Marika Könönen

Koiran tunnistaminen digitaalisesta kuvasta

Hahmontunnistussovellus KOIRAKO

Metropolia Ammattikorkeakoulu
Insinööri (AMK)
Tietotekniikka
Insinöörityö
4.6.2012

Metropolia

Tiivistelmä

Tekijä Otsikko Sivumäärä Aika	Marika Könönen Koiran tunnistaminen digitaalisesta kuvasta Hahmontunnistussovellus KOIRAKO 97 sivua + 14 liitettä 4.6.2012
Tutkinto	Insinööri (AMK)
Koulutusohjelma	Tietotekniikka
Suuntautumisvaihtoehto	Ohjelmistotekniikka
Ohjaajat	Lehtori Timo-Pekka Jäntti Lehtori Simo Silander

Tämän insinöörityön tavoitteena oli määritellä, suunnitella ja toteuttaa hahmontunnistussovellus KOIRAKO, joka tunnistaa ja paikantaa koiran digitaalisesta kuvasta. Hahmontunnistusmenetelmää valittaessa haluttiin ensisijaisesti löytää toimiva sekä kohteen tunnistamiseen ja paikantamiseen kykenevä sovellus, joka soveltuu muunneltavaksi KOIRAKOn tarpeisiin. Lisäksi valintaa rajoittivat ohjelmointikieli, käyttöjärjestelmä ja kehitysympäristö.

Insinöörityöaiheen tutkiminen aloitettiin perehtymällä hahmontunnistuksen ja koneoppimisen käsitteisiin tavoitteena selvittää olemassa olevien menetelmien ja algoritmien käytettävyyttä KOIRAKOssa. Tutkimus jatkui selvittämällä luokitteluongelman vaiheita sekä syventymällä muun muassa opetusvaiheeseen sisältyvään piirteenirrotukseen, siihen liittyviin algoritmeihin sekä oppimisen mallintamiseen. Työhön liittyvän teorian opiskelun ohessa määriteltiin ja suunniteltiin KOIRAKO, joka sopivan menetelmän löydyttyä toteutettiin.

Koiraan tunnistamiseen ja paikantamiseen valittiin VOC-sovelluksen versio 3.1, jonka diskriminatiivinen opetusvaihe on toteutettu piilomuuttujia hyödyntävällä tukivektorikonealgoritmilla. Tunnistusvaiheessa käytetään yleistettyä etäisyysfunktiota ja dynaamista ohjelmointia. Koodikirjaa opetusvaiheessaan käyttävät menetelmät, kuten Implicit Shape Model ja Bag of Visual Words, olivat tutkimuksen kohteena ennen VOCiin päätymistä. Niistä ei kuitenkaan löytynyt KOIRAKOssa hyödynnettäväksi soveltuvaa toteutusta. Piirteenirrotuksen algoritmeja tutkittiin syvemmin: Toisena kokonaisuutena oli paikallisten avainpisteiden perusteella SIFT-algoritmilla muodostetut piirrekuvaajat, ja toisena VOCin tarkoitukseen muokattu HOG-algoritmilla muodostettu globaali VOC-piirrekuvaaja.

KOIRAKO toteutettiin MATLABissa toimivana työasemasovelluksena. Se ohjelmoitiin oliopohjaisesti MATLAB-kielellä ja sen graafinen käyttöliittymä rakennettiin Swing- ja AWT-kirjastojen komponenteista. VOC-sovellus sisältää koiran tunnistamiseen ja paikantamiseen liittyvät funktiot, joita KOIRAKO käyttää tunnistusprosessissa.

Avainsanat	hahmontunnistus, koneoppiminen, piirteenirrotus, koodikirja, avainpiste, piirrevektori, diskriminanttifunktio, diskriminatiivinen todennäköisyysmalli, SIFT, HOG, ISM, BoVW

Metropolia

Abstract

Author	Marika Könönen
Title	Recognition and Localization of Dog from Digital Image Pattern Recognition Application KOIRAKO
Number of Pages	97 pages + 14 appendices
Date	4 June 2012
Degree	Bachelor of Engineering
Degree Programme	Information and Communications Technology
Specialisation option	Software Engineering
Instructors	Timo-Pekka Jäntti, Senior Lecturer Simo Silander, Senior Lecturer

The purpose of this thesis was to design and implement a pattern recognition application called KOIRAKO, which recognizes and localizes a dog possibly present in a digital image. The main research problem was to find an adequate pattern recognition method for the KOIRAKO application. The main criterion for its selection was an existing functional implementation of the method, adaptable to the needs of KOIRAKO. The selection was also constrained by programming language, operating system and development environment.

The research was started by the basics of pattern recognition and machine learning. It proceeded with exploring the phases of the classification problem and studying existing algorithms for feature extraction. The implementation of the KOIRAKO began alongside the studying of the theories related to the research problem.

The discriminatively trained part-based VOC method and its implementation were selected as the basis for the KOIRAKO application. Comparison between other pattern recognition methods was made based on the training phase. Initially, the attention fell upon methods like Implicit Shape Model and Bag of Visual Words, which utilize codebook representation, but they were not usable for KOIRAKO. The VOC model uses XML annotations, which are used in the training phase instead of a codebook.

KOIRAKO was implemented in a MATLAB environment by using MATLAB language in an object-oriented way and the user interface was constructed from Java Swing components. The VOC implementation contains the functions related to object recognition and localization, which were used in the recognition process of KOIRAKO.

Keywords	pattern recognition, machine learning, codebook, feature extraction, interest point, descriptor, discriminant function, probabilistic discriminative model, SIFT, HOG, ISM, BoVW

Sisällys

Liitteet

Lyhenteet ja määritelmät

Abstrakti hahmo	*Abstract pattern.* Kohteella on hahmo ja se on aina olemassa ja kuuluu johonkin kohdeluokkaan, vaikka siitä ei olisi tehty yhtään havaintoa. Hahmoa voidaankin kutsua abstraktiksi, koska sitä ei voida suoraan mitata eikä myöskään esittää lukuarvoin. Mikäli se olisi saatavilla, niin sen avulla tunnistusongelma ratkaistaisiin parhaalla mahdollisella tavalla. Katso luku 3.1.2.
Analysoitava kuva	Käyttäjän valitsema kuva, joka annetaan syötteenä KOIRAKO-sovellukselle.
Avainpiste	*Keypoint, interest point.* Avainpisteet ovat kuvan pisteitä, jotka ovat muuttumattomia skaalan ja kuvakulman vaihtuessa. Katso luvut 6.1.1 ja 6.1.2.
AWT	*Abstract Window Toolkit.* Javassa julkaistu käyttöliittymäkomponenttikirjasto.
BoVW	*Bag of Visual Words.* Hahmontunnistusmenetelmä, joka käyttää koodikirjaa. Katso luku 5.1.1.
Diskriminatiivinen todennäköisyysmalli	*Discriminative probability model.* Bayesin päätösteoriaa hyödyntävä oppimisen menetelmä. Katso luku 4.2.2.
DoG	*Difference of Gaussians.* Avainpisteitä hyödyntävä avainpisteidenhavaitsija-algoritmi. Katso luku 6.1.2.
Empiirinen piirre	Analysoituja ja käsiteltyjä piirteitä kutsutaan empiirisiksi piirteiksi ja niiden löytämiseksi sovellettavaa menettelyä kutsutaan piirteenirrottamiseksi. Empiirisistä piirteistä muodostetaan yleensä vektori eli piirrevektori, jota käyte-

	tään luokittelu- ja päätöksentekomenetelmissä. Katso luku 3.1.2.
Hahmo	*Pattern.* Niemannin ja Holmströmin mukaan hahmon on kohteesta mitattua tietoa ja se vaatii kohteen havaitsemisen. Katso luku 3.1.2.
Hahmontunnistus	*Pattern recognition.* Tieteenala, joka käsittelee kohteen tunnistamista, luokittelemista tai mallintamista kohteesta tehtyjen havaintojen perusteella. Katso luku 3.
Havaintoikkuna	*Detection window.* Suorakulmion muotoinen ikkuna, joka sisältää kuvassa olevan kohteen. Havaintoikkunaa käytetään HOG-algoritmissa. Katso luku 6.2.
HOG	*Histogram of Oriented Gradients.* HOG-algoritmi laskee kohteesta 36-ulotteiset HOG-piirteet. Katso luku 6.2.
HOG-piirre	*HOG feature.* HOG-algoritmilla kohteesta laskettu piirrevektori. Katso luku 6.2
ISM	*Implicit shape model.* Hahmontunnistusmenetelmä, joka käyttää koodikirjaa. Katso luku 5.1.1
Juurisuodatin	*Root filter.* VOC-sovelluksen hahmontunnistuskehykseen kuuluva komponentti eli suodatin joka peittää koko kohteen. Katso luku 5.2.1.
Komponenttimalli	*Mixture model.* VOC-menetelmän opetusvaiheessa käytetty malli, joka huomioi kohteen kuvakulmat. Jokaista kuvakulmaa vastaa yksi komponentti, jotka opetetaan erikseen. Opetusvaiheen lopussa komponentit yhdistetään yhdeksi malliksi. Katso luku 5.2.1.

Koneoppiminen	*Machine learning.* Tieteenala, joka tutkii tietokoneohjelmien kykyä tehdä kokemusten perusteella päätelmiä hahmoista, säännönmukaisuuksista tai säännöistä. Katso luku 4.
Konstellaatiomalli	*Constellation model.* Hahmontunnistusmenetelmä, joka hyödyntää koodikirjaa.
Kuvapyramidi	*Image pyramid.* Tietorakenne, joka sisältää kopioita alkuperäisestä kuvasta eli k-määrän kuvatasoja. Kuvapyramidissa kuvan pikselit ja resoluutio vähenevät asteittain ja symmetrisesti siirryttäessä kuvatasolta toiselle. Alkuperäinen kuva esitetään pyramidin juuressa ja siitä seuraavat kuvatasot tuotetaan rekursiivisesti antamalla edellisen tason kuva seuraavalle tasolle syötteenä. Katso luku 5.2.1.
LSVM	*Latent Support Vector Machine.* Piilomuuttujia käyttävä tukivektorikone. Katso SVM ja luku 5.2.2.
MATLAB	*Matrix Laboratory.* MATLAB on vektori- ja matriisilaskennan kehitysympäristö sekä siinä käytettävä ohjelmointikieli, joka mahdollistaa nopean numeerisen laskennan tieteellisten ja teknisten laskennan tehtäviin. [MATLAB – The Language Of Technical Computing.]
Nimiöinti	*Labelling.* Havaintoon liittyvän kohdeluokan merkitseminen, joka voidaan toteuttaa esimerkiksi erillisenä XML-tiedostona. Katso XML-annotaatio ja luku 4.
Ohjaamaton oppiminen	*Unsupervised learning.* Koneoppimista, joka perustuu nimiöimättömän opetusaineiston käyttämiseen. Katso luku 4.
Ohjattu oppiminen	*Supervised learning.* Koneoppimista, jonka opetusvaiheessa hyödynnetään valmiiksi nimiöityä opetusmateriaalia. Katso luku 4.

Opetusaineisto	*Training data.* Opetusvaiheessa käytettäviä kuvia, joista tehdään havaintoja (muodostetaan piirrevektoreita). Opetusmateriaali jaetaan positiivisiin (sisältää kohteen) ja negatiivisiin (ei sisällä kohdetta) kuviin. Katso luku 4.
Opetusvaihe	*Training phase.* Koneoppijan opettamisvaihe, johon kuuluu havaita esitetystä tietoaineistosta ominaispiirteitä ja rakentaa niiden pohjalta malli, jota käytetään päätöksentekoon tunnistusvaiheessa. Katso luvut 4 ja 5.2.2.
Osasuodattimet	*Part filters.* VOC-sovelluksen hahmontunnistuskehykseen kuuluva komponentti eli suodattimet, jotka peittävät kohteen osat, vertaa juurisuodattimeen. Katso luku 5.2.1.
PCA	*Principal Component Analysis*. Pääkomponenttianalyysia käytetään muun muassa kuvan ulottuvuuden alentamiseen. [Jolliffe 2002; Shlens 2009.] Katso luku 6.2.4.
Piirre	*Feature.* Kohteesta mitattu ominaisuus: Sama piirre voi olla useassa kohteessa, mutta piirteen numeerinen arvo vaihtelee kohteesta riippuen ja kohteet luokitellaan juuri näihin numeerisiin arvoihin perustuen. Vertaa empiirinen piirre. Katso luku 3.1.2. Sanaa piirre käytetään tässä insinöörityössä myös liitteenä piirteenirrotusmenetelmän nimen yhteydessä, esimerkiksi HOG-piirre ja VOC-piirre, luku 6.2.
Piirrekartta	*Feature map.* Matriisi, jonka alkiot ovat d-ulotteisia piirrevektoreita. Katso luku 5.2.1.
Piirrekuvaaja	*Feature descriptor.* Katso piirrevektori.
Piirrepyramidi	*Feature pyramid.* Koostuu kuvapyramidin kuvista lasketuista piirrekartoista. Katso luku 5.2.1.

