkramentti)))
```

Kuva 7. Alakansioiden luominen sekä uudelleen nimeäminen

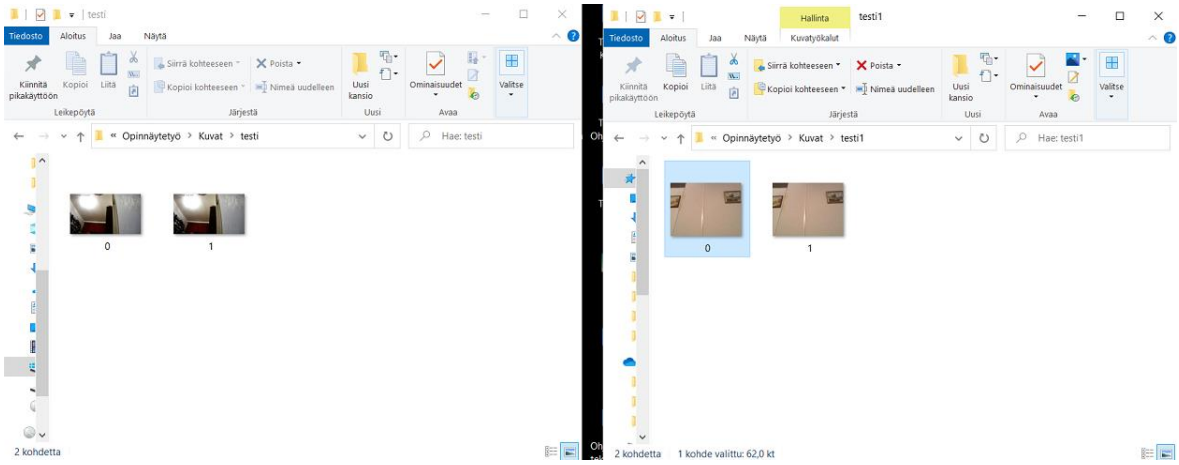
4.2.5 Kuvien luominen

Kuvassa 8 ilmenevä koodipätkä mahdollistaa kuvien luomisen, tämä koodipätkä on päivitetty versio kuvassa 4 ilmenevästä koodipätkästä. Tässä osiossa puhun pelkästään päivitetystä ominaisuuksista eli en toista asioita, jotka olen jo maininnut kohdassa 4.2.1. Rivillä 37 luodaan muuttuja nimeltä kuvainkramentti, muuttujan arvoksi määritellään numeerinen arvo 0, eli inkramentointi aloitetaan luvusta 0. Rivillä 43 luodaan if-lauseke, joka tarkistaa sisältääkö käyttäjän syöttämä alakansion nimi inkramentoituvia lukuja. Mikäli ohjelma löytää alakansion nimestä inkramentti luvun, kuvat luodaan inkramentoituun alakansioon. Rivillä 44 luodaan muuttuja nimeltä kuvanimi, muuttujan arvoksi määritellään alakansion polku, tiedostopääte sekä tiedostonimi. Kuvat luodaan edellä mainittuun polkuun, tiedostopäätteeksi määritellään jpg, kuvat nimetään kuvainkramentin perusteella. Rivillä 45 luodaan else-lauseke, eli määritellään mitä ohjelmassa tapahtuu, jos käyttäjän syöttämä alakansio nimi ei sisällä inkramentoituvia lukuja. Rivillä 46 luodaan muuttuja nimeltä kuvanimi, muuttuja on lähes identtinen verrattuna rivillä 44 luotuun muuttujaan, ainut poikkeavuus on, että alakansion polku ei sisällä inkramentti lukuja. Rivillä 47 suoritetaan kuvien tallennus, tallennus toteutetaan cv2 kirjaston metodilla. Metodiin syötetään kaksi parametria, tallennuksen sijainti sekä visuaalisen syötteen tulolähde. Rivillä 48 muuttujalle kuvainkramentti määritellään kasvava arvo 1. Tallennetut kuvat nimetään edellä mainitun muuttujan perusteella seuraavasti: 0, 1, 2, 3, 4, 5…jne. Rivillä 50 if-lausekkeeseen lisätään uusi ehto, joka mahdollistaa while-loopin katkaisun. Mikäli kuvainkramentti muuttuja on tasan 2 while-loop katkaistaan, koska aloitusarvo on 0 ja kasvava arvo on 1, ohjelma ottaa 2 valokuvaa ennen kuin while-loop katkeaa. Rivillä 53 luodaan if-lauseke, joka tarkistaa luotiinko ohjelman käynnistyksen yhteydessä inkramentoitu alakansio. Mikäli käynnistyksen yhteydessä luodaan inkramentoitu alakansio, rivillä 54 konsoliin tulostetaan viesti, jossa käyttäjälle ilmoitetaan, että käyttäjän luomat kuvat löytyvät inkramentoidusta alakansiosta. Rivillä 55 luodaan else-lauseke, joka aktivoituu, mikäli käynnistyksen yhteydessä luotu alakansio ei sisällä inkramentti lukua. Rivillä 56 konsoliin tulostetaan viesti, jossa käyttäjälle ilmoitetaan minkä nimiseen alakansioon kuvat tallennettiin.

```
kuvainkramentti = 0
#luodaan while loop, joka jatkuu ikuisesti, eli kamera yhteys pysyy auki
while True:
    ret, frame = cap.read()
    # nimetään kamera ikkuna, määritellään visuaalisen datan tulolähde
    cv2.imshow("kameranäkymä", frame)
    if os.path.exists(paakansio+"/"+alakansio+(str(alakansioninkramentti))):
        kuvanimi = paakansio + "/" + alakansio + (str(alakansioninkramentti)) + "/%d.jpg" % kuvainkramentti
    else:
        kuvanimi = paakansio + "/" + alakansio + "/%d.jpg" % kuvainkramentti
    cv2.imwrite(kuvanimi, frame)
    kuvainkramentti += 1
    # käyttäjä voi katkaista while loopin painamalla näppäimistön q painiketta
    if kuvainkramentti == 2 or cv2.waitKey(1) & 0xFF == ord('q'):
        break
#konsoli viesti kuvien sijainnista
if os.path.exists(paakansio+"/"+alakansio+(str(alakansioninkramentti))):
    print("Kuvat luotu kansioon: " +alakansio+(str(alakansioninkramentti)))
else:
    print("Kuvat luotu kansioon: " + alakansio)
```

Kuva 8. Kuvien luominen, päivitetty while-loop

Kuvassa 9 ilmenee kaksi ohjelman käynnistyksen yhteydessä luotua alakansiota, molemmat alakansiot sisältävät ohjelman luomia valokuvia. Alakansioiden luominen/uudelleen nimeäminen tapahtuu onnistuneesti, molemmissa tapauksissa alakansion nimeksi syötettiin ’testi’. Vasemmanpuoleisessa alakansiossa visuaalisen syötteen tulolähteenä käytettiin ip-kameraa, eli tallennetut kuvat ovat peräisin internettiin yhdistetystä valvontakamerasta. Oikeanpuoleisessa alakansiossa visuaalisen syötteen tulolähteenä käytettiin web-kameraa, eli tallennetut kuvat ovat peräisin kannettavan tietokoneen sisäänrakennetusta kamerasta. Ensimmäinen merkittävä ongelma ilmenee tallennettujen kuvien nimissä, molemmissa alakansioissa valokuville on luotu identtiset nimet. Tietojoukon muodostuksen yhteydessä identtiset nimet tulevat aiheuttamaan kuvien korvausta koskevia ongelmia, eli osa tallennetuista kuvista menetetään.



Kuva 9. Kuvien luominen, ensimmäinen testi

Kuvassa 10 ilmenevä koodipätkä on päivitetty versio kuvassa 8 ilmenevästä koodipätkästä. Kyseinen koodipätkä antaa ohjelman käyttäjälle mahdollisuuden valita kuinka monta valokuvaa ohjelma tallentaa. Samalla ratkaistaan kuvassa 9 ilmenevä kuvien

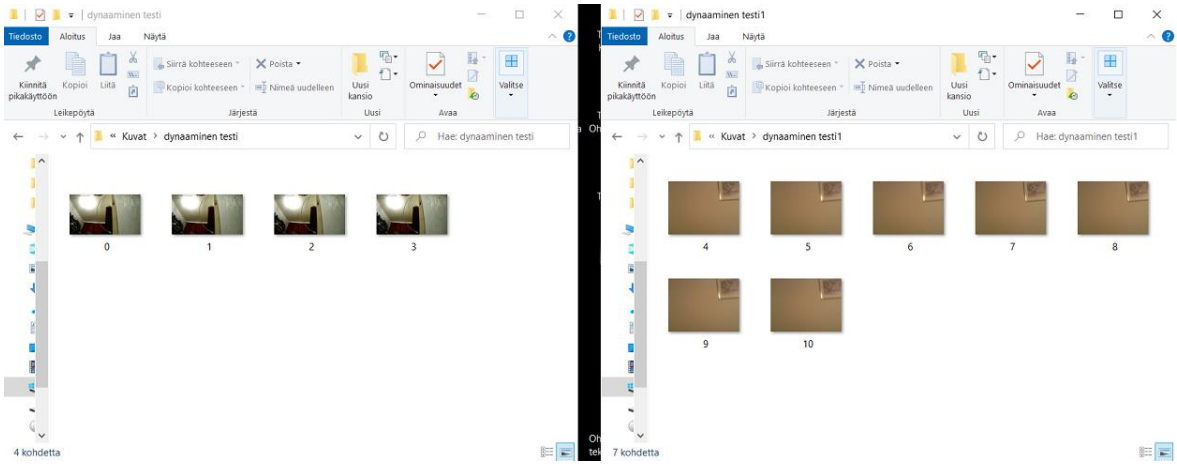
nimeämiseen kohdistuva ongelma. Rivillä 37 luodaan muuttuja nimeltä haluttukuvamaara, muuttuja arvoksi määritellään käyttäjän konsoliin syöttämä numeerinen arvo. Rivillä 38 luodaan muuttuja nimeltä kuvienkokonaismaara, muuttujan arvoksi määritellään numeerinen arvo 0. Rivillä 39 luodaan muuttuja nimeltä alakansioidenkokonaismaara, muuttujan arvoksi määritellään numeerinen arvo 0. Rivillä 41 luodaan for-loop, joka tarkkailee kolmea muuttujaa. Polku niminen muuttuja poimii talteen pääkansioon luodun alakansion polun. Kansiot niminen muuttuja luo listan, joka sisältää pääkansiossa olevien alakansioiden nimet. Tiedostot niminen muuttuja luo listan, joka sisältää pääkansiossa olevien tiedostojen nimet. Rivillä 42 lasketaan tallennettujen kuvien lukumäärä tiedostot nimisestä listasta, lukumäärä määritellään muuttujan nimeltä kuvienkokonaismaara arvoksi. Rivillä 43 lasketaan luotujen alakansioiden lukumäärä kansiot nimisestä listasta, lukumäärä määritellään muuttujan nimeltä alakansioidenkokonaismaara arvoksi. Rivillä 44 konsoliin tulostetaan viesti, jossa käyttäjälle ilmoitetaan kuinka monta alakansiota pääkansio sisältää. Rivillä 45 konsoliin tulostetaan viesti, jossa käyttäjälle ilmoitetaan kuinka monta kuvaa alakansiot sisältävät. Rivillä 47 kuvainkramentti nimisen muuttujan alkuarvoksi määritellään kuvienkokonaismaara niminen muuttuja. Käynnistyksen yhteydessä ohjelma laskee kuinka monta kuvaa alakansiot sisältävät, uusien kuvien nimeäminen aloitetaan olemassa olevien kuvien lukumäärän perusteella. Rivillä 59 päivitetään if-lauseketta, joka mahdollistaa while-loopin katkaisun. Mikäli uudistetun kuvainkramentti muuttujan kasvava arvo on yhtä suuri kuin haluttujen kuvien sekä tallennettujen kuvien yhteenlaskettu lukumäärä while-loop katkaistaan.