Piirrevektori	*Feature vector.* Empiirisistä piirteistä muodostetaan yleensä vektori eli piirrevektori, jota käytetään luokittelu- ja päätöksentekomenetelmissä. Se on laskettu opetus- ja testiaineistosta piirteenirrotusmenetelmällä (esimerkiksi HOG tai SIFT). Katso empiirinen piirre ja luku 3.1.2. Piirrevektorista käytetään tässä insinöörityössä myös nimeä piirrekuvaaja (SIFT ja VOC katso luku 6).
Pistearvo	*Score.* Juurisuodattimille ja osasuodattimille lasketaan pistearvo jokaisessa kuvan pisteessä sekä kuvapyramidin skaaloissa. Katso luku 5.2.
Piirteenirrotus	*Feature extraction.* Menetelmä, jolla lasketaan piirrevektorit opetus- ja testiaineistosta. Katso luku 6.
Pisteytysfunktio	*Scores.* Pistearvot kerätään yhteen pisteytysfunktiolla, jota käytetään niin opetus- kuin tunnistusvaiheessa. Katso luku 5.2.
SIFT	*Scale Invariant Feature Transformation.* Piirteenirrotusmenetelmä, jossa piirrekuvaaja muodostetaan avainpisteidenhavaitsija-algoritmilla DoG ja piirrekuvaajat muodostetaan SIFT-algoritmilla. Katso luku 6.1.
SVM	*Support Vector Machine.* Tukivektorikone Katso luku 5.2.2.
Testiaineisto	*Test data.* Tunnistusvaiheessa käytettäviä kuvia, joista tehdään havaintoja (muodostetaan piirrevektoreita). Katso luku 4.
Tunnistuskehys	*Bounding box.* Tunnistuskehys on suorakulmion muotoinen kehys, joka rajaa kohteen kuvasta. Sen tarkoituksena on ilmoittaa, että kohde on tunnistettu ja paikannettu. Katso esimerkki luvusta 7.2.

Tunnistusmalli	*Model.* Hahmontunnistusmenetelmän opetusvaiheessa luoma malli, jota käytetään tunnistusvaiheessa. Katso luku 4.
Tunnistustulos	Kuva, jonka KOIRAKO-sovellus näyttää tunnistusvaiheen päätyttyä. Onnistuneen tunnistuksen tulos sisältää kohteen rajaavan tunnistuskehyksen.
Tunnistusvaihe	Koostuu testiaineiston piirteenirrotuksesta ja tunnistusprosessista.
UML	*Unified Modeling Language.* Ohjelmistosuunnittelussa käytettävä standardisoitu kokoelma kuvaustekniikoita, jota on kehitetty erityisesti oliopohjaisten järjestelmien rakenteen ja toiminnan kuvaamiseen [Object Management Group–UML].
Vahvistusoppiminen	*Reinforcement learning.* Koneoppimisen laji, jonka algoritmit perustuvat oppimiseen yrityksen ja erehdyksen kautta, missä kehotetaan eniten positiivista palautetta tuottavaan toimintaan. Katso luku 4.
VOC-menetelmä	VOC-menetelmän avulla tunnistetaan kohteita kuvasta. Se on kohteen osiin perustuva komponenttimalli, joka huomioi kohteen kuvakulmat ja se käyttää diskriminatiivista opetusvaihetta.
VOC-piirre	VOC-piirre on 31-ulotteinen HOG-piirteestä VOC-menetelmällä muokattu piirre, jota käytetään Felzenszwalbin VOC-nimisessä hahmontunnistusmenetelmässä. Katso luku 6.2.
VOC-piirrekuvaaja	VOC-piirrekuvaaja koostuu VOC-piirteistä. Katso luku 6.2.

VOC-sovellus	Felzenszwalbin [Felzenszwalb ym. 2009] toteuttama hahmontunnistussovellus, jota käytetään KOIRAKOssa. Perustuu VOC-hahmontunnistusmenetelmään, josta kertoo artikkeli Discriminatively Trained Part Based Models [Felzenszwalb ym. 2010]. Katso luku 5.2.
XML	*Extensible Markup Language.* XML on tiedon merkityksen kuvaamiseen tarkoitettu merkintäkieli.
XML-annotaatio	*XML-annotation.* Esimerkiksi VOC-menetelmässä kuvien nimiöinti (katso nimiöinti) on toteutettu XML-tiedostoina. Katso luku 5.2.

1 Johdanto

Tämän insinöörityön aiheena on koiran tunnistaminen digitaalisesta kuvasta. Tavoitteena on määritellä, suunnitella ja toteuttaa graafisen käyttöliittymän sisältävä hahmontunnistussovellus KOIRAKO. Se saa syötteenä käyttäjän valitseman digitaalisen kuvan ja palauttaa käyttöliittymään tunnistustuloksen kuvana. Löytäessään annetusta kuvasta kohteen, eli tässä tapauksessa koiran, sovellus piirtää sen ympärille tunnistuskehyksen.

Työn aihe on lähtöisin yöllä pohdiskellusta vilkasliikenteistä helsinkiläistä koirapuistoa koskevasta testitilanteesta. Siinä koirapuiston porttia kuvataan yhtäjaksoisesti vuorokauden ajan, ja tallennettu videokuva annetaan syötteenä tietokonesovellukselle, jonka on tarkoitus laskea sekä puistossa käyneiden koirien että kyseisten koirien edustamien koirarotujen lukumäärä. Tällainen sovellus on kuitenkin liian laaja tässä insinöörityössä toteutettavaksi. Siksi työ on rajattu käsittelemään vain koiran tunnistamista kuvasta, mikä itsessäänkin on haasteellinen päämäärä. Aiheeseen perehtyminen alkoi digitaalisen kuvankäsittelyn perusteista sekä MATLAB-ohjelman käytön alkeista, edeten hahmontunnistukseen ja sen menetelmiin.

Hahmontunnistusmenetelmiin perehtymisen haasteena tässä insinöörityössä on löytää menetelmä, joka koiran tunnistamisen ja paikantamisen lisäksi soveltuu KOIRAKO-sovelluksen sovelluslogiikan pohjaksi. Koska menetelmiin liittyvien algoritmien ohjelmoiminen alusta loppuun asti itse veisi tutkimusongelman laajuuden vuoksi liian pitkään, on hahmontunnistusmenetelmän valitsemisen ensimmäisenä kriteerinä valmiin (ja toimivan) toteutuksen olemassa olo. Sen lisäksi tulee tarkastella toteutusten teknisiä ominaisuuksia ja käytettävyysmahdollisuuksia: Millä ohjelmointikielellä toteutus on ohjelmoitu? Minkä käyttöjärjestelmän se vaatii? Missä kehitysympäristössä sitä voidaan käyttää? Voiko siihen liittää Javan Swing-luokkia tai voiko sitä kutsua Java-kielisestä ohjelmakoodista?

Tämä insinöörityöraportti alkaa KOIRAKO-sovelluksen määrittelyllä (luku 2) ja hahmontunnistuksen (luku 3) sekä koneoppimisen (luku 4) perusteilla ja jatkuu tutkittuihin menetelmiin (luku 5) tutustumisella. Niiden jälkeen syvennytään hahmontunnistusjärjestelmässä käytettyihin algoritmeihin (luku 6). Sitten esitellään KOIRAKO-sovelluksen toteutus (luku 7) ja luodaan siltä pohjalta katsaus olio-ohjelmointiin MATLABilla (luku 8). Viimeisessä luvussa 9 kootaan yhteen insinöörityön ongelmat ja tulokset.

2 KOIRAKO-sovelluksen määrittely

Tässä luvussa esitellään KOIRAKO-sovelluksen määrittely sisältäen järjestelmävaatimusmäärittelyn (luku 2.1), sovellussanaston (luku 2.2), käyttötapaukset (luku 2.3) ja käyttöliittymäprototyypin (luku 2.4). KOIRAKO-sovelluksen toteutusta ei käsitellä tässä luvussa vaan hahmontunnistuksen menetelmiin liittyvän teorian jälkeen luvussa 7.

2.1 Järjestelmävaatimusmäärittely

Järjestelmän vaatimusten määrittely aloitetaan toiminnallisien vaatimuksien tarkastelulla, minkä jälkeen perehdytään ei-toiminnallisiin vaatimuksiin.

2.1.1 Toiminnalliset vaatimukset

Kehitettävän KOIRAKO-sovelluksen tulee näyttää, onko käyttäjän antamassa syötteessä eli kuvassa koira. Käyttäjä valitsee analysoitavan kuvan lisäksi tunnistuksessa käytettävän tunnistusmallin, jota myös havainnollistetaan sovelluksessa kuvana. KOIRAKO-sovellus kommunikoi hahmontunnistussovelluksen kanssa ja palauttaa tunnistustuloksen kuvana käyttäjälle. Tunnistusulos on pohjimmiltaan sama kuin analysoitava kuva, mutta mikäli siitä on löytynyt koira, on tunnistustuloksena palautettavaan kuvaan piirretty tunnistuskehys koiran ympärille. Käyttäjä voi halutessaan tallentaa tunnistustuloksen.

Sidosryhmiksi luetaan sovellusta hyödyntävät käyttäjät ja hahmontunnistussovellus VOC (luku 7.2), jota KOIRAKO tarvitsee tunnistusvaiheessa.

2.1.2 Ei-toiminnalliset vaatimukset

Sovellus toteutetaan MATLAB-sovelluksen alaisuudessa toimivana työasema-sovelluksena. KOIRAKO on VOCista eriytetty sovellus, vaikka se käyttääkin VOCia koiran tunnistukseen. Ohjelmointikielinä on MATLAB ja Java. MATLABia sovelletaan olioperusteisesti ja käyttöliittymäkomponentit luodaan Javalla.

2.2 Sovellussanasto

Hahmontunnistussovellus
Tunnistusvaiheen käsittelevä sovellus, joka saa syötteekseen sekä tunnistettavan kuvan että tunnistusmallin ja palauttaa tunnistustuloksen.

Analysoitava kuva
Käyttäjän valitsema kuva, joka annetaan syötteenä sovellukselle.

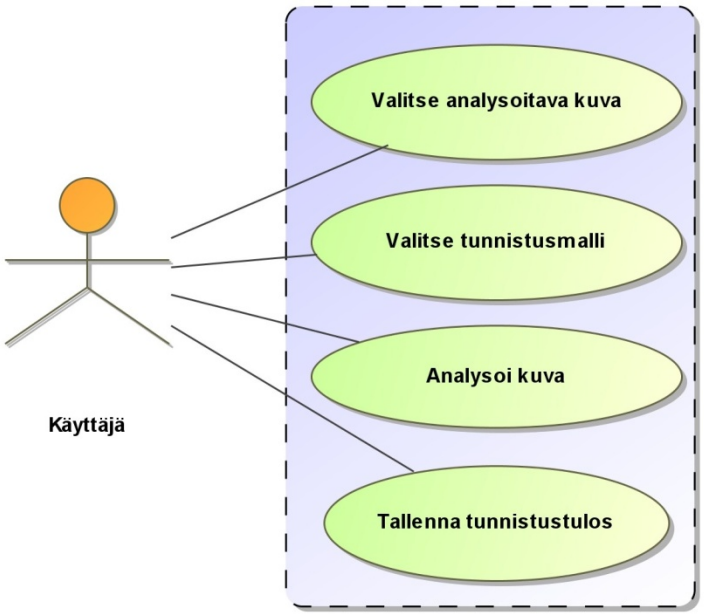
Tunnistusmalli
Hahmontunnistusmenetelmän (katso luku 5) rakentama malli, jota käytetään tunnistusvaiheessa (katso luku 4).

Tunnistuskehys
Tunnistuskehys on suorakulmionmuotoinen kohdetta rajaava kehys. Sen tarkoituksena on ilmoittaa, että kohde on tunnistettu ja paikannettu. Tässä insinöörityössä rajojen sisäpuolella tulisi onnistuneessa tunnistustapauksessa olla koira.

Tunnistustulos
Kuva, jonka sovellus näyttää tunnistusvaiheen päätyttyä. Onnistuneen tunnistuksen tulos sisältää kohteen rajaavan tunnistuskehyksen.

2.3 Käyttötapaukset

Käyttötapaukset on esitelty sanallisesti taulukoissa 1–4 ja niitä havainnollistavat sekvenssikaaviot kuvissa 2–5.



Kuva 1. KOIRAKO-sovelluksen käyttötapauskaavio

KOIRAKOn toiminnallisuutta on mallinnettu neljällä järjestelmätason käyttötapauksella (kuva 1): Valitse analysoitava kuva (taulukko 1 ja kuva 2), Valitse tunnistusmalli (taulukko 2 ja kuva 3), Analysoi kuva (taulukko 3 ja kuva 4) ja Tallenna tunnistustulos (taulukko 4 ja kuva 5).

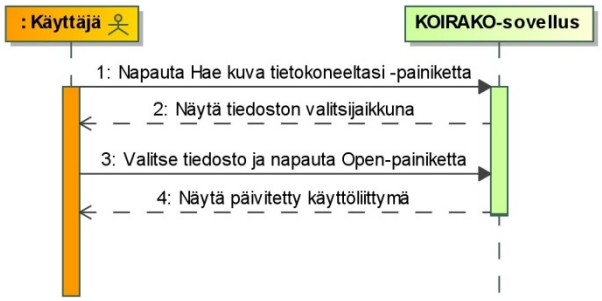
2.3.1 Käyttötapaus K1: Valitse analysoitava kuva

Taulukko 1 sisältää sanallisen kuvauksen käyttötapaukselle Valitse analysoitava kuva.

Taulukko 1. Sanallinen kuvaus käyttötapaukselle Valitse analysoitava kuva (K1).

Käyttötapauksen ID:	K1–Valitse analysoitava kuva
Tarkoitus:	Käyttäjä valitsee tietokoneeltaan kuvan, jota käytetään tunnistus-prosessissa.
Toimijat:	Käyttäjä
Toiminnot:	Käyttäjä napauttaa Hae kuva tietokoneeltasi -painiketta. Sovellus avaa tiedostonvalintaikkunan. Käyttäjä valitsee tiedoston ja napauttaa Open-painiketta. Sovellus päivittää valitun kuvan käyttöliittymään.
Esivaatimukset:	Käyttäjä on käynnistänyt sovelluksen.
Poikkeukset:	Käyttäjä valitsee tiedoston, joka ei ole kuva.
Jälkiehdot:	Sovellus on päivittänyt valitun kuvan käyttöliittymään.
Viimeksi muokattu:	21.5.2011

Kuva 2 esittää graafisesti taulukossa 1 kirjatun kuvauksen käyttötapaukselle Valitse analysoitava kuva.



Kuva 2. Valitse analysoitava kuva (K1) käyttötapausta kuvaava sekvenssikaavio.

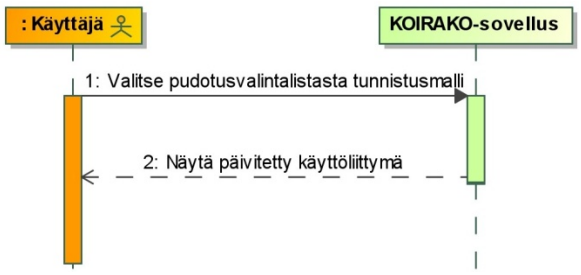
2.3.2 Käyttötapaus K2: Valitse tunnistusmalli

Taulukko 2 sisältää sanallisen kuvauksen käyttötapaukselle Valitse tunnistusmalli.

Taulukko 2. Sanallinen kuvaus käyttötapaukselle Valitse tunnistusmalli (K2).

Käyttötapauksen ID:	K2–Valitse tunnistusmalli
Tarkoitus:	Käyttäjä valitsee koiran tunnistamisessa käytettävän tunnistusmallin.
Toimijat:	Käyttäjä
Toiminnot:	Käyttäjä valitsee pudotusvalintalistasta tunnistusmallin. Sovellus päivittää mallin kuvan käyttöliittymään.
Esivaatimukset:	Käyttäjä on käynnistänyt sovelluksen.
Poikkeukset:	
Jälkiehdot:	Sovellus on päivittänyt käyttäjän valitseman tunnistusmallin kuvan käyttöliittymään.
Viimeksi muokattu:	21.5.2011

Kuva 3 esittää graafisesti taulukossa 2 kirjatun kuvauksen käyttötapaukselle Valitse tunnistusmalli.



Kuva 3. Valitse tunnistusmalli (K2) käyttötapausta kuvaava sekvenssikaavio.

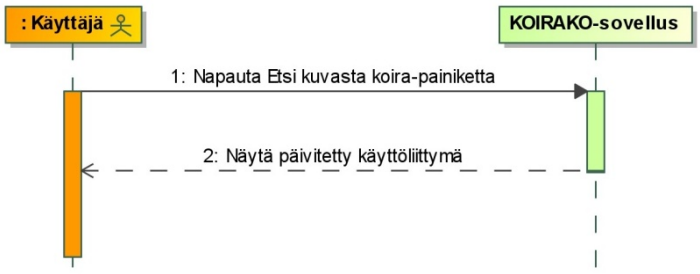
2.3.3 Käyttötapaus K3: Analysoi kuva

Taulukko 3 sisältää sanallisen kuvauksen käyttötapaukselle Analysoi kuva.

Taulukko 3. Sanallinen kuvaus käyttötapaukselle Analysoi kuva (K3).

Käyttötapauksen ID:	K3–Analysoi kuva
Tarkoitus:	Käyttäjä käynnistää tunnistamistoiminnon.
Toimijat:	Käyttäjä
Toiminnot:	Käyttäjä napauttaa Etsi kuvasta koira -painiketta. Sovellus päivittää tunnistustuloskuvan käyttöliittymään.
Esivaatimukset:	Käyttäjä on käynnistänyt sovelluksen ja valinnut tunnistettavan kuvan sekä tunnistusmallin.
Poikkeukset:	Käyttäjä ei ole valinnut kuvaa tai tunnistusmallia.
Jälkiehdot:	Sovellus on päivittänyt käyttöliittymään tunnistustuloskuvan.
Viimeksi muokattu:	21.5.2011

Kuva 4 esittää graafisesti taulukossa 3 kirjatun kuvauksen käyttötapaukselle Analysoi kuva.



Kuva 4. Analysoi kuva (K3) käyttötapausta kuvaava sekvenssikaavio.

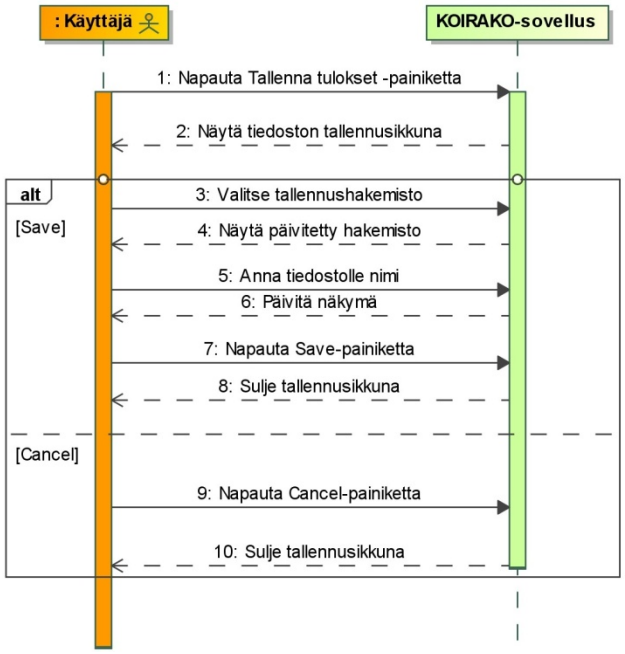
2.3.4 Käyttötapaus K4: Tallenna tunnistustulos

Taulukko 4 sisältää sanallisen kuvauksen käyttötapaukselle Tallenna tunnistustulos.

Taulukko 4. Sanallinen kuvaus käyttötapaukselle Tallenna tunnistustulos (K4).

Käyttötapauksen ID:	K4–Tallenna tunnistustulos
Tarkoitus:	Käyttäjä tallentaa tunnistustuloksen.
Toimijat:	Käyttäjä
Toiminnot:	Käyttäjä napauttaa Tallenna tulokset -painiketta. Sovellus avaa hakemistonvalintaikkunan. Käyttäjä valitsee hakemiston, jonne haluaa tallentaa kuvan, antaa nimen tallennettavalle tiedostolle ja napauttaa Save-painiketta. Sovellus tallentaa kuvan kyseiseen hakemistoon.
Esivaatimukset:	Käyttäjä on käynnistänyt sovelluksen ja vallinnut tallennettavan kuvan sekä tunnistusmallin.
Poikkeukset:	Kuvan tunnistustapahtumaa ei ole suoritettu. Käyttäjä ei haluakaan tallentaa kuvaa ja painaa Cancel-painiketta. Sovellus sulkee hakemistonvalintaikkunan.
Jälkiehdot:	Sovellus on tallentanut tunnistustuloksen käyttäjän valitsemaan hakemistoon, käyttäjän antamalla nimellä.
Viimeksi muokattu:	21.5.2011

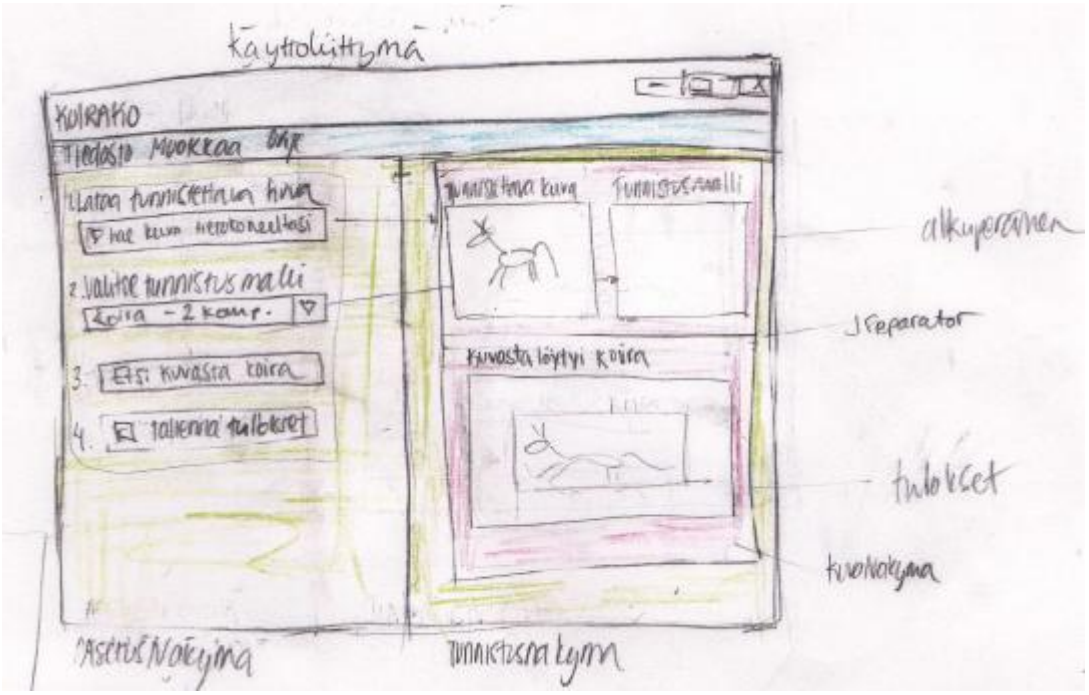
Kuva 5 esittää graafisesti taulukossa 4 kirjatun kuvauksen käyttötapaukselle Tallenna tunnistustulos.



Kuva 5. Tallenna tunnistustulos (K4) käyttötapausta kuvaava sekvenssikaavio.

2.4 Käyttöliittymäprototyyppi

KOIRAKO-sovelluksen käyttöliittymäprototyyppi (kuva 6) sisältää yhden ikkunan, joka koostuu asetusnäkymästä ja tunnistusnäkymästä. Käyttöliittymä on yksinkertainen, joten ei ollut tarpeen suunnitella sitä ohjelmistolla.



Kuva 6. KOIRAKO-sovelluksen käyttöliittymäprototyyppi.

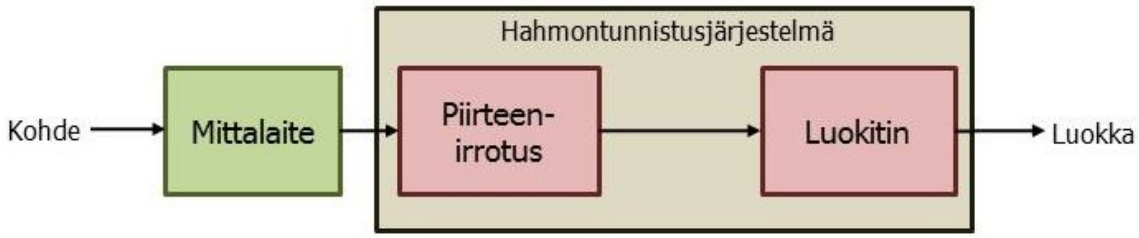
Asetusnäkymä sisältää analysoitavan kuvan valitsemiseen tietokoneelta (K1) tarkoitetun komponentin, tunnistusmallin, joka valitaan pudotusvalintalistasta (K2) ja lisäksi painikkeet käyttötapausten Analysoi kuva (K3) ja Tallenna tunnistustulos (K4) toteuttamiseen. Tunnistusnäkymään päivittyvät asetusnäkymästä suoritettujen toimintojen tulokset: analysoitava kuva, tunnistusmalli ja tunnistustulos.

3 Hahmontunnistus

Hahmontunnistus *(pattern recognition)* on kohteen tunnistamista, luokittelemista *(classification)* tai mallintamista kohteesta tehtyjen havaintojen *(observations)* perusteella. Se toteutetaan soveltamalla muun muassa signaalinkäsittelyn, neurolaskennan ja tilastotieteen menetelmiä mitattuun tietoaineistoon, kuten esimerkiksi kuvaan, puheeseen tai tekstiin. Hahmontunnistuksen sovellusalueisiin kuuluvat muun muassa lääketiede, robotiikka, tietoliikenne, teollisuusautomaatio sekä ihmisen ja koneen vuorovaikutus. Pullonpalautusautomaatti ja roskapostin automaattinen tunnistaminen ovat arkipäiväisiä esimerkkejä hahmontunnistusjärjestelmistä. Tässä luvussa käsitellään ensin hahmontunnistusjärjestelmää yleisemmällä tasolla, minkä jälkeen perehdytään hahmontunnistuksen käsitteistöön tarkemmin. [Mitä on hahmontunnistus?; Tohka 2009: 1.]