```
37  haluttukuvamaara = input("Syötä tallennettavien kuvien lukumäärä: ")
38  kuvienkokonaismaara = 0
39  alakansioidenkokonaismaara = 0
40  #dynaaminen kuvien nimeäminen
41  for polku, kansiot, tiedostot in os.walk(paakansio):
42      kuvienkokonaismaara += len(tiedostot)
43      alakansioidenkokonaismaara += len(kansiot)
44  print(paakansio + " kansio sisältää: " + (str(alakansioidenkokonaismaara)) + " alakansiota")
45  print("Alakansiot sisältävät: " + (str(kuvienkokonaismaara)) + " kuvaa")
46  #luodaan while loop, joka jatkuu ikuisesti, eli kamera yhteys pysyy auki
47  kuvainkramentti = kuvienkokonaismaara
48  while True:
49      ret, frame = cap.read()
50      # nimetään kamera ikkuna, määritellään visuaalisen datan tulolähde
51      cv2.imshow("kameranäkymä", frame)
52      if os.path.exists(paakansio+"/"+alakansio+(str(alakansioninkramentti))):
53          kuvanimi = paakansio + "/" + alakansio + (str(alakansioninkramentti)) + "/%d.jpg" % kuvainkramentti
54      else:
55          kuvanimi = paakansio + "/" + alakansio + "/%d.jpg" % kuvainkramentti
56      cv2.imwrite(kuvanimi, frame)
57      kuvainkramentti += 1
58      # käyttäjä voi katkaista while loopin painamalla näppäimistön q painiketta
59      if kuvainkramentti == int(haluttukuvamaara) + int(kuvienkokonaismaara) or cv2.waitKey(1) & 0xFF == ord('q'):
60          break
```

Kuva 10. Kuvien luominen, dynaaminen versio

Kuvassa 11 suoritetaan ohjelman toimivuuden testaus, testi toteutettiin kuvassa 10 toteutettujen päivitysten jälkeen. Testin avulla selvitetään, kykeneekö ohjelma nimeämään kuvia dynaamisesti. Ohjelma käynnistettiin kahdesti, ensimmäisen käynnistyksen

yhteydessä luotiin vasemmanpuoleinen alakansio sekä alakansion sisältö. Alakansiossa olevien kuvien luomisessa visuaalisen syötteen tulolähteenä toimi verkkoon yhdistetty ip-kamera. Vasemmanpuoleisessa alakansiossa haluttujen kuvien lukumääräksi syötettiin 4 kappaletta, ohjelma kykeni luomaan ja nimeämään kuvat onnistuneesti. Ohjelman luomat kuvat nimettiin seuraavasti: 0, 1, 2, 3. Toisen käynnistyksen yhteydessä luotiin oikeanpuoleinen alakansio sekä alakansion sisältö. Alakansiossa olevien kuvien luomisessa visuaalisen syötteen tulolähteenä toimi kannettavan tietokoneen web-kamera. Oikeanpuoleisessa alakansiossa haluttujen kuvien lukumääräksi syötettiin 7 kappaletta, ohjelma kykeni luomaan ja nimeämään kuvat dynaamisesti. Ohjelman luomat kuvat nimettiin seuraavasti: 4, 5, 6, 7, 8, 9, 10. Molemmissa tapauksissa alakansion nimeksi syötettiin ’dynaaminen testi’, ohjelma suoritti alakansioiden luomisen sekä uudelleen nimeämisen onnistuneesti. Testi oli menestyksekäs kuvassa 9 havaittu ongelma ratkesi, ohjelman luomissa alakansioissa ei ilmennyt identtisiä nimiä. Alakansioiden sisällöstä on mahdollista muodostaa laaja tietojoukko korvaamatta/menettämättä yhtäkään tallennettua kuvaa.



Kuva 11. Kuvien luominen, toinen testi

4.2.6 Kuvien yhtenäistäminen sekä värikanavan valinta

Kuvassa 12 ilmenevä koodipätkä yhtenäistää ohjelman tallentamat kuvat sekä antaa ohjelman käyttäjälle mahdollisuuden valita mitä värikanavaa kuvien tallentamisessa käytetään. Rivillä 48 projektiin tuodaan np niminen kirjasto, kirjasto sisältää lukuisia taulukoiden käsittelyä helpottavia funktioita, metodeja. Rivillä 49 konsoliin tulostetaan viesti, jossa käyttäjälle ilmoitetaan mahdollisista värikanava vaihtoehdoista. Rivillä 50 luodaan muuttuja nimeltä haluttuvarikanava, muuttujan arvoksi määritellään konsoliin kirjoitettu syöte. Rivillä 52–57 olevan koodin toiminta periaate on selostettu aikaisemmissa kappaleissa. Tarkempi selostus löytyy kappaleesta 4.2.1 ja 4.2.5. Rivillä 58 luodaan muuttuja nimeltä mitoitus, muuttujan arvoksi määritellään uudelleen mitoitettu visuaalinen syöte. Tallennettavia kuvia pyritään yhtenäistämään määrittelemällä visuaaliselle syötteelle kiinteä leveys sekä korkeus. Rivillä 59 luodaan muuttuja nimeltä harmaa, muuttujan arvoksi määritellään harmaan värinen uudelleen mitoitettu visuaalinen syöte. Muuttujan ansiosta kuvat voidaan

tallentaa harmaan värisinä. Rivillä 60 luodaan muuttuja nimeltä nollamatriisi, muuttujan arvoksi määritellään mitoitus nimisen muuttujan perusteella luotu 0 taulukko. Edellä mainittu muuttuja toimii tyhjänä värikanavana, jossa numeerinen arvo on aina 0. Rivillä 61 luodaan kolme värikanava muuttujaa, muuttujat nimetään seuraavasti: punainen, virhea, sininen. Edellä mainitut muuttujat luodaan pilkkomalla mitoitus niminen muuttuja kolmeen värikanavaan. Rivillä 62 punainen nimiselle muuttujalle määritellään arvo yhdistämällä kolme värikanavaa. Sinisen ja vihreän värikanavan arvona käytetään nollamatriisi nimistä muuttujaa, punaisen värikanavan arvona käytetään punainen nimistä muuttujaa. Mikäli ohjelman käyttäjä tallentaa kuvat hyödyntämällä punainen nimistä muuttuja, kaikki ohjelman tallentamat kuvat tallennetaan punaisen värisinä. Rivillä 63–64 toistetaan edellä mainittu prosessi, muuttujille nimeltä vihrea ja sininen määritellään arvo. Värikanavien yhdistyksen yhteydessä, halutun värikanavan arvona käytetään kyseisen värin muuttujaa, ei haluttujen värikanavien arvona käytetään nollamatriisi nimistä muuttujaa. Muuttujien vihrea ja sininen ansiosta ohjelma kykenee tallentamaan kuvat sinisen ja vihreän värisinä. Rivillä 65–66 luodaan if-lauseke, mikäli käyttäjä syöttää haluttuvarikanava nimisen muuttujan arvoksi numeron 1, kuvat tallennetaan uudelleen mitoitettuna normaalin värikanavan kautta. Rivillä 67–74 luodaan neljä elif-lauseketta, lausekkeiden ansiosta käyttäjä kykenee valitsemaan värikanavan, jonka kautta kuvat tallennetaan. Mikäli käyttäjä syöttää haluttuvarikanava nimisen muuttujan arvoksi numeron 2, 3, 4 tai 5. Kuvat tallennetaan harmaan, punaisen, vihreän tai sinisen värisinä. Rivillä 75–76 luodaan else-lauseke, joka määrittelee mitä ohjelmassa tapahtuu, mikäli haluttuvarikanava nimisen muuttujan arvoksi annetaan määrittelemätön numero. Kyseisessä tilanteessa kuvat tallennetaan uudelleen mitoitettuna normaalin värikanavan kautta.