3.1.1 Hahmontunnistusjärjestelmä

Hahmontunnistusjärjestelmät koostuvat useimmiten seuraavista vaiheista: mittaus, esikäsittely *(pre-processing)*, segmentointi *(segmentation)*, piirteenirrotus *(feature extraction)*, luokitteleminen ja jälkikäsittely. Järjestelmän päämääränä on tunnistaa kohde ja sijoittaa se mitatun tietoaineiston perusteella oikeaan luokkaan. Esikäsittelyvaiheeksi tai piirteenirrotukseksi saatetaan kutsua myös kaikkia luokittelua ennen mainittuja vaiheita yhteensä [Bishop 2006: 2]. Kuva 7 esittää hahmontunnistusjärjestelmää ja sen osia. [Tohka 2009: 3.]



Kuva 7. Esimerkki hahmontunnistusjärjestelmän osista [Holmström 2010: 4].

Tunnistusprosessi alkaa tietoaineiston mittaamisella luokiteltavasta kohteesta eli tässä työssä koiraa esittävästä digitaalisesta kuvasta, josta mitataan tunnistuksessa käytettävä tietoaineisto. Kuva on esikäsiteltävä, mikäli se sisältää esimerkiksi kohinaa tai sitä pitää muuten muokata seuraavaa vaihetta varten. Esikäsittelymenetelmät ja niiden käyttämisen tarpeellisuus riippuvat käytettävästä tunnistusmenetelmästä. Esimerkiksi joissakin tässä työssä tutkituissa menetelmissä (luku 5.1) koira pitää rajata taustasta pois eli segmentoida, kun taas toisissa (luku 5.2) piirteenirrotus tehdään niin sanotusta luonnollisesta eli esikäsittelemättömästä kuvasta.

Mittausta ja mahdollista esikäsittelyä seuraa piirteenirrotusvaihe, jonka tavoitteena on etsiä kohteen luokituksessa käytettävät piirteet *(features)* eli piirrevektorit *(feature vectors)* ja parantaa luokittelua vähentämällä sen kannalta epäolennaisen tiedon määrää [Holmström 2010: 3]. Jäljempänä tässä työssä (esimerkiksi luvussa 6) piirrevektoria kutsutaan piirrekuvaajaksi, joka kuvastaa paremmin siinä yhteydessä esitettyä asiaa. Hahmontunnistuksen perusteisiin tutustuessa on kuitenkin hyvä käyttää yleisesti tunnettua termiä piirrevektori. Piirrevektorin avulla luokitin sijoittaa kohteen sopivimpaan luokkaan ja jälkikäsittelyssä perehdytään luokittimien tuloksiin. [Tohka 2009: 3–5.]

3.1.2 Hahmo, piirteet ja piirrevektori

Tässä työssä on koettu yllättävän haastavaksi hahmontunnistuksen peruskäsitteiden määritteleminen, sillä yhtä termiä saatetaan käyttää useassa eri yhteydessä, näkemykset lähteiden välillä vaihtelevat ja lisäksi tarkkoja määritelmiä on hankala löytää. Verhagen toteaa artikkelissaan [1975: 109], jonka tarkoitus on koota yhteen kirjallisuudessa esiintyviä hahmontunnistukseen liittyviä käsitteiden määritelmiä, että termeille ei löydy identtisiä määritelmiä, vaan kulloinkin käytettävät määritelmät riippuvat

kirjoittajasta. Esimerkiksi sana hahmo *(pattern)* on määritelty monella eri tavalla, toiset ovat määritelleet sen yleisemmin ja toiset täsmällisesti matematiikan keinoin [Verhagen 1975: 114].

Tässä luvussa esiteltyjen käsitteiden lähteinä on käytetty Holmströmin [2010] luentomateriaalia, Timo-Pekka Jäntin [2010–2011] pohdintoja ja Heinrich Niemannin kirjaa Klassifikation von Mustern [2003].

Niemann [2003: 12–13] määrittelee täsmällisesti muun muassa hahmon, siihen liittyvän ympäristön sekä ongelmakehyksen. Tässä työssä ei kuitenkaan ole syytä syventyä hänen tulkintoihinsa perin pohjin. Siitä huolimatta tarkastelemme hänen määrittelemäänsä hahmokäsitettä. Havaintojen käsittelyä varten esitetään ympäristö U fysikaalisesti mitattavilla suureilla. Ympäristöä U vastaa siis mitattavien suureiden tai funktioiden joukko

$$U = \left\{ {}^{\rho}f(x) \middle| \rho = 1,2, \dots \right\} . \tag{1}$$

Käsiteltävän ongelman aihepiiriin kuuluvat kyseiseen sovellusalueeseen liittyvät kohteet tai funktiot, joita kutsutaan ongelmakehykseksi Ω. Sen alkiot eli funktiot muodostavat hahmon, joten hahmo on siis mittausfunktio,

$$f(x) = \begin{pmatrix} f_1(x_1, \cdots, x_n) \\ f_2(x_1, \cdots, x_n) \\ \vdots \\ f_m(x_1, \cdots, x_n) \end{pmatrix} . \tag{2}$$

Lasse Holmström esittää tilastollisen hahmontunnistuksen luentomateriaalissaan [2010: 1–4] käsitteen hahmo olevan esimerkiksi $m \times n$ pikselin kokoinen kuva, josta etsitään piirteet ja muodostetaan piirrevektorit. Kohteen mittaus siis vastaa hahmoa, jonka hahmovektori syötetään hahmontunnistusjärjestelmälle ja tuloksena saadaan luokka, johon hahmo kuuluu. Niemannin ja Holmströmin näkemykset hahmon esitystavasta vastaavat paljon toisiaan, sillä kummatkin mainitsevat sen olevan kohteesta mitattua tietoa.

Seuraavaksi tutustutaan Timo-Pekka Jäntin [2010–2011] näkemykseen hahmosta ja piirteistä. Hänen mukaansa voidaan ajatella, että kohteella on hahmo ja että kaikki kohteet, joiden hahmot ovat samanlaisia, kuuluvat samaan kohdeluokkaan. Hahmo on

siis aina olemassa ja kuuluu johonkin kohdeluokkaan, vaikka siitä ei olisi tehty yhtään havaintoa. Niemannin ja Holmströmin esitykset eroavat tuolta osin, sillä heidän määritelmiensä mukainen hahmo on mittaus ja täten vaatii kohteen havaitsemisen. Jäntin mielestä hahmo sisältää kaiken tarpeellisen tiedon kohteen luokittelemiseksi ja tunnistamiseksi. Hahmoa voidaankin kutsua abstraktiksi, koska sitä ei voida suoraan mitata eikä myöskään esittää lukuarvoin. Mikäli se olisi saatavilla, niin sen avulla tunnistusongelma ratkaistaisiin parhaalla mahdollisella tavalla.

Piirteet ovat kohteesta mitattuja ominaisuuksia. Yksi piirre ei yleensä riitä luokittelemaan ja tunnistamaan kohdetta. Sen sijaan tavoitteena on koota tyhjentävä piirrejoukko, mikä luokittelun kannalta vastaa tietosisällöltään edellä mainittua kohteen hahmoa. Tyhjentävyydellä (lisätietoa lähteessä [Luku 9, Uskottavuuden ominaisuuksia: 263]) tarkoitetaan havaintoihin sisältyvän tiedon määrittelyä kvantitatiivisesti eli määrällisesti, toisin sanoen parametrin θ estimaatti (katso lähde [Estimaatti]) $T(x)$ on tyhjentävä, jos se sisältää kaiken havaintoja θ:aa koskevan tietosisällön. Oleellista on ymmärtää, että sama piirre voi olla useassa kohteessa, mutta piirteen numeerinen arvo vaihtelee kohteesta riippuen ja kohteet luokitellaan juuri näihin numeerisiin arvoihin perustuen. Kohteesta siis mitataan esimerkiksi avainpisteitä (luku 5.1.2) sekä niiden ympäristöjä ja mittauksille tehdään mahdollisesti monimutkaisiakin laskutoimituksia, joista piirteille saadaan numeeriset arvot. Näitä analysoituja ja käsiteltyjä piirteitä kutsutaan empiirisiksi piirteiksi ja niiden löytämiseksi sovellettavaa menettelyä kutsutaan piirteenirrottamiseksi. Empiirisistä piirteistä muodostetaan yleensä vektori eli piirrevektori, jota käytetään luokittelu- ja päätöksentekomenetelmissä.

Piirteenirrotuksessa lasketut empiiriset piirteet eli piirrevektori x muodostavat kaikki yhdessä piirreavaruuden *(feature space)* $\mathbb{F}$, joten voimme sanoa, että $x \in \mathbb{F}$ eli piirrevektori x kuuluu piirreavaruuteen $\mathbb{F}$ [Tohka 2009: 21]. Luokittelun tarkoituksena on jakaa piirreavaruus $\mathbb{F}$ luokkia vastaaviin alueisiin ja muun muassa sitä käsitellään seuraavassa luvussa. Lisäksi perehdytään oppimisen mallintamiseen ja menetelmiin.

4 Koneoppiminen

Koneoppiminen *(machine learning)* on tieteenala, joka tutkii tietokoneohjelmien kykyä tehdä kokemusten perusteella päätelmiä hahmoista, säännönmukaisuuksista tai säännöistä. Alalla ollaan kiinnostuneita siitä, kuinka rakennetaan kokemuksista automaattisesti oppivia ohjelmia. Tom Mitchell kiteyttää kirjassaan Machine Learning [Mitchell 1997] hyvin asetetun *(well-posed)* oppimisongelman määritelmän:

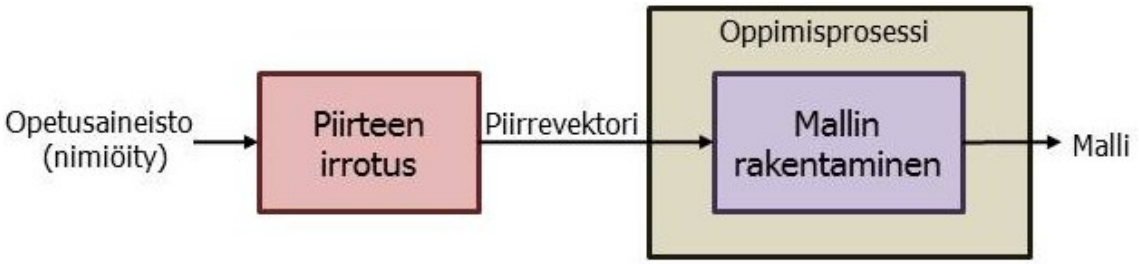
> A computer program is said to *learn* from experience E with respect to some task T and some performance measure P, if its performance on T, as measured by P, improves with experience E [Mitchell 1997: 2].

Edellä mainituilla kokemuksilla tarkoitetaan tutkittavaan oppimisongelmaan liittyvästä aineistosta tehtyjä mittauksia, joita käytetään ilmiön käyttäytymisen ennustamiseen. Koneoppiminen perustuu muun muassa todennäköisyys- ja päätösteoriaan *(probability and decision theory)*, joiden perusteella koneoppija *(learning machine)* tekee optimaalisia päätöksiä [Bishop 2006: 38].

Koneoppimisprosessiin kuuluu seuraavat vaiheet [Kääriäinen 2008a: 14]:

- tavallisen ongelman muotoilu oppimisongelmaksi
- oppimisongelmaan liittyvän tiedon kerääminen
- oppimismenetelmän valinta ja suunnittelu
- koneoppijan opettaminen – opetusvaihe *(training phase, learning phase)*
- koneoppijan testaaminen – tunnistusvaihe.

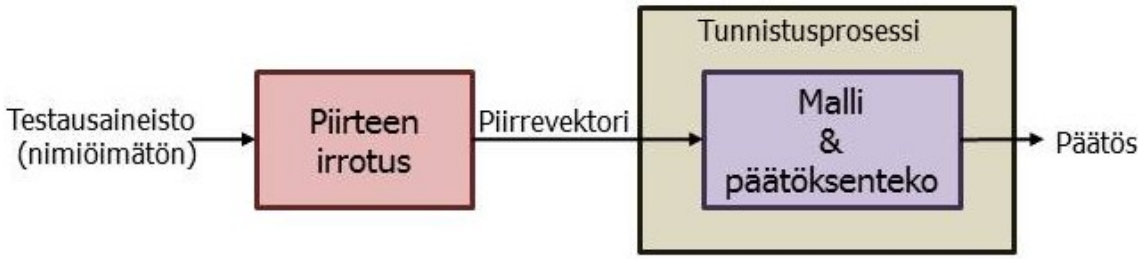
Listassa esiintyvällä koneoppijalla tarkoitetaan oppivaa ohjelmaa. Sen tehtävänä on opetusvaiheessa havaita esitetystä tietoaineistosta ominaispiirteitä ja rakentaa niiden pohjalta malli, jota käytetään päätöksentekoon tunnistusvaiheessa. Jatkossa tässä insinöörityöraportissa käytetään käsitteitä opetusvaihe ja tunnistusvaihe, joista ensimmäisellä (kuva 8) tarkoitetaan koneoppijan opettamista, jälkimmäisen (kuva 9) viitatessa luokittelutehtävän suorittamiseen niin sanottua valmiiksi opetettua koneoppijaa hyödyntäen.



Kuva 8. Opetusvaiheessa rakennetaan malli opetusaineiston perusteella. Opetusaineisto koostuu nimiöidyistä positiivisista ja negatiivisista opetuskuvista, joista muodostetaan piirrevektorit. Ne annetaan syötteenä koneoppijalle, joka muodostaa oppimisprosessin aikana mallin. Lähdettä [Jäntti 2011] mukaillen.

Koneoppija opetetaan ja testataan toisistaan eroavilla opetus- ja testausmateriaaleilla *(training and test data)*. Mitä opetus- ja testausmateriaalit käytännössä ovat? Tässä insinöörityössä ne ovat kuvia, tai tarkemmin sanottuna kuvista muodostettuja piirrevektoreita (katso luku 6). On tärkeää muistaa, että molempia opetus- ja testiaineistoa käsitellään samalla tavalla esikäsittelyvaiheesta piirteenirrotukseen asti.

Opetusmateriaali jaetaan positiivisiin ja negatiivisiin aineistoihin, joista negatiiviset eivät sisällä kohdetta, kun taas positiiviset sisältävät [Jäntti 2011]. Tämä mainittakoon siitä syystä, että opetusvaiheessa piirrevektorit muodostetaan samalla menetelmällä koko opetusaineistosta, sisälsi se kohdetta tai ei. Lisäksi opetusaineisto saattaa olla nimiöity *(labelled)* riippuen siitä, mitä koneoppimisen algoritmia käytetään. Nimiöinti tarkoittaa sitä, että havaintoihin liittyvät kohdeluokat ovat merkattu yksitellen ja se on tehty käsin. Merkitseminen voidaan toteuttaa esimerkiksi erillisenä XML-tiedostona, jota kutsutaan tässä insinöörityöraportissa XML-annotaatioksi (luku 5.2). Nimiöinnin helpottamiseksi on olemassa myös tarkoitukseen kehitettyjä sovellusohjelmia, joista esimerkkinä mainittakoon internetissä vapaasti käytettävissä oleva LabelMe [Russell ym. 2008]. [Bishop 2006: 2–3.]



Kuva 9. Nimiöimättömästä testiaineistosta muodostetaan tunnistusvaiheessa piirrevektorit, joita käytetään tunnistusprosessissa. Siinä tehdään opetusvaiheessa luodun mallin avulla päätös testiaineiston luokittelusta. Lähdettä [Jäntti 2011] mukaillen.

Tarkastellaan seuraavaksi joitakin koneoppimiseen liittyviä algoritmeja. Suurin osa näistä algoritmeista voidaan niiden lopputuloksen perusteella luokitella kolmeen luokkaan: ohjattuun oppimiseen *(supervised learning)*, ohjaamattomaan oppimiseen *(unsupervised learning)* tai vahvistusoppimiseen *(reinforcement learning)* [Bishop 2006: 3]. On olemassa myös näihin luokkiin kuulumattomia algoritmeja, mutta niiden tarkastelu ei niiden harvinaisuuden vuoksi kuulu tämän insinöörityön laajuuteen.

Ohjatulla oppimisella tarkoitetaan valmiiksi nimiöityä opetusmateriaalia (sisältäen $(havainto, kohdeluokka)$-pareja) käyttäviä algoritmeja, kun taas ohjaamattoman oppimisen algoritmien opetusmateriaalia ei ole nimiöity, vaan se sisältää vain havainnot. Vahvistusoppiminen perustuu niin sanotusti oppimiseen yrityksen ja erehdyksen kautta. Siinä ei käytetä $(havainto, kohdeluokka)$-pareja, kuten ohjatun oppimisen luokitteluongelmassa, vaan kehotetaan eniten positiivista palautetta tuottavaan toimintaan. Ohjaamattoman oppimisen algoritmeihin kuuluvaan klusteroinnin *(clustering)* tavoitteena on jakaa samankaltaiset opetusaineiston alkiot luokkiin eli klustereihin. Koska tämä insinöörityö kuitenkin keskittyy vain ohjattuun oppimiseen, niin ohjaamattoman oppimisen ja vahvistusoppimisen tarkempi tarkastelu sivuutetaan. [Bishop 2006: 3.]

Luokittelu *(classification)* ja regressio *(regression)* ovat ohjatun oppimisen algoritmeja, joissa valmiiksi nimiöidyn opetusaineiston avulla pyritään luomaan yleistys *(generalization)* tutkittavan ilmiön toimintatavasta ja sitä kautta tekemään päätös uusien havaintojen kohdeluokasta tai -arvosta. Ne eroavat toisistaan siten, että luokittelun lopputulos on diskreetti muuttuja, kun taas regressiossa se koostuu yhdestä tai useammasta jatkuva-arvoisesta muuttujasta. Esimerkkinä luokittelumenetelmän soveltamisesta olkoon tämän insinöörityön päämäärä, eli syötteenä saadun koiraa esittävän kuvan

sijoittaminen johonkin annetuista luokista. Toisaalta, regressiomenetelmässä koneoppijan tarkoituksena voisi olla vaikkapa asunnon hinnan ennustaminen syötteenä annettujen tietojen (esimerkiksi asunnon pinta-ala sekä huoneiden lukumäärä ja sijainti) perusteella. [Bishop 2006: 3, Kääriäinen 2008a: 10.]

4.1 Oppimisen mallintaminen

Ohjatun oppimisen luokitteluongelmassa koneoppijan on tarkoitus oppia päättämään opetusaineiston perusteella, mihin luokkaan tuntematon havainto *(input, instance, observation, pattern)* kuuluu [Kääriäinen 2008a: 8]. Olkoon esimerkkitapauksena koiran tunnistaminen kuvasta, jossa tehdään päätös kuuluuko havainto luokkaan ”kuvassa on koira” vai ”kuvassa ei ole koiraa”. Sille annetaan syötteenä havainto eli kuva, tarkemmin sanottuna piirrevektori ja se tekee päätöksen [Schölkopf & Smola 2002: 1]

$$f(x) = \begin{cases} 1, & jos\ on\ koira \\ -1, & jos\ ei\ ole\ koira\ . \end{cases} \tag{3}$$

Määritellään edellisen kappaleen käsitteet tarkemmin. Havainnot sisältyvät joukkoon $\mathcal{X}$, ja havaintoihin liittyvät kohdeluokat *(labels, targets, outputs)*, niin sanotut oikeat tulokset, sisältyvät joukkoon $\mathcal{Y} = \{-1,1\}$. Päätöksentekofunktio *(decision function)* $f(x)$ palauttaa arvonaan joukon $\mathcal{Y}$ alkion (-1 tai 1) sen mukaan, mihin luokkaan annettu havainto kuuluu (katso kaavaa 3). Edellä mainitunlaista luokittelua kutsutaan binääriseksi luokitteluksi *(binary classification)*, jolla tarkoitetaan luokittelun perustumista tietyn ominaisuuden olemassaoloon, havainnolla joko on se, tai sitten ei [Schölkopf & Smola 2002: 3]. Kohdeluokkia binäärisessä luokittelussa on siis kaksi kappaletta.

Opetusaineiston joukko

$$S = \{(x_i, y_i) |\ i = 1, \dots, n\} \tag{4}$$

sisältää $(havainto, kohdeluokka)$ -pareja siten, että

$$(x_i, y_i) \in \mathcal{X} \times \mathcal{Y}\ . \tag{5}$$

Kaavoissa 4 ja 5 esiintyvä x_i vastaa siis havaintoa ja y_i kohdeluokkaa, eli jokaiselle havainnolle määritellään kyseistä havaintoa vastaava kohdeluokka. Koneoppijan tekemien sallittujen päätösten joukkoa merkitään

$$\mathcal{Y}' = \{f(x_i)|i = 1, \dots, n\} \ . \tag{6}$$

$\mathcal{Y}'$ (kaava 6) sisältää koneoppijan havainnoille $x_i \in \mathcal{X}$ tekemät oikeat kohdeluokkapäätökset. Koneoppija saa lopullisen muotonsa opetusvaiheen päätyttyä [Bishop 2006: 2]. Sen kuvaus on $f: \mathcal{X} \to \mathcal{Y}'$ ja se antaa päätöksen $f(x) \in \mathcal{Y}'$. [Kääriäinen 2008a: 20, 24.]

Satunnaismuuttujan (X, Y) todennäköisyysjakauma $P_{X,Y}$

Satunnaismuuttujat X ja Y ovat kuvauksia perusjoukon eli alkeistapauksien Ω joukolta reaalilukujen joukolle eli $\mathrm{X}: \Omega \to \mathbb{R}$ ja $Y: \Omega \to \mathbb{R}$. Satunnaismuuttuja (X, Y) on määritelty todennäköisyysavaruudessa $(\Omega, \mathcal{F}, P)$, jossa Ω on perusjoukko, jonka alkiot ovat alkeistapauksia ω, $\mathcal{F}$ on tapahtumien joukkokokoelma ja P on todennäköisyys [Jäntti 2012]. Tässä insinöörityöraportissa ei tarkastella tarkemmin todennäköisyyslaskennan käsitteitä todennäköisyyskenttä *(probability space)*, alkeistapaus, satunnaismuuttuja *(random variable)*, todennäköisyysjakauma *(probability distribution)* ja kertymäfunktio *(cumulative distribution function)*. Lisätietoa käsitteistä voi lukea muun muassa Ilkka Mellinin [2010a; 2010b] luentomateriaaleista.

X ja Y ovat satunnaismuuttujia ja niiden järjestetty pari (X, Y) määrittelee kaksiulotteisen satunnaismuuttujan $(X, Y): \mathcal{X} \times \mathcal{Y} \to \mathbb{R}^2$, jonka karteesisen tulon muodostama arvojoukko on $\mathcal{X} \times \mathcal{Y} = \{x, y | x \in \mathcal{X}, y \in \mathcal{Y}\}$, jossa $\mathcal{Y} = \{-1, 1\}$ kuten jo edellä mainittiin [Mellin 2010c: 5]. Satunnaismuuttujan (X, Y) todennäköisyysjakaumaa $P_{X,Y}$ ja kertymäfunktiota $F_{X,Y}$ ei tunneta. Opetusaineisto S on otos tästä jakaumasta $P_{X,Y}$ [Kääriäinen 2008a: 26].

Mielenkiintoista on se, että koneoppijalle annettavat havainnot sekä sen tekemät sallitut päätökset eli oikein luokiteltujen kohdeluokkien arvot ovat myös peräisin jakaumasta $P_{X,Y}$. Sanotaan, että tietoaineisto (sisältäen opetus- ja testiaineiston) on tuotettu tästä tuntemattomasta ja muuttumattomasta satunnaismuuttujan jakaumasta $P_{X,Y}$, mikä on yleinen olettamus tilastollisessa oppimisteoriassa *(statistical learning theory)* [Schölkopf & Smola 2002: 8]. Lisäksi tällä tavalla luotu tietoaineisto todetaan riippu-

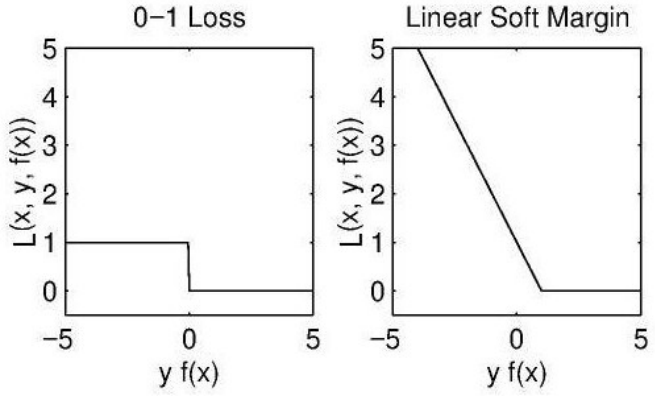
mattomaksi ja samoin jakautuneeksi *(independent and identically distributed* eli *i.i.d.*). [Kääriäinen 2008a: 26–27; Schölkopf & Smola 2002: 8, 65.]

4.1.1 Kustannusfunktio

Oppimiseen tarvitaan myös kustannusfunktio *(loss function)*, joka määrittelee päätöksen aiheuttaman kustannuksen. Sillä mitataan luokittelun oikeellisuutta siten, että väärästä päätöksestä tulee suurempi tappio kuin oikeasta. Kustannusfunktion L kuvaus on $L: \mathcal{X} \times \mathcal{Y} \times \mathcal{Y}' \to \mathbb{R}^+$ ja binäärisen luokittelun yksinkertaisin kustannusfunktio on muotoa

$$L_{0/1}(x, y, f(x)) = \begin{cases} 0, & jos\ y = f(x) \\ 1, & muuten\ . \end{cases} \tag{7}$$

Väärästä päätöksestä funktio (kaava 7) saa arvokseen 1 ja siitä kärsitään tappio, oikea päätös tuottaa arvon 0, kuten kuvan 10 vasemmanpuoleinen kuvaaja havainnollistaa. [Kääriäinen 2008a: 23; Schölkopf & Smola 2002: 8, 62.]