```
import np
print("Värikanava vaihtoehdot: 1 = normaali, 2 = harmaa, 3 = punainen, 4 = vihreä, 5 = sininen")
haluttuvarikanava = input("Valitse värikanava: ")
while True:
    ret, frame = cap.read()
    cv2.imshow("kameranäkymä", frame)
    if os.path.exists(paakansio+"/"+alakansio+(str(alakansioninkramentti))):
        kuvanimi = paakansio + "/" + alakansio + (str(alakansioninkramentti)) + "/%d.jpg" % kuvainkramentti
    else:
        kuvanimi = paakansio + "/" + alakansio + "/%d.jpg" % kuvainkramentti
    mitoitus = cv2.resize(frame, (1024, 1024))
    harmaa = cv2.cvtColor(mitoitus, cv2.COLOR_BGR2GRAY)
    nollamatriisi = np.zeros(mitoitus.shape[:2], dtype = "uint8")
    (punainen, vihrea, sininen) = cv2.split(mitoitus)
    punainen = cv2.merge([nollamatriisi, nollamatriisi, punainen])
    vihrea = cv2.merge([nollamatriisi, vihrea, nollamatriisi])
    sininen = cv2.merge([sininen, nollamatriisi, nollamatriisi])
    if int(haluttuvarikanava) == 1:
        cv2.imwrite(kuvanimi, mitoitus)
    elif int(haluttuvarikanava) == 2:
        cv2.imwrite(kuvanimi, harmaa)
    elif int(haluttuvarikanava) == 3:
        cv2.imwrite(kuvanimi, punainen)
    elif int(haluttuvarikanava) == 4:
        cv2.imwrite(kuvanimi, vihrea)
    elif int(haluttuvarikanava) == 5:
        cv2.imwrite(kuvanimi, sininen)
    else:
        cv2.imwrite(kuvanimi, mitoitus)
```

Kuva 12. Tallennettavien kuvien yhtenäistäminen, värikanavan valinta

Kuvassa 13 ilmenee neljä kuvaa, jotka tallennettiin testin toteutuksen yhteydessä. Kuvat tallennettiin käyttämällä neljää eri värikanavaa. Testi suoritettiin kuvassa 12 ilmenevän koodipätkän implementaation jälkeen. Visuaalisen syötteen tulolähteenä käytettiin verkkoon yhdistettyä ip-kameraa. Testi oli menestyksekäs, ohjelmaan implementoitu värikanavan valinta ominaisuus toimi halutulla tavalla, kuvien tallentamisessa/nimeämisessä ei ilmennyt ongelmia. Ohjelman käyttäjä kykeni onnistuneesti valitsemaan värikanavan, jota käytetään kuvien tallentamisessa.

Kuva 13. Värikanavien testaus

4.2.7 Reaaliaikainen kuvalaskuri ja keskitetty ikkuna

Kuvassa 14 ilmenevä koodipätkä luo kameranäkymä nimiseen ikkunaan reaaliaikaisen kuvalaskurin. Laskurin ansiosta käyttäjä voi seurata reaaliaikaisesti, kuinka monta kuvaa ohjelma on tallentanut. Laskurin toteutuksen yhteydessä kameranäkymä niminen ikkuna keskitetään näytön keskikohtaan. Rivillä 52 projektiin tuodaan pyautogui niminen kirjasto,

kyseinen kirjasto sisältää funktioita, jotka helpottavat laitteistoa koskevien tietojen selvittämistä. Rivillä 57–58 luodaan kaksi muuttujaa nimeltä kuvanleveys ja kuvankorkeus, muuttujien arvoiksi määritellään visuaalisen syötteen tulolähteen leveys sekä korkeus. Rivillä 60–61 luodaan kaksi muuttujaa nimeltä ruudunleveys ja ruudunkorkeus, muuttujien arvoiksi määritellään käyttäjän tietokoneruudun leveys sekä korkeus. Rivillä 62 luodaan muuttuja nimeltä resoluutio, muuttujan arvoksi määritellään muuttujien kuvanleveys ja kuvankorkeus yhdistelmä. Muuttujan avulla pyritään helpottamaan koodin lukemista. Rivillä 63 luodaan muuttuja nimeltä fontti, muuttujan arvoksi määritellään teksti tyyli. Rivillä 64 luodaan muuttuja nimeltä kuvalaskuri, muuttujan arvoksi määritellään kolmesta muuttujasta koostuva matemaattinen laskutoimitus. Laskutoimituksen ansiosta käyttäjällä on mahdollisuus seurata kuvien tallentamista reaaliaikaisesti. Rivillä 65–66 visuaaliseen syötteeseen kirjoitetaan kaksi tekstiä, rivillä 65 tekstiksi määritellään kuvalaskuri niminen muuttuja. Rivillä 66 tekstiksi määritellään resoluutio niminen muuttuja, eli saatavilla olevan kameran resoluutio. Tekstien sijainniksi määritellään visuaalisen syötteen vasemmanpuoleinen alakulma. Molemmissa tapauksissa tekstien tyylinä käytetään fontti nimistä muuttujaa, fonteille määritellään identtinen fonttikoko, väri, paksuus sekä rivityyppi. Rivillä 68 kameranäkymä nimiselle ikkunalle määritellään uusi sijainti, uudeksi sijainniksi määritellään käyttäjän tietokoneruudun keskipiste.

```
52  import pyautogui
53  # luodaan while loop, joka jatkuu ikuisesti, eli kamera yhteys pysyy auki
54  while True:
55      ret, frame = cap.read()
56      # ratkaistaan visuaalisen syötteen tulolähteen leveys ja korkeus
57      kuvanleveys = cap.get(cv2.CAP_PROP_FRAME_WIDTH)
58      kuvankorkeus = cap.get(cv2.CAP_PROP_FRAME_HEIGHT)
59      # ratkaistaa ruudun leveys, korkeus
60      ruudunleveys = pyautogui.size().width
61      ruudunkorkeus = pyautogui.size().height
62      resoluutio = str(kuvanleveys) + "x" + str(kuvankorkeus)
63      fontti = cv2.FONT_HERSHEY_SIMPLEX
64      kuvalaskuri = str(kuvainkramentti-kuvienkokonaismaara)+"/"+str(haluttukuvamaara)
65      cv2.putText(frame, kuvalaskuri, (10, int(kuvankorkeus)-30), fontti, 0.5, (255, 255, 255), 1, cv2.LINE_4)
66      cv2.putText(frame, resoluutio, (10, int(kuvankorkeus) - 10), fontti, 0.5, (255, 255, 255), 1, cv2.LINE_4)
67      cv2.imshow("kameranäkymä", frame)
68      cv2.moveWindow("kameranäkymä", int(ruudunleveys / 2 - kuvanleveys / 2), int(ruudunkorkeus / 2 - kuvankorkeus / 2))
```

Kuva 14. Reaaliaikainen kuvalaskuri, keskitetty ikkuna

4.3 Toteutus (object-detector.py)

Tässä osiossa luodaan ohjelma, joka visualisoi luotuja xml luokittelijoita.

4.3.1 Havaitun objektin ympäröiminen rajauslaatikolla

Kuvassa 15 ilmenevä koodipätkä implementoi objektin tunnistimen sekä muodostaa rajauslaatikon havaitun objektin ympärille. Ohjelman toiminta vaatii ip-kamera yhteyden, konsoli kirjautumisen sekä kamera yhteyden validoinnin. Välttääkseni käsiteltyjen asioiden toistamisen mainitsen lyhyesti, että kuvan ulkopuolelle jätetyssä koodissa, eli rivillä 0–15 toistettiin kappaleissa 4.2.1 ja 4.2.2 mainitut toiminnallisuudet. Rivillä 17 luodaan muuttuja

nimeltä tunnistin, muuttujan arvoksi määritellään luokittelija tiedoston nimi. Luokittelija tiedosto löytyy custom-cascades nimisessä kansiossa, joka sijaitsee projektin juuressa. Tiedosto nimeltä cascadetest.xml on alkeellinen luokittelija, jonka tarkoitus on helpottaa rajauslaatikon luomista/visualisointia. Rivillä 21 luodaan muuttuja nimeltä objekti, muuttujan arvoksi määritellään visuaalinen syöte, johon muuttuja nimeltä tunnistin implementoidaan. Samalla objektin tunnistimelle määritellään parametrit, jotka kompensoiva havaittavan kohteen etäisyydessä tapahtuvia muutoksia. Rivillä 23 luodaan for-loop, joka tarkkailee dynaamisesti objekti nimisessä muuttujassa tapahtuvia koordinaatti muutoksia. Toisin sanoen luokittelija kykenee suorittamaan objektin tunnistuksen kohteen liikkeessä tapahtuneesta muutoksesta huolimatta. Parametrit: x, y, w ja h ovat määritelty objekti nimisen muuttujan koordinaateiksi. Rivillä 24 visuaaliseen syötteeseen piirretään rajauslaatikko havaitun objektin ympärille. Rajauslaatikon piirtäminen vaatii kaksi koordinaattia aloituskoordinaatti sekä lopetuskoordinaatti. Aloituskoordinaatiksi määritellään piste (x, y), eli rajauslaatikon vasemmanpuoleinen yläkulma. Lopetuskoordinaatiksi määritellään piste (x + w, y + h), eli rajauslaatikon oikeanpuoleinen alakulma. Lisäksi rajauslaatikolle määritellään väri sekä viivan paksuus.