Kuva 10. Kustannusfunktiotyyppien 0–1 ja hinge loss -kuvaajat. Lähdettä [Schölkopf & Smola 2002: 64] mukaillen.

Kustannusfunktioita on useita eri tarkoituksiin. Yksi tämän insinöörityön kannalta tärkeä on pehmeän marginaalin kustannusfunktio *(soft margin loss function*, kutsutaan myös nimellä *hinge loss)*,

$$L_{hinge}(x, y, f(x)) = max\big(0{,}1 - yf(x)\big) = \begin{cases} 0, & jos\ \ yf(x) \geq 1 \\ 1 - yf(x), & jos\ yf(x) < 1 \end{cases}, \quad (8)$$

jota käytetään luvun 5.2 VOC-menetelmässä, tarkemmin sanottuna tukivektorikoneessa *(support vector machine* eli *SVM)* [Schölkopf & Smola 2002: 63]. Kuvan 10 hinge loss -kuvaajasta on nähtävissä tukivektorikoneen marginaalialueen (katso luku 5.2.2) vaikutus kärsittävään tappioon, sillä $L\big(x, y, f(x)\big)$ kasvaa lineaarisesti, kun $yf(x) \leq 1$. Marginaalialueella tehdyt päätökset sijaitsevat arvovälillä $-1 < yf(x) < 1$ ja kustannuksen käyttäytyminen sillä arvovälillä tapahtuu seuraavasti: Koneoppijan tekemästä oikeasta päätöksestä ehdolla $yf(x) \geq 1$ ei tule tappiota, mutta oikeasta päätöksestä arvovälillä $0 < yf(x) < 1$ kärsitään tappio, ja mitä lähempänä päätös on lukua 0, niin sitä suurempi on kustannuskin. Arvovälillä $-1 < yf(x) < 0$ olevat päätökset ovat vääriä ja niistä maksetaan kustannus. Sen sijaan 0–1-kustannusfunktio (kuva 10 vasemmalla) antaa kaikista vääristä päätöksistä yhtä suuren tappion, kun $yf(x) < 0$, eikä oikeista päätöksistä sakoteta lainkaan. [Jäntti 2011; Schölkopf & Smola 2002: 62–63.]

4.1.2 Empiirisen riskin minimointi

Oppimisprosessin päämääränä on löytää riskin *(risk)* minimoiva funktio. Päätökseen f liittyvä riski R esitetään kustannusfunktion L (luku 4.1.1) odotusarvona $E[L]$ ja se on muotoa [Schölkopf & Smola 2002: 66]

$$R(f) = E\big[L\big(x, y, f(x)\big)\big] = \int_{X \times Y} L\big(x, y, f(x)\big) dP_{X,Y}\,. \quad (9)$$

Odotusarvo tarkoittaa satunnaismuuttujien X ja Y arvojen painotettua keskiarvoa ja painokertoimet saadaan kaavan 9 tapauksessa todennäköisyysjakaumasta $P_{X,Y}$ eli kustannusfunktion odotusarvo kertoo keskimääräisen kustannuksen yhtä päätöstapausta $(x, y, f(x))$ kohden [Mellin 2010d: 7, 10–12]. Schölkopf ja Smola toteavat [2002: 65], että täytyy löytää menetelmä, jolla yhdistetään paikalliset kustannukset. Toisin sanoen luvun 4.1.1 kustannusfunktio käsittelee yhtä päätöstapausta $(x, y, f(x))$ kerral-

laan, ja tässä luvussa määriteltävä empiirinen riski yhdistää niin sanotut erilliset päätöstapaukset.

Riskin määritelmä (kaava 9) sisältää aikaisemmin tässä luvussa esitellyn satunnaismuuttujan (X, Y) tuntemattoman todennäköisyysjakauman $P_{X,Y}$. Sovelluksissa jakaumaa on hankala käsitellä, joten sen sijaan käytetään opetusaineistosta (x_i, y_i) laskettavaa empiiristä todennäköisyystiheyttä $p_{(X,Y)emp}$ [Jäntti 2012; Schölkopf & Smola 2002: 66]. Seuraavaksi määritellään empiirinen todennäköisyystiheys $p_{(X,Y)emp}$ satunnaismuuttujajonolle muodostetun empiirisen kertymäfunktion $F_{(X,Y)emp}$ avulla. Lisäksi luodaan katsaus Heavisiden funktioon ja Diracin distribuutioon, joita tarvitaan empiirisen kertymäfunktion laskemiseen.

Empiirinen kertymäfunktio

Satunnaismuuttujasta (X, Y) muodostetaan n-jono eli $(X_1, Y_1), \cdots (X_n, Y_n)$. Jonon alkioiden havaitut arvot ovat opetusaineisto $S = \{(x_i, y_i) | 1 \leq i \leq n\}$. Satunnaismuuttujien jonolle määritellään empiirinen kertymäfunktio $F_{(X,Y)emp}$ seuraavasti

$$F_{(X,Y)emp}(u_1, u_2) = \frac{\sum_{i=1}^{n} I_{\{X_i \leq u_1, Y_i \leq u_2\}}(\omega)}{n} = \frac{card\{i \in \{1, \dots, n\} | X_i(\omega) \leq u_1, Y_i(\omega) \leq u_2\}}{n}, \tag{10}$$

missä $I_A(\omega)$ on asiaintilan ilmaisin, joka määritellään kaavalla

$$I_A(\omega) = \begin{cases} 1, & \omega \in A \\ 0, & \omega \notin A \end{cases} \tag{11}$$

ja $card(A)$ tarkoittaa joukon A kardinaliteettia eli joukon A mahtavuutta. Äärellisessä joukossa mahtavuus on joukon alkioiden lukumäärä [Koistinen 2005: 15]. Merkintä $\{X_i(\omega) \leq u_1, Y_i(\omega) \leq u_2\}$ tarkoittaa tapahtumaa $\{\omega \in \Omega | X_i(\omega) \leq u_1, Y_i(\omega) \leq u_2\} \in \mathcal{F}$. [Jäntti 2012.]

Heavisiden funktiot

Seuraavaksi määritellään empiirinen todennäköisyystiheys $p_{(X,Y)emp}$ empiirisen kertymäfunktion avulla

$$p_{(X,Y)emp}(x,y) = \frac{\partial^2 F_{(X,Y)emp}(x,y)}{\partial x \partial y} = \frac{1}{n}\sum_{i=1}^{n} \frac{\partial^2 I_{\{X_i \leq x, Y_i \leq y\}}}{\partial x \partial y} \quad . \tag{12}$$

Kaavan 12 empiirinen kertymäfunktio esitetään yksiulotteisien askelfunktioiden eli niin kutsuttujen Heavisiden funktioiden $Hx(x)$ ja $Hy(y)$ tulojen summana

$$F_{(X,Y)emp}(x,y) = \frac{1}{n}\sum_{i=1}^{n} Hx(x - x_i)Hy(y - y_i) \quad . \tag{13}$$

Heavisiden funktiot [Liukkonen 2007: 29] määritellään

$$Hx(x) = \begin{cases} 0, & x < 0 \\ 1, & x \geq 0 \end{cases}, \qquad Hy(y) = \begin{cases} 0, & y < 0 \\ 1, & y \geq 0 \end{cases}, \tag{14}$$

joista saadaan

$$Hx(x)Hy(y) = \begin{cases} 0, & x < 0, y < 0 \\ 1, & x \geq 0, y \geq 0 \end{cases} . \tag{15}$$

Diracin distribuutio

Distribuutioteorian mukaan

$$\frac{dHx(x - x_i)}{dx} = \delta(x - x_i) \quad . \tag{16}$$

Kaavassa 16 esiintyvä $\delta(x - x_i)$ on Diracin distribuutio eli yleistetty funktio [Liukkonen 2007: 25, 27–28]. Liukkonen [2007] tiivistää artikkelissaan seuraavasti:

> Distribuution erikoistapaus on mitta. Sitä voidaan havainnollistaa jakaumana, joka voi kuvata esimerkiksi todennäköisyyden jakautumista. Fyysikko Paul Diracin mukaan nimetty Diracin mitta tai Diracin delta kuvaa yhteen ainoaan pisteeseen keskittynyttä tiheysjakaumaa. [Liukkonen 2007: 25.]

Diracin distribuutiolla $\delta(x)$ on ominaisuus

$$\int_{\mathcal{X}} g(x)\,\delta(x - x_i)dx = g(x_i)\,, \tag{17}$$

jossa $g\colon \mathcal{X} \to \mathbb{R}$ on jatkuva funktio. Lisäksi distribuutioiden tulolle on voimassa

$$\int_{\mathcal{X}\times\mathcal{Y}} h(x,y)\,\delta(x - x_i)\delta(y - y_i)dxdy = h(x_i, y_i)\,, \tag{18}$$

jossa kuvaus $h\colon \mathcal{X} \times \mathcal{Y} \to \mathbb{R}$ on myös jatkuva funktio [Jäntti 2012].

Empiirinen todennäköisyystiheys

Empiirinen todennäköisyystiheys saadaan edellä esiteltyjen empiirisen kertymäfunktion, Heavisiden funktion ja distribuutioteorian avulla muotoon

$$\begin{aligned} p_{(X,Y)emp}(x,y) &= \frac{1}{n}\sum_{i=1}^{n} \frac{\partial^2 F_{(X,Y)emp}(x,y)}{\partial x \partial y} \\ &= \frac{1}{n}\sum_{i=1}^{n} \frac{\partial^2 (Hx(x - x_i)Hy(y - y_i))}{\partial x \partial y} \\ &= \frac{1}{n}\sum_{i=1}^{n} \frac{dHx(x - x_i)}{dx}\frac{dHy(y - y_i)}{dy} \\ &= \frac{1}{n}\sum_{i=1}^{n} \delta\,(x - x_i)\delta(y - y_i) \quad . \end{aligned} \tag{19}$$

Empiirinen riski

Määritellään riski kustannusfunktion odotusarvona kaavalla

$$R(f) = E\big[L\big(x, y, f(x)\big)\big] = \int_{\mathcal{X}\times\mathcal{Y}} L\big(x, y, f(x)\big)p(x,y)dxdy \quad . \tag{20}$$

Käytetään kaavassa 19 määriteltyä empiiristä todennäköisyystiheyttä $p_{(X,Y)emp}(x,y)$, ja sijoitetaan se todennäköisyystiheyden $p(x,y)$ tilalle:

$$R(f) = \int_{X \times y} L(x,y,f(x)) p_{(X,Y)emp}(x,y) dxdy \,. \tag{21}$$

Kaavaan 21 on sijoitettu todennäköisyystiheys $p_{(X,Y)emp}(x,y)$ ja täten kaavan 19 mukaan riskifunktionaali on muotoa

$$R(f) = \int_{X \times y} L\big(x,y,f(x)\big) \frac{1}{n} \sum_{i=1}^{n} \delta(x - x_i)\delta(y - y_i)\ dxdy \,, \tag{22}$$

jonka termien järjestystä muokkaamalla saadaan

$$R(f) = \frac{1}{n} \sum_{i=1}^{n} \int_{X \times y} L\big(x,y,f(x)\big)\delta(x - x_i)\delta(y - y_i) dxdy \,. \tag{23}$$

Ottaen huomioon Diracin distribuution ominaisuudet, jotka on esitelty kaavoissa 17 ja 18, saadaan poistettua Diracin distribuutiot $\delta(x - x_i)$ ja $\delta(y - y_i)$ sekä niihin liittyvä integraali ja siten muodostettua empiirinen riski $R_{emp}(f)$, kuten kaavassa 24:

$$R(f) = \frac{1}{n} \sum_{i=1}^{n} L\big(x_i, y_i, f(x_i)\big) \equiv R_{emp}(f) \tag{24}$$

Tavoitteena on löytää sellainen funktio f, joka minimoi empiirisen riskin eli $\min R_{emp}(f)$. [Schölkopf & Smola 2002: 65–67; Vapnik 2000: 18–21.]

Empiirisen riskin laskeminen ja minimointi on vaivatonta, mikäli ei päädytä huonosti asetettuun ongelmaan *(ill-posed problem)*. Sanotaan, että matemaattinen tehtävä on huonosti asetettu, jos pienetkin muutokset lähtöarvoissa aiheuttavat suuria muutoksia ratkaisussa. Jossain tapauksessa tehtävän pystyy muotoilemaan paremmin, mutta mikäli se ei onnistu niin tulee käyttää säännöllistämismenetelmää, jolla tehtävä saadaan vakaaksi. Säännöllistäminen *(regularization)* pienentää lähtöarvojen virheiden vaikutusta, mutta samalla siitä aiheutuu säännöllistämisvirhe. Nämä kaksi virhettä tulee saada tasapainoon määrittämällä säännöllistämisehtoja sekä valitsemalla oikeat para-

metrit. Säännöllistämisestä laajemmin lähteessä Schölkopf & Smola 2002: 87–122. [Leino 2004: 3; Schölkopf & Smola 2002: 67.]