```
16  #alkeellinen tunnistin, ohjelman käynnistämistä, kokeilua varten
17  tunnistin = cv2.CascadeClassifier('custom-cascades\cascadetest.xml')
18  while True:
19      ret, frame = cap.read()
20      #tunnistimen hienosäätö
21      objekti = tunnistin.detectMultiScale(frame, 1.1, 4)
22      #laatikon piirtäminen objektin ympärille
23      for (x, y, w, h) in objekti:
24          cv2.rectangle(frame, (x, y), (x + w, y + h), (255, 255, 255), 1)
25
26      cv2.imshow("objectdetection", frame)
27      if cv2.waitKey(1) & 0xFF == ord('q'):
28          break
29  cap.release()
30  cv2.destroyAllWindows()
```

Kuva 15. Rajauslaatikon luominen

4.3.2 Rajauslaatikon muokkaus

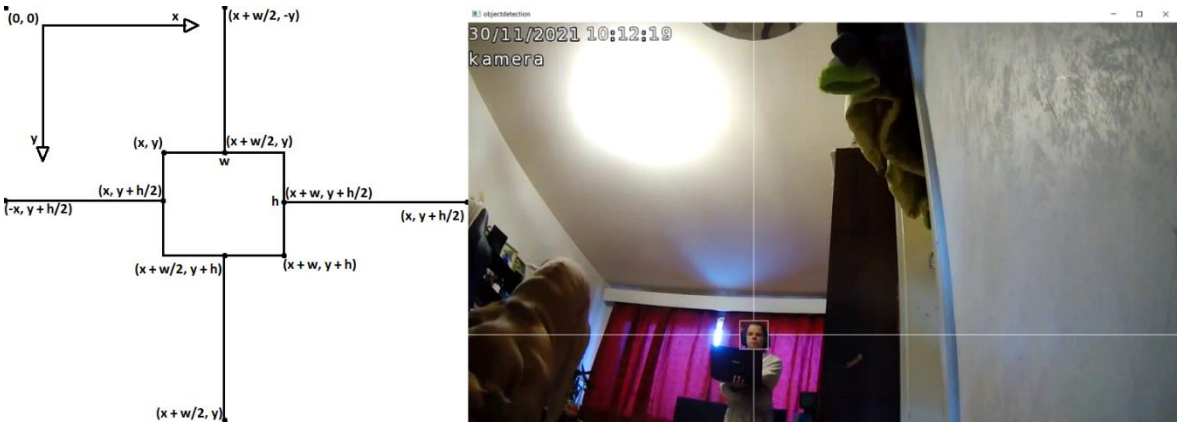
Kuvassa 16 ilmenevä koodipätkä muokkaa rajauslaatikon ulkonäköä ainutlaatuisemman näköiseksi. Koodipätkä ei vaikuta ohjelman toimivuuteen muuten kuin tekemällä rajauslaatikosta visuaalisesti miellyttävämmän näköisen, verrattuna muokkaamattomaan rajauslaatikkoon. Tarkoitus oli muokata rajauslaatikon ulkonäköä siten, että se muistuttaa tähtäintä. Rivillä 24–25 luodaan kaksi muuttujaa nimeltä leveys sekä korkeus, muuttujien arvoiksi määritellään saatavilla olevan visuaalisen syötteen tulolähteen korkeus ja leveys. Kyseisiä arvoja käytetään rajauslaatikkoon kiinnitettyjen viivojen pituuden määrittelyssä.

Rivillä 29–35 visuaaliseen syötteeseen piirretään neljä viivaa, viivojen piirtäminen vaatii kaksi koordinaattia aloituskoordinaatin sekä lopetuskoordinaatin. Jokaisen viivan aloituskoordinaatiksi määritellään rajauslaatikon kyseisen särmän keskipiste, tilanteesta riippuen viivojen lopetuskoordinaatiksi määritellään visuaalisen syöte ikkunan ulkopuolella oleva vertikaalinen tai horisontaalinen piste. Mikäli haluttu viiva on vertikaalinen, eli y-akselin suuntainen lopetuskoordinaatin määrittelyssä, hyödynnetään korkeus nimistä muuttujaa. Jos viiva on horisontaalinen, eli x-akselin suuntainen lopetuskoordinaatin määrittelyssä, hyödynnetään leveys nimistä muuttujaa. Lisäksi viivoille määritellään väri sekä viivan paksuus.

```
23      # selvitetään ikkunan leveys sekä korkeus, perusteella määritellään loppupiste koordinaatit
24      leveys = cap.get(cv2.CAP_PROP_FRAME_WIDTH)
25      korkeus = cap.get(cv2.CAP_PROP_FRAME_HEIGHT)
26      for (x, y, w, h) in objekti:
27          cv2.rectangle(frame, (x, y), (x + w, y + h), (255, 255, 255), 1)
28          # viiva ylhäällä, lopetuspisteeksi voidaan määritellä ikuinen -y koordinaatti
29          cv2.line(frame, (int(x) + int(w/2), int(y)), (int(x) + int(w/2), int(-korkeus)), (255, 255, 255), 1)
30          # viiva vasemmalla, lopetuspisteeksi voidaan vääritellä ikuinen -x koordinaatti
31          cv2.line(frame, (int(x), int(y) + int(h/2)), (int(-leveys), int(y) + int(h/2)), (255, 255, 255), 1)
32          # viiva alhaalla, lopetuspisteeksi ei voida määritellä ikuista y koordinaattia
33          cv2.line(frame, (int(x) + int(w/2), int(y) + int(h)), (int(x) + int(w/2), int(korkeus)), (255, 255, 255), 1)
34          # viiva oikealla, lopetuspisteeksi ei voida määritellä ikuista x koordinaattia
35          cv2.line(frame, (int(x) + int(w), int(y) + int(h/2)), (int(leveys), int(y) + int(h/2)), (255, 255, 255), 1)
```

Kuva 16. Rajauslaatikon ulkonäön muokkaus

Kuvassa 17 ilmenee kaksi kuvaa, vasemmanpuoleisessa kuvassa näkyy muokatun rajauslaatikon suunnitelma. Kuvassa hahmotellaan rajauslaatikon särmiin kiinnitettyjen suorille aloituskoordinaatit ja lopetuskoordinaatit. Oikeanpuoleisessa kuvassa suoritetaan suunnitelman pohjalta toteutetun rajauslaatikon toimivuuden testaus. Toteutettu testi oli menestys, rajauslaatikon särmiin kiinnitetyt suorat pysyivät kiinni määrittelyissä koordinaateissa objektin liikkeestä huolimatta, suorissa ei ilmennyt vääristymistä, eli kaikki suorat pysyivät horisontaalisina tai vertikaalisina objektin liikkuessa. Suorien dynaaminen pituus toimi erinomaisesti riippumatta objektin sijainnista kaikki suorat olivat aina sopivan mittaisia.



Kuva 17. Rajauslaatikon suunnitelma + muokatun rajauslaatikon testaus

4.4 Tietojoukkojen muodostus

Tässä luvussa muodostetaan luokittelijoiden koulutukseen soveltuvat tietojoukot. Tietojoukot muodostetaan hyödyntämällä luvussa 4.2 luotua ohjelmaa. Tarkoituksena on luoda kahdenlaisia luokittelijoita, yleisiä- sekä ennakkoluuloisia luokittelijoita. Yleisen luokittelijan avulla pyritään päättelemään, onko visuaalisessa syötteessä ilmenevä objekti ihmisen kasvot. Luokittelija ei kuitenkaan kykene päättelemään kenelle kasvot kuuluvat, mutta tietää että kyseessä ovat ihmisen kasvot. Luokittelijan tulee kyetä havaitsemaan kasvot kohteen sukupuolesta, iästä, etnisyydestä riippumatta. Ennakkoluuloisen luokittelijan avulla pyritään tunnistamaan visuaalisessa syötteessä ilmenevä objekti yksityiskohtien perusteella. Luokittelijan pyrkimyksenä on tunnistaa yksittäinen objekti, esimerkiksi onko visuaalisessa syötteessä ilmenevä objekti tietyn valmistajan tuottama pullo. Molemmissa tapauksissa luokittelijan koulutusta varten muodostetaan tietojoukko, joka koostuu positiivisista sekä negatiivisista valokuvista.

4.4.1 Tietojoukko (Yleinen luokittelija)

Positiivisten valokuvien luominen omatoimisesti on lähes mahdotonta, joten tutkimuksessa jouduttiin käyttämään kolmannen osapuolen luomia valokuvia. Kuvien luominen omatoimisesti on haastavaa, koska yleinen luokittelija vaatii suuren määrän monipuolisia valokuvia havaittavasta objektista. Valmiit valokuvat ovat peräisin Alexander Reben luomasta positiivisesta tietojoukosta. Edellä mainittu tietojoukko koostuu tekoälyn luomista kasvoista, eli yksikään valokuvassa ilmenevistä henkilöistä ei ole oikeasti olemassa. Kyseinen tietojoukko on opensource ja täten kaikkien saatavilla ilmaiseksi. (Reben 2019) Tutkimuksen toteutusta varten sivustolta ladattiin 60000 positiivista valokuvaa.

Negatiivisten valokuvien luominen aloitettiin poistamalla kuvaus ympäristöstä kaikki esineet, joissa positiivisissa valokuvissa ilmenevä objekti saattaa esiintyä. Poistettavia esineitä olivat esimerkiksi: taulut, valokuvat, dvd kannet, kirjat, patsaat sekä piirustukset. Seuraavaksi kameralle valittiin sijainti, jossa kameran näkökenttä on mahdollisimman laaja. Mitä laajempi kameran näkökenttä on sitä enemmän valokuviin, saadaan mahtumaan karsittavia objekteja. Kameran lopulliseksi sijainniksi valittiin kuvattavan huoneen nurkka.

Seuraavaksi muokattiin kameran PTZ (Pan, Tilt, Zoom) asetuksia. PTZ asetuksien muokkaus toteutettiin valvontakameran mukana tulleella ohjelmalla, muokattavia asetuksia olivat valvontakameran kääntönopeus sekä panorointirata. Kameran kääntönopeudeksi valittiin kaikista hitain vaihtoehto. Kääntönopeuden hidastamisella pyrittiin minimoimaan valokuvissa ilmenevää liikesumeutta. Kameran automatisoitu panorointirata luotiin määrittelemällä kahdeksan asetuspistettä, asetuspisteiden avulla valvontakameraa pystyttiin

kääntämään sekä pyörittämään akseleiden ympäri. Asetuspisteiden väliseksi viivästysajaksi määriteltiin 0 sekuntia, eli kamera pysyi jatkuvassa liikkeessä. Panorointiradan avulla pyrittiin välttämään duplikaatti valokuvien tallentamista, valvontakameran liikkuessa tallennetut valokuvat pysyvät monipuolisina.

Lopuksi käynnistettiin luvussa 4.2 luotu ohjelma, kaikki valokuvat tallennettiin harmaan värisinä. Valokuvia tallennettiin yhteensä 170000 kappaletta, valokuvia lähdettiin tallentamaan 10000 kappaleen erissä, yhden erän tallennus kesti noin 15 minuuttia. Kuvattavia huoneita olivat: makuuhuone, olohuone, eteinen ja luonto. Makuuhuoneesta tallennettiin 20000 valokuvaa, olohuoneesta tallennettiin 90000 valokuvaa, luonnosta tallennettiin 40000 valokuvaa ja eteisestä tallennettiin 20000 valokuvaa.

4.4.2 Tietojoukko (Ennakkoluuloinen luokittelija)

Positiivisten valokuvien luominen aloitettiin valitsemalla kuvausympäristö, jonka tausta on mahdollisimman tyhjä. Kuvausympäristöksi valittiin keittiö, koska tausta on valkoisen värinen eikä sisällä kuvausta hankaloittavia häiriötekijöitä. Seuraavaksi kuvausympäristöön asetettiin kuvattava objekti, kuvattavana objektina käytettiin gini pulloa. Valvontakameran ja kuvattavan objektin väliseksi etäisyydeksi määriteltiin 35 senttimetriä, tällä etäisyydellä kuvattavan objektin koko oli ideaalinen pysyäkseen kameran näkökentässä.

Lopuksi käynnistettiin luvussa 4.2 luotu ohjelma, valokuvia tallennuksessa hyödynnetiin neljää värikanavaa. Valokuvia tallennettiin harmaan-, punaisen-, vihreän- ja sinisen värisinä. Valokuvia tallennettiin yhteensä 40000 kappaletta, kuvia tallennettiin 10000 kappaleen erissä. Jokaisen erän tallennuksessa hyödynnettiin eri värikanavaa. Tässä tapauksessa valvontakameralle ei määritelty panorointirataa, koska tavoitteena oli tallentaa valokuvia, joissa ilmeni mahdollisimman vähän monipuolisuutta.

Luokittelijan koulutusta varten ei tallennettu lisää negatiivisia valokuvia. Luokittelijan koulutus toteutetaan hyödyntämällä luvussa 4.4.1 muodostettua 120000 negatiivisesta valokuvasta koostuvaa kuvakokoelmaa.

4.5 Luokittelijoiden koulutus

Tässä luvussa suoritetaan luokittelijoiden koulutus sekä aikaansaannosten testaus. Luokittelijoiden koulutus toteutetaan windows ympäristössä, joten koulutuksessa joudutaan käyttämään ”Cascade-Trainer-GUI” nimistä ohjelmaa. (AMIN s.a.) Ohjelma on Amin Ahmadin luoma opensource ohjelma, jonka latauslinkki löytyy lähdeluettelosta. Ohjelman ansiosta luokittelijan koulutus prosessi on hyvin suoraviivainen ja vaatii minimaalisen määrän käyttäjän sekä ohjelman välistä vuorovaikutusta. Koulutus prosessi eteni seuraavasti:

luvussa 4.4.1 luotu tietojoukko syötettiin ohjelmaan, luokittelijalle määriteltiin koulutus parametrit, koulutus käynnistettiin. Tietyn ajan kuluttua ohjelma oli luonut syötetyn tietojoukon perusteella xml tiedoston, joka kykeni suorittamaan objektintunnistuksen onnistuneesti.

Objektintunnistimen maksimi kantaman ratkaiseminen suljetussa ympäristössä on hyvin haastavaa rajoittavien tekijöiden takia, rajoittavia tekijöitä ovat esimerkiksi: huoneen pieni koko. Tästä syystä tunnistimen testauksessa joudutaan käyttämään valokuvia, jossa havaittava kohde on huomattavasti pienempi kuin oikea kohde. Testauksen yhteydessä kerättyä tietoa voidaan kuitenkin hyödyntää objektintunnistimen todellisen maksimi kantaman ratkaisussa. Objektintunnistimen testauksen yhteydessä kerättävää tietoa ovat esimerkiksi: valokuvassa ilmenevän objektin korkeus ja objektintunnistimen kantama valokuvassa ilmenevälle objektille. Kuvassa 18 ilmenee matemaattinen kaava, jota sovelletaan objektintunnistimen todellisen kantaman ratkaisussa.

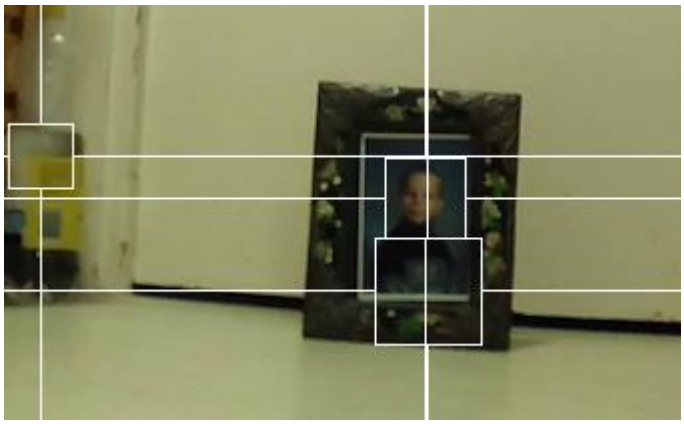
OK = Objektin korkeus
K = Kantama

$$\frac{OK\ (Valokuvassa)}{K\ (Valokuvassa)} = \frac{OK\ (Todellinen)}{K\ (Todellinen)}$$

Kuva 18. Verranto kaava

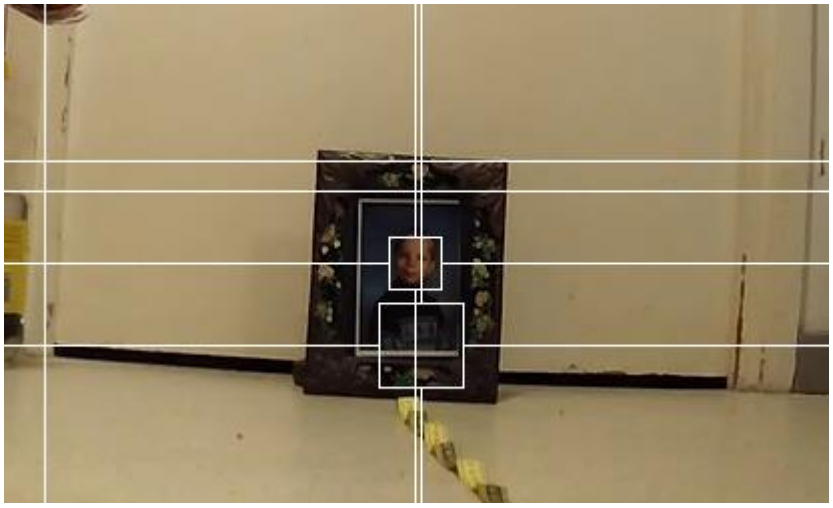
4.5.1 Koulutettujen luokittelijoiden testaus

Kuvassa 19 ilmenee ensimmäisen objektintunnistus testi, luokittelijan koulutuksessa sovellettiin LBP algoritmia. Kyseinen luokittelija koulutettiin tunnistamaan ihmisen kasvot. Koulutuksessa käytettiin 10000 valokuvaa, jotka sisälsivät kasvoja ja 25000 valokuvaa, jotka eivät sisältäneet kasvoja. Koulutuksessa käytettyjen valokuvien leveydeksi ja korkeudeksi määriteltiin 10 pixeliä. Kaikki muut koulutusparametrit jätettiin oletusasetusten mukaiseksi. Aikaansaatu luokittelija kykeni havaitsemaan 6 senttimetriä korkean kohteen 1.40 metrin etäisyydeltä. Vaikka luokittelija kykeni tunnistamaan kasvot onnistuneesti, luokittelija teki myös runsaasti virheellisiä havaintoja. Testin yhteydessä huomattiin myös, että objektin tunnistus ei tapahtunut yhtäjaksoisesti, eli välillä objektintunnistin saattoi kadottaa havaittavan kohteen väliaikaisesti.



Kuva 19. Ensimmäinen objektintunnistus testi

Kuvassa 20 ilmenee toinen objektintunnistus testi, luokittelijan koulutuksessa sovellettiin LBP algoritmia. Testin avulla pyrittiin selvittämään kuinka osumamäärä parametri vaikuttaa luokittelijan toimintaan. Tunnistimen koulutuksessa käytettiin sama määrä valokuvia kuin ensimmäisen luokittelijan koulutuksessa. Luokittelijan tarkkuutta pyrittiin parantamaan määrittelemällä ankarammat koulutusparametrit. Koulutuksesta tehtiin haastavampaa määrittelemällä uusi minimi osumamääräparametri, koulutusparametrin arvoksi asetetiin 0.999. Kaikki muut koulutusparametrit jätettiin oletusasetusten mukaiseksi. Aikaansaatu luokittelija kykeni havaitsemaan 6 senttimetriä korkean kohteen 1.66 metrin etäisyydeltä. Testin yhteydessä huomattiin, että luokittelija teki lähes yhtä paljon virheellisiä havaintoja kuin ensimmäinen luokittelija. Toisaalta luokittelija kadotti havaittavan kohteen huomattavasti harvemmin verrattuna ensimmäiseen luokittelijaan.