Tämän luvun lopuksi tiivistetään vielä oppimisen mallintaminen yhteen lauseeseen: Koneoppimismenetelmän tulee opetusaineiston S avulla tehdä koneoppijasta f sellainen, jonka odotusarvoinen kustannus $E\big[L\big(x, y, f(x)\big)\big]$ on mahdollisimman pieni. Tässä luvussa käsiteltiin vain empiirisen riskin minimointia. On myös muita vaihtoehtoja ratkaista tuntematon satunnaismuuttujan todennäköisyysjakauma $P_{X,Y}$ ja yksi niistä on EM- eli Expectation Maximization -algoritmi [Jäntti 2012]. Siihen ei perehdytä tässä insinöörityössä, mutta siihen voi tutustua esimerkiksi lähteessä Gupta & Chen 2010.

4.2 Oppimisen menetelmiä

Oppimisen menetelmiä ovat lineaarisiin malleihin kuuluvat diskriminanttifunktio *(discriminant function)* sekä diskriminatiivinen ja generatiivinen todennäköisyysmalli *(discriminative and generative probability models)* [Bishop 2006: 43]. Tässä insinöörityöraportissa esitellään diskriminanttifunktio (luku 4.2.1) ja diskriminatiivinen todennäköisyysmalli (luku 4.2.2).

Diskriminanttifunktion tehtävänä on jakaa piirreavaruus päätösalueisiin *(decision regions)*, joiden rajoja kutsutaan päätöspinnoiksi *(decision boundaries, decision surfaces)*. Lineaarisia malleja käytetään ainoastaan lineaarisesti erottuvan *(linearly separable)* tietoaineiston päätösalueiden määrittämiseen. Diskriminatiivinen ja generatiivinen todennäköisyysmalli hyödyntävät Bayesin päätösteoriaa *(Bayesian decision theory),* lisätietoa lähteestä Seppänen 2012: 13–53. [Bishop 2006: 43; 179–180.]

4.2.1 Diskriminanttifunktio

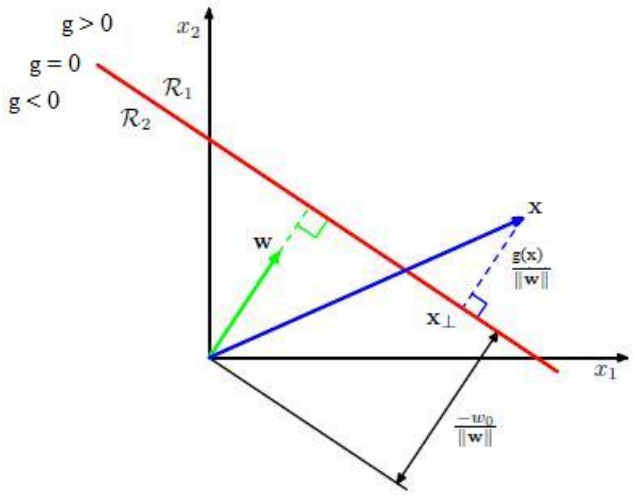
Diskriminanttifunktiota $g(x)$ voidaan kutsua suomenkielellä erottelufunktioksi, joka kuvaa sen toimintaa $\mathcal{X}$-avaruuden jakamiseksi kohdeluokkia vastaaviin päätösalueisiin R_K. Havainto luokitellaan siihen luokkaan, jonka diskriminanttifunktio saa suurimman arvon pisteessä x. Aiempaan tapaan myös tässä luvussa tutkitaan vain kahden luokan tapausta, jolloin $K = 2$ ja päätösalueet vastaavasti R_1 sekä R_2. Yleisesti diskriminantti-

funktioita on yhtä monta kuin luokkiakin, mutta kaksiluokkaisessa tapauksessa ne voidaan yhdistää yhdeksi funktioksi $g(x) = g_1(x) - g_2(x)$. [Bishop 2006: 181; Seppänen 2012: 20, 22, 75.]

Yksinkertaisin diskriminanttifunktio on

$$g(x) = w^T x + w_0 \ , \tag{25}$$

jossa x on havainto eli piirrevektori, w on painokerroinvektori *(weight vector)*, w_0 on bias-termi *(bias)* eli kynnyspainokerroin ja $w^T x$ merkitsee painokerroinvektorin transpoosin ja piirrevektorin pistetuloa. Mainittakoon vielä, että painokerroinvektori ja piirrevektori ovat samanulotteisia [Tohka 2009: 30]. Lineaarisuus tarkoittaa tässä sitä, että diskriminanttifunktio on piirrevektorin x suhteen lineaarinen. Päätösalueet erottavan päätöspinnan, jota kutsutaan myös hypertasoksi *(hyperplane)*, määrittelee $g(x) = 0$, jolloin kaikilla x:n arvoilla diskriminanttifunktio saa arvokseen 0. Kuvassa 11 sitä havainnollistaa punainen suora. Mikäli $g(x) \geq 0$, kuuluu piirrevektori luokkaan C_1 eli päätösalueeseen R_1 ja jos $g(x) < 0$, niin $x \in C_2$ eli päätösalueeseen R_2. [Bishop 2006: 181.]



Kuva 11. Havainnollistaa geometrisesti kaksiulotteista lineaarista diskriminanttifunktiota, joka jakaa avaruuden $\mathcal{X}$ kuvassa olevalla punaisella suoralla eli päätöspinnalla kahteen päätösalueeseen [Bishop 2006: 182]. Tällainen päätöspinta voidaan toteuttaa esimerkiksi perceptron-algoritmilla [Vapnik 2000: 1–7]. Tässä kuvassa origo on päätöspinnan negatiivisella puolella ja siitä syystä kynnyspainokerroin $w_0 < 0$, lisäksi painokerroinvektorin $w = [w_1, w_2]^T$ komponentit $w_1 > 0$ ja $w_2 > 0$, koska ne sijaitsevat koordinaattiakseleiden x_1 ja x_2 positiivisella puolella. Kuva lähdettä [Bishop 2006: 182] mukaillen.

Kuvan 11 vihreä vektori on päätösalueen R_1 suuntaan eli se on avaruuden $\mathcal{X}$ positiiviselle puolelle osoittava painokerroin w [Kääriäinen 2008b: 17] ja se on kohtisuorassa jokaista päätöspinnalla sijaitsevaa vektoria vastaan. Olkoot siis x_1 ja x_2 päätöspinnalla sijaitsevia pisteitä. Tällöin toteutuu

$$g(x_1) = g(x_2) = 0 \tag{26}$$

ja

$$w^T(x_1 - x_2) = 0 \tag{27}$$

joka johdetaan yhtälöstä

$$w^T x_1 + w_0 = w^T x_2 + w_0 \ . \tag{28}$$

Painokerroinvektori w päättää siis päätöspinnan suunnan. Mikäli päätöspinta kulkee origon kautta, sen kynnyspainokerroin $w_0 = 0$. Toisaalta, jos $w_0 > 0$ on origo päätöspinnan positiivisella puolella ja jos $w_0 < 0$ se on päätöspinnan negatiivisella puolella. Kynnyspainokertoimen avulla päätöspintaa voidaan siirtää origosta poispäin ja siten määrätä sen sijainti. [Bishop 2006: 181–182; Seppänen 2012: 78.]

Diskriminanattifunktion $g(x)$ arvo määrää pisteen x etäisyyden r päätöspinnasta. Olkoon x sattumanvarainen piste ja $x_\perp$ sen ortogonaalinen projektio päätöspinnalle (projektiopiste $x_\perp$ sijaitsee päätöspinnalla). Tällöin saadaan yhtälö

$$x = x_\perp + r\frac{w}{\|w\|} \ , \tag{29}$$

joka sijoitetaan kaavaan 30 seuraavasti:

$$g(x) = g\left(x_\perp + r\frac{w}{\|w\|}\right) \ . \tag{30}$$

Kaavan 30 yhtälöä muokataan yleisen diskriminanttifunktion (kaava 25) mukaiseksi, jolloin päästään muotoon

$$g(x) = w^T\left(x_\perp + r\frac{w}{\|w\|}\right) + w_0 \ . \tag{31}$$

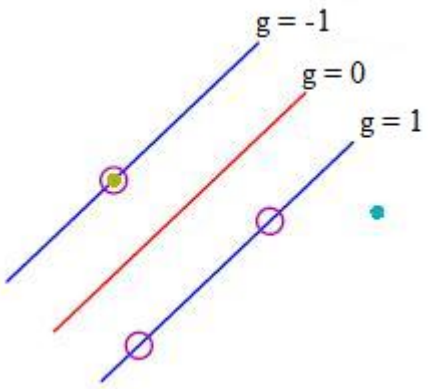
Kaavasta 31 ratkaistaan r, ja täten pisteen x etäisyys päätöspinnasta on

$$r_{PäätöspintaPistex} = \frac{|g(x)|}{\|w\|} \quad . \tag{32}$$

Seuraavaksi lasketaan päätöspinnan etäisyys origosta. Siinä tapauksessa $x = 0$ eli $g(0)$, ja kaava 25 sijoitetaan kaavaan 32 x:n arvolla 0 ja näin saadaan yhtälö

$$r_{origoPäätöspinta} = \frac{|g(0)|}{\|w\|} = \frac{|w^T \times 0 + w_0|}{\|w\|} = \frac{|w_0|}{\|w\|} \quad . \tag{33}$$

Kaavoihin 29–33 liittyvät geometriset tulkinnat löytyvät kuvasta 11. [Bishop 2006: 181–182; Seppänen 2012: 77–78.]



Kuva 12. Tukivektorikone laskee päätöspinnan (punainen suora) lisäksi marginaalitasot (siniset suorat). Lähdettä [Bishop 2006: 327] mukaillen.

Diskriminanttifunktiota hyödynnetään muun muassa Perceptron-algoritmissa, johon voi tutustua Vladimir Vapnikin kirjassa The Nature of Statistical Learning Theory [2000: 1–7]. Perceptron on yli 60 vuoden takainen F. Rosenblattin esittelemä malli koneoppijasta, joka jakaa tietoaineiston kahteen luokkaan annettujen esimerkkien eli opetusaineiston avulla [Vapnik 2000: 1–2]. Toinen lineaarisiin luokittimiin kuuluva, perceptronia kehittyneempi malli on tukivektorikone [Bishop 2006: 326–344; Vapnik 2000: 131–146], joka päätöspinnan löytämisen lisäksi maksimoi sen ja lähimpien tietoaineistopisteiden väliset marginaalit [Bishop 2006: 326]. Kuvasta 12 löytyy päätöspinnan lisäksi marginaalitasot, joita kuvassa 11 ei ole. Tukivektorikonetta käytetään VOC-menetelmän opetusvaiheessa ja siitä lisää luvussa 5.2.2.

4.2.2 Diskriminatiivinen todennäköisyysmalli

Tässä luvussa tutustutaan luokitteluongelmaan diskriminatiivisesta näkökulmasta. Esitellään ensin kaavoissa käytettävät merkinnät: Luokkia merkitään muuttujalla y ja havainnosta laskettu piirrevektori on x. Muuttuja n on luokkien lukumäärä, joten $y_1, \dots, y_n$, ja kun n saa arvokseen kaksi, niin luokat ovat y_1 ja y_2.

Seuraavaksi määritellään ehdollinen todennäköisyys [Mellin 2010e: 69]. Sitä merkitään $P(y_1|x)$, joka tarkoittaa tapahtuman y_1 (luokka 1) ehdollista todennäköisyyttä ehdolla, että tapahtuma x on jo sattunut. Ehdollista todennäköisyyttä kuvataan yhtälöllä

$$P(y|x) = \frac{P(y \cap x)}{P(x)} , \tag{34}$$

joka on siis y:n ja x:n leikkauksen (alkiot, jotka kuuluvat sekä joukkoon y ja joukkoon x) todennäköisyyden suhde x:n todennäköisyyteen.

Diskriminatiivinen luokittelu perustuu ehdollisia todennäköisyyksiä sisältävään Bayesin päätösteoriaan [Seppänen 2012: 13–15], kuten edellä tässä luvussa mainittiin. Sen mukaan Bayesin päätössääntö [Seppänen 2012: 15] on

$$x \in y_1, \qquad jos\ P(y_1|x) > P(y_2|x) \tag{35}$$

ja

$$x \in y_2, \qquad jos\ P(y_2|x) > P(y_1|x). \tag{36}$$

Yllä esitetty päätössääntö minimoi luokitteluvirheen keskimääräisen todennäköisyyden, joka on pienin mahdollinen todennäköisyys kaikilla x:n arvoilla. Päätössäännön (kaava 35 ja 36) perusteella muodostetaan päätöspinta

$$P(y_1|x) - P(y_2|x) = 0, \tag{37}$$

jonka toisella puolella pätee $P(y_1|x) > P(y_2|x)$ ja toisella $P(y_2|x) > P(y_1|x)$, näin avaruus $\mathcal{X}$ on jaettu lineaarisen malli tapauksessa kahteen päätösalueeseen [Jäntti 2011].