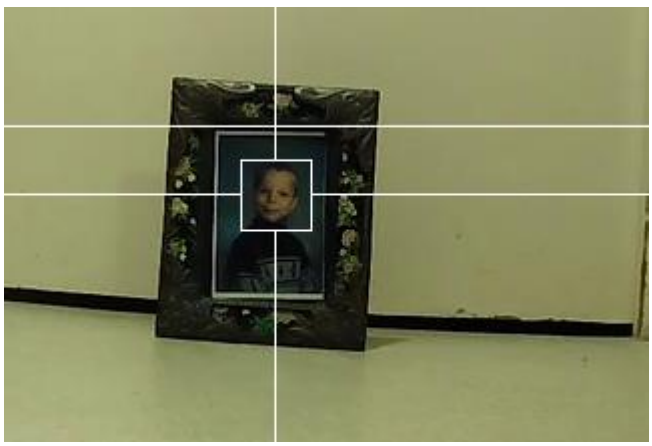
Kuva 20. Toinen objektintunnistus testi

Kuvassa 21 ilmenee kolmas objektintunnistus testi, luokittelijan koulutuksessa sovellettiin LBP algoritmia, tarkoitus oli selvittää kuinka valokuvan leveys ja korkeus parametrin kasvattaminen vaikuttaa koulutetun luokittelijan toimintaan. Luokittelijan koulutuksessa käytettiin täysin sama määrä valokuvia kuin kahden edellisen luokittelijan koulutuksessa. Lisäksi luokittelijan koulutuksessa käytettiin lähes identtisiä parametrejä kuin edellisen luokittelijan

koulutuksessa. Poikkeavia koulutusparametreja olivat valokuvien leveys ja korkeus, kyseisten parametrien arvoksi määriteltiin 25 pixeliä. Aikaansaatu luokittelija kykeni havaitsemaan 6 senttimetriä korkean kohteen 1.02 metrin etäisyydeltä. Objektintunnistimen testauksen yhteydessä huomattiin, että luokittelija teki huomattavasti vähemmän virheellisiä havaintoja kuin edelliset luokittelijat. Lisäksi objektin tunnistin kadotti havaittavan kohteen harvemmin kuin edelliset luokittelijat. Vaikka luokittelija teki vähän virheellisiä havaintoja, objektintunnistin toimi moitteettomasti vain lähietäisyydellä.

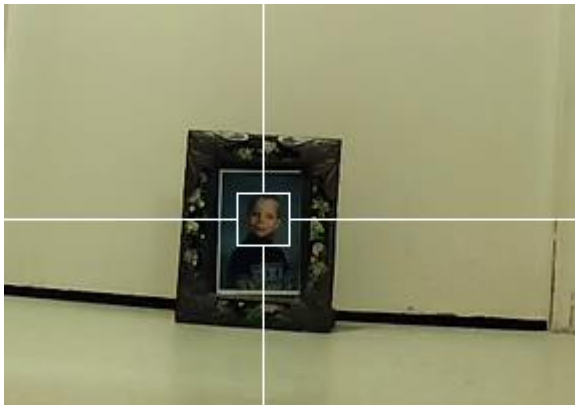
Kuva 21. Kolmas objektintunnistus testi

Kuvassa 22 ilmenee neljäs objektintunnistus testi. Luokittelijan koulutuksessa sovellettiin haar algoritmia, testin avulla pyrittiin selvittämään kuinka haar algoritmilla koulutettu luokittelija poikkeaa LBP algoritmilla koulutetusta luokittelijasta. Luokittelijan koulutuksessa käytettiin sama määrä valokuvia kuin kolmen edellisen luokittelijan koulutuksessa. Luokittelijan koulutuksen yhteydessä valokuville syötettiin uudet leveys ja korkeus parametrit, molempien parametrien arvoksi määriteltiin 15 pixeliä. Lisäksi minimi osumamäärä parametrin arvoksi määriteltiin 0.999. Kaikki muut koulutusparametrit jätettiin oletusasetusten mukaiseksi. Aikaansaatu luokittelija kykeni havaitsemaan 6 senttimetriä korkean kohteen 1.52 metrin etäisyydeltä. Haar algoritmilla koulutettu luokittelija teki paljon vähemmän virheellisiä havaintoja kuin LBP algoritmilla koulutettu luokittelija. Luokittelijan kokeilun yhteydessä huomattiin, että aikaansaannos oli huomattavasti tarkempi kuin yksikään LBP algoritmilla koulutetuista luokittelijoista. Luokittelija kykeni tunnistamaan objektin yhtäjaksoisesti, eli rajauslaatikko ei hävinnyt objektin ympäriltä.



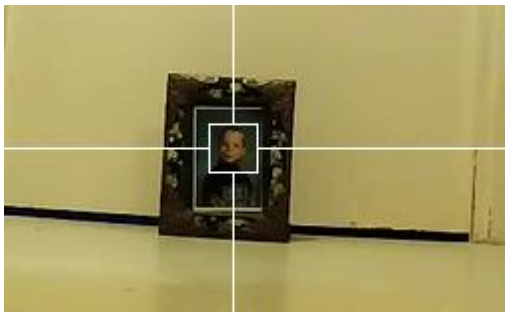
Kuva 22. Neljäs objektitunnistus testi

Kuvassa 23 ilmenee viides objektitunnistus testi. Luokittelijan koulutuksessa sovellettiin LBP algoritmia, tarkoitus oli selvittää kuinka tietojoukon kasvattaminen ja koulutusvaiheiden lisääminen vaikuttaa luokittelijan tarkkuuteen. Luokittelijan koulutuksessa käytettiin 60000 valokuvaa, jotka sisälsivät havaittavan objektin ja 170000 valokuvaa, jotka eivät sisältäneet havaittavaa objektia. Koulutuksen yhteydessä valokuville syötettiin uudet leveys ja korkeus parametrit, molempien parametrien arvoksi määriteltiin 10 pixeliä. Luokittelijan koulutuksesta tehtiin entistä haastavampaa kasvattamalla luokittelijan koulutusvaiheita, koulutusvaihe parametrin arvoksi määriteltiin 18. Osumamäärä parametrin arvo oli täysin sama kuin kolmen edellisen luokittelijan koulutuksessa, muut parametrit jätettiin oletusasetusten mukaiseksi. Aikaansaatu luokittelija kykeni havaitsemaan 6 senttimetriä korkean kohteen 2.06 metrin etäisyydeltä. Luokittelija teki hyvin vähän virheellisiä havaintoja verrattuna edelliseen testiin, virheellisiä havaintoja ilmeni noin 5 sekunnin välein. Luokittelija kykeni havaitsemaan objektin lähes yhtäjaksoisesti, välillä luokittelija saattoi kadottaa havaittavan kohteen, mutta objektin uudelleenpaikannus tapahtui lähes välittömästi.



Kuva 23. Viides objektintunnistus testi

Kuvassa 24 ilmenee kuudes objektintunnistus testi. Luokittelijan koulutuksessa sovellettiin haar algoritmia, tarkoitus oli selvittää, kuinka aikaansaatu luokittelija poikkeaa edellisten testien luokittelijoista. Mikäli luokittelijan koulutuksessa käytetty tietojoukko on yhtä laaja kuin edellisessä testissä. Luokittelijan koulutus olosuhteet olivat lähes samat kuin edellisessä testissä, koulutuksessa käytettiin täysin samaa tietojoukkoa ja täysin identtisiä koulutusparametrejä. Ainut poikkeavuus oli luokittelijan koulutus algoritmi. Aikaansaatu luokittelija kykeni havaitsemaan 6 senttimetriä korkean kohteen 2.50 metrin etäisyydeltä. Luokittelija teki huomattavasti vähemmän virheellisiä havaintoja kuin yksikään edeltäjä, olosuhteista riippuen virheellisiä havaintoja ilmeni noin 3 kertaa minuutissa. Luokittelija kykeni havaitsemaan objektin lähes yhtäjaksoisesti, mutta vain pitkältä etäisyydeltä. Lähietäisyydeltä luokittelija kadotti havaittavan objektin hyvin useasti, joskus jopa pysyvästi.



Kuva 24. Kuudes objektintunnistus testi

5 Tutkimustulokset

5.1 Testitulosten yhteenvetotaulukko

Taulukossa 1 ilmenee luvussa 4.5.1 toteutettujen testien koulutusparametrit sekä koulutuksen ja testauksen yhteydessä kartutettua tietoa. Taulukkoon kirjattua tietoa hyödynnetään yleistävien johtopäätösten tekemisessä. Ensimmäisen sarakkeen alapuolelle kirjattiin toteutettujen testien nimet. Toisen sarakkeen alapuolelle kirjattiin testien toteutuksessa käytetyt koulutusalgoritmit. Kolmannen sarakkeen alapuolelle kirjattiin koulutuksessa käytettyjen positiivisten sekä negatiivisten valokuvien lukumäärä. Neljännen sarakkeen alapuolelle kirjattiin saavutettu koulutusvaihe ja haluttu kulutusvaihe. Koulutusvaihe ilmaisee tilaa, jolloin jokaisen koulutusparametrin ehdot ovat täyttyneet. Viidennen sarakkeen alapuolelle kirjattiin valokuvien kutistettu leveys sekä korkeus pixeleinä. Kuudennen sarakkeen alapuolelle kirjattiin valokuvien leveyden sekä korkeuden käsittelyyn kohdistettu ram muisti. Seitsemännen sarakkeen alapuolelle kirjattiin osumamäärä parametrin arvo. Kahdeksannen sarakkeen alapuolelle kirjattiin lyhyt kuvailu luokittelijan toimivuuden tilasta. Yhdeksännen sarakkeen alapuolelle kirjattiin luokittelijan koulutuksen kesto. Viimeiseen sarakkeeseen kirjattiin luokittelijoiden todellinen maksimi kantama.

Nimi	Algoritmi	Kuvat (pxn)	Koulutus vaiheet	Näyte (wxh)	Ram mb (wxh)	Osumamäärä	Tila	Koulutuksen kesto	Kantama 30 cm kohde
Testi 1	LBP	10000x25000	(11/15)	10x10	11000x11000	0.995	Toimiva	9 min	7.0 metriä
Testi 2	LBP	10000x25000	(12/15)	10x10	11000x11000	0.999	Toimiva	4 min	8.3 metriä
Testi 3	LBP	10000x25000	(10/15)	25x25	11000x11000	0.999	Toimiva	14 min 7 sek	5.1 metriä
Testi 4	HAAR	10000x25000	(13/15)	15x15	11000x11000	0.999	Toimiva	29 min 26 sek	7.6 metriä
Null	HOG	10000x25000	(0/15)	10x10	11000x11000	0.999	Epäonnistunut	Null	0 metriä
Testi 5	LBP	60000x170000	(14/18)	10x10	11000x11000	0.999	Toimiva	2 h 43 min 21 sek	10.3 metriä
Testi 6	HAAR	60000x170000	(15/18)	10x10	11000x11000	0.999	Toimiva	4 h 29 min 22 sek	12.5 metriä

Taulukko 1. Testien koulutusparametrit sekä lopputulokset

5.2 Testitulosten analyysi ja tulkinta

Tutkimuksen yksi päätavoite oli selvittää, kuinka koulutusalgoritmi sekä koulutusparametrit vaikuttavat luokittelijan tarkkuuteen ja kantamaan. Visuaalisiin havaintoihin vedoten

osumamäärä parametrin kasvattamisella ei vaikuta olevan suurta vaikutusta luokittelijan tarkkuuteen. Mikäli kuvissa 19 ja 20 ilmeneviä testejä vertaillaan keskenään, osumamäärä parametrin kasvattamisen hyötyjä ei voida havaita silmämääräisesti. Vaikka kuvassa 20 ilmenevän testin koulutusparametrit olivat ankarammat kuin kuvassa 19 ilmenevän testin. Aikaansaatu luokittelija vaikutti tekevän yhtä paljon, ellei jopa enemmän virheellisiä havaintoja. Taulukkoon 1 kirjatuista tiedoista huomataan että, testissä 1 ja testissä 2 osumamäärä parametrin arvon kasvattamisella on suora vaikutus luokittelijan kantamaan. Mitä suurempi parametrin arvo on sitä pidempi luokittelijan kantama, tulee olemaan. Lisäksi havaittiin, että vaikka osumamäärä parametrin kasvattaminen lisää luokittelijan koulutusvaiheita tämä ei automaattisesti tarkoita, että luokittelijan koulutus kestäisi pidempään. Edellä mainittujen testien tuloksista huomataan, että parametrin kasvattamisen ansiosta luokittelijan koulutus tapahtui yli kaksi kertaa nopeammin.

Taulukon 1 tuloksista huomataan, että luokittelijan kantamaa voidaan kasvattaa määrittelemällä valokuville mahdollisimman pienet leveys ja korkeus koulutusparametrit. Tämä ilmenee hyvin, mikäli testien 2–3 koulutusparametrejä vertaillaan keskenään. Edellä mainittujen testien parametreistä huomataan, että valokuvien leveydellä ja korkeudella on huomattava vaikutus luokittelijan kantamaan. Mitä pienempiä leveys ja korkeus parametrit ovat sitä pidempi luokittelijan kantama tulee olemaan. Taulukkoon 1 vedoten edellä mainittujen parametrien pienentäminen lisää luokittelijan koulutusvaiheita, joka puolestaan vaikutti nopeuttavan luokittelijan koulutusta.

Visuaalisten havaintojen ja taulukon 1 tietojen perusteella huomataan, että koulutusalgoritmin valinnalla on merkittävän vaikutus luokittelijan tarkkuuteen sekä kantamaan. Koulutusalgoritmin vaikutus luokittelijan tarkkuuteen ilmenee hyvin vertailemalla kuvissa 19, 20, 21 ja 22 ilmeneviä luokittelijoita keskenään. Kuvissa 23 ja 24 ilmeneviä luokittelijoita ei voida käyttää tarkkuutta koskevien johtopäätösten tekemisessä, koska kyseisissä tapauksissa luokittelijoiden tarkkuutta on mahdotonta päätellä silmämääräisten havaintojen perusteella. Visuaalisten havaintojen perusteella huomataan, että kuvassa 22 haar algoritmilla koulutettu luokittelija on huomattavasti tarkempi kuin kuvissa 19, 20, 21 ilmenevät lbp algoritmilla koulutetut luokittelijat. Taulukkoon 1 kirjatusta tiedosta huomataan, kuinka testissä 5 ja testissä 6 käytetyt koulutusalgoritmit vaikuttavat luokittelijan kantamaan. Tulosten mukaan haar algoritmilla koulutetulla luokittelijalla on hiukan pidempi kantama kuin lbp algoritmilla koulutetulla luokittelijalla. Samalla huomataan, että luokittelijan koulutus on huomattavasti pitkäkestoisempi prosessi, mikäli koulutuksessa sovelletaan haar algoritmia.

Luokittelijan tarkkuutta ja kantamaa voidaan parantaa entisestään kasvattamalla koulutuksessa käytettävän tietojoukon kokoa, eli lisäämällä kuvia. Kuvien lukumäärän vaikutus

luokittelijan tarkkuuteen ilmenee hyvin vertailemalla kuvien 20 ja 23 sisältöä keskenään. Edellä mainittuja kuvia vertailemalla huomataan, että suuremmalla kuvamäärällä koulutettu luokittelija tekee huomattavasti vähemmän virheellisiä havaintoja. Lisäksi kuvamäärän vaikutus luokittelijan kantamaan ilmenee selkeästi, mikäli testin 2 ja testin 5 tietoja verrataan keskenään. Taulukkoon 1 kirjatuista tiedoista huomataan, että edellä mainituissa testeissä koulutukseen käytetty kuvamäärä on vaikuttanut eksponentiaalisesti luokittelijan kantamaan. Tuloksista havaitaan, että mitä suurempaa kuvamäärää luokittelijan koulutukseen käytetään sitä pidempi luokittelijan kantama, tulee olemaan. Toisaalta kuvamäärän kasvattaminen lisää luokittelijan koulutusvaiheita sekä pitkittää koulutuksen kestoa huomattavasti.

6 Pohdinta

Opinnäytetyön johdannossa määritelty tehtävä ja tavoitteet onnistuttiin saavuttamaan suunnitelman mukaisesti muutamaa seikkaa lukuun ottamatta. Tutkimuksen toteutuksen yhteydessä onnistuttiin luomaan objektintunnistin, joka tekee vähän virheellisiä havaintoja sekä kykenee havaitsemaan objekteja onnistuneesti yli kymmenen metrin etäisyydeltä. Tutkimuksen edetessä saatiin kattava käsitys objektintunnistimen tarkkuuteen ja kantamaan vaikuttavista tekijöistä. Tutkimuksen toteutuksen yhteydessä kävi myös ilme, että objektintunnistimen kantama ja tarkkuus ovat riippuvaisia lukuisten tekijöiden yhteisvaikutuksesta. Objektintunnistimen tarkkuuteen ja kantamaan vaikuttavia tekijöitä olivat: osumamäärä parametrin arvo, koulutuksessa käytettyjen valokuvien kutistetut mitat, koulutusalgoritmin valinta sekä valokuvien lukumäärä.

Opinnäytetyön tuloksena syntyi kuusi objektintunnistukseen soveltuvaa luokittelijaa, edellä mainituista luokittelijoista 4 olivat prototyyppi versioita ja loput 2 lopullisia versioita. Prototyyppien olemassaolon ainut tarkoitus oli selvittää kuinka parametrien muokkaaminen vaikuttaa luokittelijoiden toimintaan. Opittua tietoa sovellettiin erityisesti lopullisten luokittelija versioiden toteutuksessa. Opitun tiedon ansiosta lopullisten luokittelija versioiden koulutus onnistuttiin toteuttamaan 3–4 kertaa nopeammin.

Alun perin tarkoituksena oli kouluttaa luokittelijoita hyödyntämällä kolmea eri koulutusalgoritmi vaihtoehtoa, mutta luokittelijoiden koulutukseen soveltuva ohjelma ei jostain syystä kyennyt kouluttamaan luokittelijoita HOG algoritmilla. Ongelma olisi voitu ratkaista luomalla kolmas ohjelma, joka soveltuu pelkästään HOG luokittelijoiden koulutukseen, mutta tämä ratkaisu olisi pitkittänyt opinnäytetyön valmistumista huomattavasti. Tästä syystä tutkimuksessa käsitellään vain haar ja lbp algoritmeilla koulutettuja luokittelijoita. Lisäksi opinnäytetyön toteutukseen varattu aika ei riittänyt ennakkoluuloisten luokittelijoiden analyysia varten. Edellä mainittuja seikkoja lukuun ottamatta tutkimuksessa toteutuksessa ei ilmennyt muita ongelmia.

6.1 Jatkokehitys

Tutkimuksen aikaansaannoksista löytyy lukuisia jatkokehitys mahdollisuuksia. Yksi jatkokehityksen kohde voisi olla automatisaatio ratkaisuihin panostaminen esimerkiksi tietojoukon muodostuksessa. Valvontakameran PTZ asetuksien muokkauksesta huolimatta kameran staattinen sijainti vaikutti negatiivisesti valokuvien monipuolisuuteen. Valokuvien monipuolisuuteen kohdistuva ongelma voidaan ratkaista rakentamalla autonomisesti liikkuva alusta, johon valvontakamera kiinnitetään. Kyseinen alusta voidaan toteuttaa esimerkiksi hyödyntämällä mikro-ohjaimia, kuten arduino tai raspberry pi.

Tässä tutkimuksessa jokaisen luokittelijan koulutuksessa sovellettiin vain koneoppimiseen perustuvia algoritmeja. Tästä syystä aikaansaannokset ovat hyvin rajallisia, jos havaittava objekti on epäsymmetrinen, luokittelija kykenee tunnistamaan objektin vain ennalta määritellystä kuvakulmasta. Tästä syystä toinen jatkokehityksen kohde voisi olla luokittelijoiden koulutus hyödyntämällä syväoppimiseen perustuvia algoritmeja. Tarkemman tarkastelun kohteena voisi olla koneoppiin ja syväoppiin perustuvien luokittelijoiden väliset poikkeavuudet. Samalla voitaisiin analysoida syväoppimiseen perustuvien algoritmien avulla koulutetun luokittelijan tarkkuutta ja kantamaa. Joustavuuden ansiosta kyseiset algoritmit tarjoavat paremmat mahdollisuudet kehittää entistä tarkempia ja paremman kantaman omaavia objektintunnistimia.

6.2 Oman oppimisen arviointi

Tutkimuksen toteutus eteni hyvin suoraviivaisesti, vastoinkäymisiin törmättiin erityisesti teoriaosuuden toteutuksessa. Tutkimuksen aiheesta oli hyvin vaikeata löytää aloittelija ystävällistä ja helposti ymmärrettävää kirjallisuutta. Aiheeseen liittyvän kirjallisuuden läpikäyminen kulutti ylivoimaisesti eniten aikaa. Lopulta aiheen tiettyä osa-aluetta käsittelevä kirjallisuus löytyi keskustelupalstoilla esitettyjen suositusten kautta. Kirjallisuuden ansiosta opin paljon uutta konenäön toimintaan liittyen. Erityisesti mieleenpainuvaa oli koulutusalgoritmien toimintaperiaate.

Lähes kaikki työkalut, joita tutkimuksen empiirisen osion toteutuksessa käytettiin eivät olleet ennestään tuttuja. Työkaluihin tutustuminen kulutti paljon aikaa, mutta tämä vaikutti lopputulokseen positiivisesti. Empiirisen osion ohjelmointi osuudessa opittiin käyttämään Pycharm nimistä kehitysympäristöä, kyseisen kehitysympäristön ansiosta monimutkaisten toiminnallisuuksien toteutuksessa tuli huomattavasti yksinkertaisempaa. Samalla tutustuttiin lukuisiin kirjastoihin, jotka helpottivat ohjelmointi osuudessa ilmenevien toiminnallisuuksien toteutusta. Tutkimukseni todistaa, että aihetta on mahdollista tutkia vailla aikaisempaa aiheeseen perehtymistä, mikäli tekijä on määrätietoinen eli ymmärretään mitä täytyy tehdä, jotta tutkimukselle asetetut tavoitteet saavutetaan.

Tutkimuksen edetessä projektinhallinta taitojen ja ajanhallinta taitojen merkitys alkoi korostumaan. Projektin toteutus olisi ollut hyvin haastavaa ilman edellä mainittujen taitojen aikaista sisäistämistä. Projektinhallinta taidosta oli hyötyä vaatimusmäärittelyn toteutuksessa, projektin ohjelmointi osuudessa ja tuotosten testauksessa. Projektinhallinta taitojen avulla onnistuttiin välttämään kriittisiä ongelmia, jotka olisivat vaikuttaneet tutkimuksen edistymiseen negatiivisesti. Ajanhallinta taitojen ansiosta projektin eteneminen onnistuttiin pitämään projektisuunnitelmassa määritellyn aikataulun mukaisena. Viikoittain toteutettiin

sopiva määrä projektiin liittyviä tehtäviä, tarkoituksena oli välttää viikoittaisen työmäärän kasvamista liian suureksi.

Kaiken kaikkiaan olin erittäin tyytyväinen aikaansaatuun tuotokseen. Mielestäni projektin aikana toteutettu ohjelmointi osuus onnistui odotettua paremmin. Kaikki vaatimusmäärittelyssä mainitut toiminnallisuudet onnistuttiin toteuttamaan ennalta määritellyllä tavalla. Lisäksi luokittelijoiden koulutus onnistui hyvin. Alkuperäinen odotukseni oli, että luokittelijoiden koulutus olisi ollut huomattavasti pitkäkestoisempi prosessi. Myös johdannossa esitettyyn tutkimusongelmaan löydettiin selkeät vastaukset, eli saatiin kattava käsitys objektintunnistimen tarkkuuteen ja kantamaan vaikuttavista tekijöistä.

Lähteet

Ajgaonkar, A. 15.03.2021. The Value of Computer Vision: More Than Meets the Eye. Tech Journal. Luettavissa: https://www.insight.com/en_US/content-and-resources/tech-journal/spring-2021/the-value-of-computer-vision--more-than-meets-the-eye.html. Luettu: 05.12.2021.

AMIN s.a. CASCADE TRAINER GUI. Luettavissa: https://amin-ahmadi.com/cascade-trainer-gui/. Luettu: 16.12.2021.

Behera, G. 24.12.2020. Face Detection with Haar Cascade. Towards Data Science artikkeli. Luettavissa: https://towardsdatascience.com/face-detection-with-haar-cascade-727f68dafd08. Luettu: 20.10.2021.

Dorrier, J. 17.05.2020. OpenAI Finds Machine Learning Efficiency Is Outpacing Moore's Law. SingularityHub. Luettavissa: https://singularityhub.com/2020/05/17/openai-finds-machine-learning-efficiency-is-outpacing-moores-law/. Luettu: 31.10.2021.

Escrivá, D. & Laganiere, R. 2019. OpenCV 4 Computer Vision Application Programming Cookbook: Build complex computer vision applications with OpenCV and C++. Packt Publishing. E-kirja. Luettu: 21.10.2021.

Fisher, R., Breckon, T., Dawson-Howe, K., Fitzgibbon, A., Robertson, C., Trucco, E. & Williams, C. 2014. Dictionary of Computer Vision and Image Processing. John Wiley & Sons, Incorporated. Hoboken. E-kirja. Luettu: 24.10.2021.

FRITZ LABS INCORPORATED s.a. OBJECT DETECTION GUIDE. Luettavissa: https://www.fritz.ai/object-detection/. Luettu: 04.12.2021.

Harmouch, M. 19.06.2020. Local Binary Pattern Algorithm: The Math Behind It. Medium blogi. Luettavissa: https://medium.com/swlh/local-binary-pattern-algorithm-the-math-behind-it-%EF%B8%8F-edf7b0e1c8b3. Luettu: 21.10.2021.

Howse, J. & Minichino, J. 2020. Learning OpenCV 4 Computer Vision with Python 3: Get to grips with tools, techniques, and algorithms for computer vision and machine learning. Packt Publishing, Limited. E-kirja. Luettu: 22.10.2021.

JuliaImages s.a. Local Binary Patterns. Luettavissa: https://juliaimages.org/stable/examples/image_features/lbp/. Luettu: 31.10.2021.

Kapur, S. & Thakkar, N. 2015. Mastering OpenCV Android applications programming: master the art of implementing computer vision algorithms on Android platforms to build robust and efficient applications. Packt Publishing. E-kirja. Luettu: 20.10.2021.

Koivunen, V. 2021. Opinnaytetyo-konenako. Luettavissa: https://github.com/a1800572/Opinnaytetyo-konenako. Luettu: 07.12.2021.

López, A., Imiya, A., Pajdla, T. & Álvarez, J. 2017. Computer Vision in Vehicle Technology: Land, Sea and Air. John Wiley & Sons, Incorporated. United Kingdom. E-kirja. Luettu: 12.10.2021.

Mohamed, E. 2020. Deep Learning for Vision Systems. Manning Publications. Shelter Island, NY. E-kirja. Luettu: 18.10.2021.

OpenCV s.a. Face Detection using Haar Cascades. Luettavissa: https://docs.opencv.org/4.x/d2/d99/tutorial_js_face_detection.html. Luettu: 31.10.2021.

Puiu, T. 16.12.2019. AI is outpacing Moore's Law. ZME SCIENCE. Luettavissa: https://www.zmescience.com/science/ai-is-outpacing-moores-law/. Luettu: 30.10.2021.

Reben, A. 2019. 1 million fake faces. Luettavissa: https://archive.org/details/1mFakeFaces. Luettu: 07.12.2021.

Research and Markets 2021. AI in Computer Vision Market. Luettavissa: https://www.researchandmarkets.com/reports/5350928/ai-in-computer-vision-market-with-covid-19-impact?utm_source=CI&utm_medium=PressRelease&utm_code=c8g8mq&utm_campaign=1553448+-+The+Worldwide+AI+in+Computer+Vision+Industry+is+Expected+to+Reach+%2451.3+Billion+by+2026+at+a+CAGR+of+26.3%25+from+2021&utm_exec=jamu273prd. Luettu: 04.12.2021.

Schramm, L. 2017. Innovation Technology: A Dictionary. De Gruyter. Berlin. E-kirja. Luettu: 26.10.2021.

scikit-image s.a. Histogram of Oriented Gradients. Luettavissa: https://scikit-image.org/docs/dev/auto_examples/features_detection/plot_hog.html?highlight=hog. Luettu: 31.10.2021.

Singh, H. 16.03.2021. How Images are stored in the computer?. Analytics Vidhya blogi. Luettavissa: https://www.analyticsvidhya.com/blog/2021/03/grayscale-and-rgb-format-for-storing-images/. Luettu: 21.10.2021.

Szeliski, R. 2021. Computer Vision: Algorithms and Applications 2nd Edition. Springer. Luettavissa: https://www.dropbox.com/s/8bf4feleifhrvl6/Szeliski-BookDraft_20210930.pdf?dl=0. Luettu: 8.10.2021.

Tyagi, M. 04.07.2021. HOG (Histogram of Oriented Gradients): An Overview. Towards Data Science artikkeli. Luettavissa: https://towardsdatascience.com/hog-histogram-of-oriented-gradients-67ecd887675f. Luettu: 21.10.2021.

Liitteet

Liite 1. Liitteen nimi

Liite 2. Liitteen nimi