Satunnaismuuttujan X, Y todennäköisyysjakaumaa $P_{X,Y}$ ei kuitenkaan tunneta, mikä tarkoittaa sitä, että kaavojen 35, 36 ja 37 todennäköisyyksiä ei myöskään tiedetä. Yksi ratkaisuvaihtoehto on diskriminanttifunktioiden käyttö. Sijoitetaan $g_1(x) = P(y_1|x)$ ja $g_2(x) = P(y_2|x)$, joiden erotuksesta $g(x) = g_1(x) - g_2(x)$ muodostuu edellisestä luvusta tuttu päätöspinta $g(x) = w^T x + w_0 = 0$. Tähän voidaan päätyä erilaisilla menetelmillä, mutta niihin ei perehdytä tässä insinöörityössä. Vaihtoehtoisesti voidaan soveltaa luvussa 4.1.2 esitettyä empiirisen riskin minimointia. [Jäntti 2011–2012.]

5 Tutkitut hahmontunnistusmenetelmät

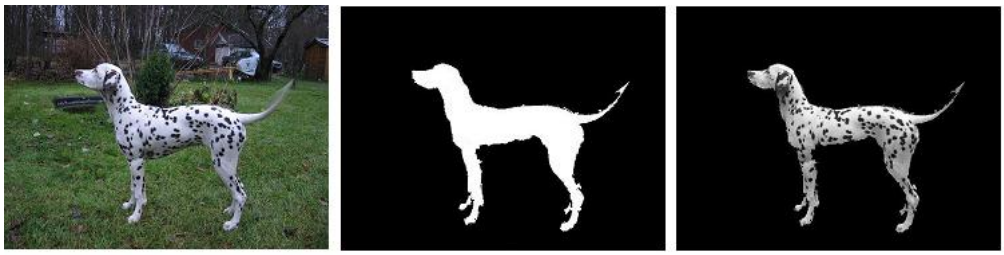
Hahmontunnistuksen menetelmät voidaan luokitella monin tavoin. Yleisellä tasolla ne kuuluvat tilastolliseen, syntaktiseen tai neuroverkkoihin perustuvaan hahmontunnistukseen [Tohka 2009: 1]. Tässä insinöörityöraportissa luokittelu perustuu vain opetusvaiheessa tai tunnistusvaiheessa käytettyihin malleihin. Tarkastelun kohteena olevat hahmontunnistusmenetelmät ovat kaikki tilastollisia ja ne jaetaan opetusvaiheen perusteella kahteen luokkaan: koodikirjaan *(codebook, vocabulary)* pohjautuviin (luku 5.1) sekä diskriminatiivisesti opetettuihin menetelmiin (luku 5.2).

5.1 Koodikirjaan perustuvat menetelmät

Tässä luvussa tarkastellaan ensin tässä insinöörityössä tutkittuja opetus- ja tunnistusvaiheessa koodikirjaa työkalunaan käyttäviä menetelmiä (luku 5.1.1), minkä jälkeen tutustutaan käsitteeseen koodikirja ja perehdytään sen sisällön määrittämiseen (luku 5.1.2).

5.1.1 ISM-, BoVW- ja konstellaatiomallit

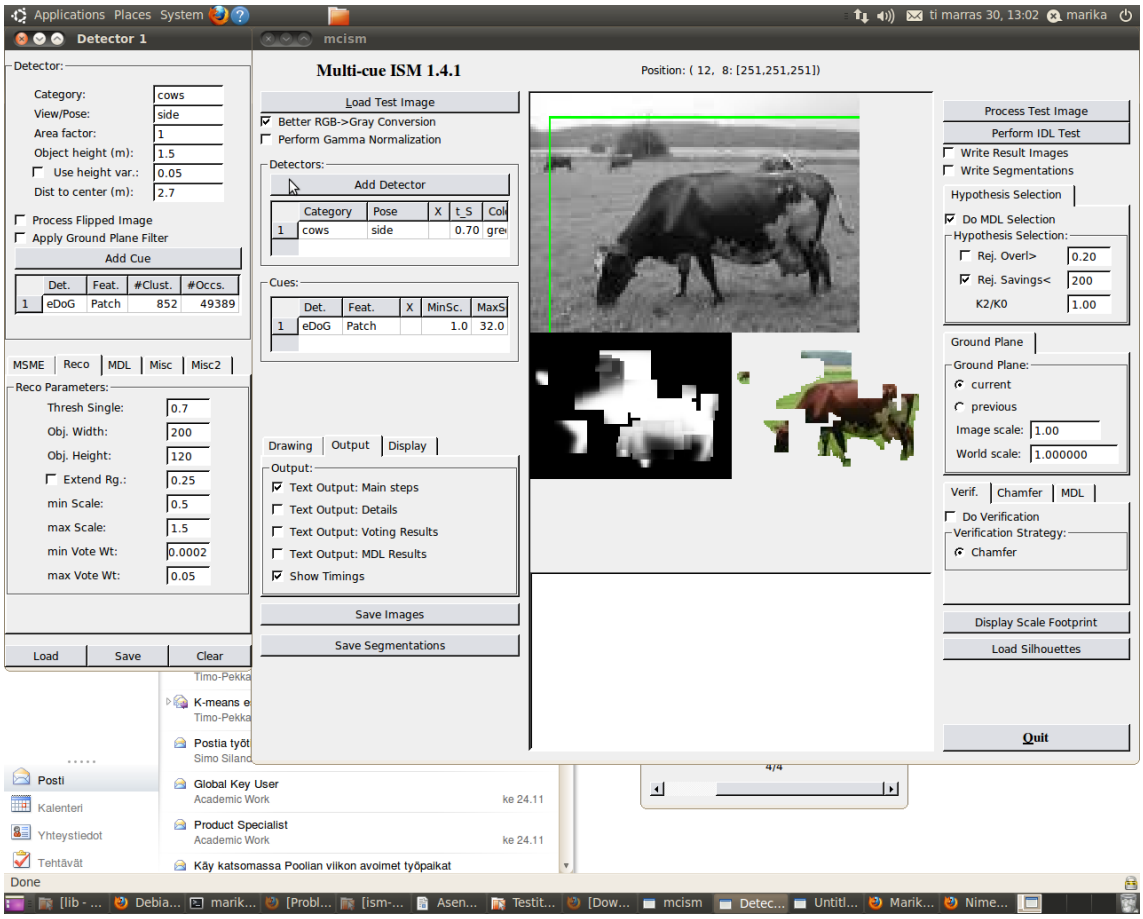
Tässä insinöörityössä hahmontunnistusmenetelmien tutkiminen on alkanut Leiben, Leonardisin ja Schielen artikkelista [2008] sekä Seemannin [2007] väitöskirjassa esitetystä Implicit Shape Model- eli ISM-menetelmästä, jossa järjestelmä opetetaan tunnistamaan kohteita visuaalisen koodikirjan *(visual codebook)* avulla.



Kuva 13. Vasemmalta oikealle: alkuperäinen kuva, segmentointimaski ja segmentoitu kuva.

ISM:n pohjalta on toteutettu hahmontunnistussovellus nimeltä Multi-Cue ISM [Leibe 2008], jonka toiminta vastaa ensin mainitussa artikkelissa esitettyä hahmontunnistusmenetelmää. Kyseessä on GNU/Linux-käyttöjärjestelmässä toimiva työasemasovellus, jonka versio 1.4.1 on tätä insinöörityötä toteutettaessa testattu toimivaksi GNU/Linux-jakelu Ubuntu versiolla 10.04 LTS. Liite 1 tarkentaa sovelluksen omia asennusohjeita, jotka löytyvät <http://www.vision.ee.ethz.ch/~bleibe/code/ism.html> ladatun jakelupaketin tiedostosta INSTALL.txt.

Multi-Cue ISM saa syötteenä käyttäjän antaman kuvan ja se palauttaa käyttäjälle segmentoidun kuvan. Kuva 13 havainnollistaa segmentointia *(segmentation)*, jonka tavoitteena on jakaa kuva haluttuihin alueisiin tai kohteisiin [Gonzales & Woods: 567]. Segmentointimaski on toteutettu kuvassa 13 käsin, ja kun kuva kerrotaan segmentointimaskilla (käyttäen esimerkiksi MATLAB-ohjelmaa), saadaan taustastaan poisrajattu koira eli segmentoitu kuva. Kuva 14 edustaa sovelluksella tehtyä segmentointia, ja kuten siitä voi havaita, segmentointitulokset eroavat paljon verrattuna käsin toteutettuun segmentointiin: käsin segmentoidussa kuvassa koira on tarkasti rajattu, kun taas automatisoidulla segmentoinnilla tuotettu kuva ei sisällä kohdetta kokonaan, ja sen lisäksi segmentoidussa kuvassa on taustaakin mukana. Tästä voi päätellä sovelluksella toteutetun segmentointitehtävän hankaluuden.



Kuva 14. Multi-cue ISM 1.4.1 -sovelluksen käyttöliittymä. Valokuvista ylin on käyttäjältä saatu syötekuva. Sen alla vasemmalla on segmentointimaski, jonka sovellus on syötekuvaan pohjautuen laskenut. Maskin oikealla puolella on segmentoinnin tulos.

Tämän insinöörityön tavoitteiden näkökulmasta katsottuna ongelmana on ISM-sovelluksen käyttökelvottomuus KOIRAKOssa, koska ohjelmaan ladattavat koodikirjat ovat binääritiedostoja, eikä koirasta tehtyä koodikirjaa ole saatavilla. Koodikirjaa ei pysty ohjelmoimaan itse, sillä koodikirjan tiedostomuotoa ei ole julkisesti dokumentoitu.

Tämän insinöörityön aihepiiriin perehtymistä aloittaessa ei ole ollut tiedossa kuin koodikirjan avulla opetettavia hahmontunnistusmenetelmiä. Siitä syystä visuaalinen koodikirja on ollut ratkaisevassa osassa ja sitä on tutkittu syvemmin. Toinen ISM:n rinnalle tässä insinöörityössä noussut hahmontunnistusmenetelmä on Bag of Visual Words eli BoVW [Lin ym. 2009: 1308; Perronnin 2008: 1243], jonka opetusvaihe myös sisältää visuaalisen koodikirjan ja jopa koodikirjan luominen tapahtuu lähes samoin tavoin kuin ISM:ssä. BoVW-menetelmä lähestyy tunnistusongelmaa kuitenkin erinäkökulmasta: Se on tarkoitettu tunnistamaan monia kohdeluokkia, kun taas ISM keskittyy yhteen

kohteeseen kerrallaan. Myöskään tästä menetelmästä ei ole apua KOIRAKOn sovelluslogiikan toteuttamiseen, sillä tätä kirjoittaessa siitä ei ole saatavilla muokattavissa olevaa toteutusta.

Seuraava tämän insinöörityön tutkimuksen aikana esiin tullut menetelmä on Fergusin [2005] sekä Weberin [2000] väitöskirjoissa käsiteltävä konstellaatiomalli *(constellation model)*, johon tässä työssä ei perehdytä lainkaan. Tämän insinöörityön kannalta kiinnostavaa näissä väitöskirjoissa on se, että Fergus on ohjelmoinut kaksi konstellaatiomalliin liittyvää MATLAB-pohjaista toteutusta [Fergus 2003; Fergus 2005a], joiden ohjelmakoodit ovat julkisesti saatavilla. Kyseisten toteutusten funktiot olisivat voineet olla hyödynnettävissä KOIRAKO-sovelluksen toteuttamisessa. Menetelmää ei kuitenkaan ole valittu, koska Fergusin toteutukset ovat vaikeaselkoisesti ohjelmoituja ja vaativat liikaa muokkaamista KOIRAKOn käyttötarpeisiin.

Vaikka ISM-, BoVW- ja konstellaatiomenetelmät eivät ole tuoneet ratkaisua tämän insinöörityön tutkimusongelmaan, niin tässä insinöörityössä tutustutaan silti koodikirjaan (luku 5.1.2) sekä perehdytään piirteenirrotukseen muun muassa Leiben, Leonardisin ja Schielen [2008: 17–19] artikkelissa päteväksi todetun Scale Invarian Feature Transform -algoritmin eli SIFTin (luku 6.1) avulla.

5.1.2 Koodikirja

Käsitteellä koodikirja tarkoitetaan muun muassa Implicit Shape Model- ja Bag of Visual Words -menetelmien opetus- ja tunnistusvaiheissa käytettävää työkalua. Tässä luvussa sekä perehdytään visuaaliseen koodikirjaan että tarkastellaan sen sisällön muodostamista yleisellä tasolla. Selkeimmin visuaalista koodikirjaa käsitellään Leiben [Leibe ym. 2008: 4–7] sekä Agarwalin [2004: 1476–1479] artikkeleissa.

Kuva 15. Esimerkki ISM-menetelmällä muodostetusta koodikirjasta. Lähdettä [Leibe ym. 2008: 7] mukaillen.

Koodikirja on tietovarasto eli ohjelmoinnin näkökulmasta katsottuna yksinkertainen tietorakenne, jonka sisältö koostuu opetuskuvista kerätystä ja erilaisilla menetelmillä käsitellystä tiedosta. Koodikirjan luomiseen vaikuttavat ympäristö, käyttötarkoitus ja tallennettava sisältö. Kuva 15 havainnollistaa ISM-menetelmän [Leibe ym. 2008: 7] mukaista koodikirjan muodostamista, jota käsitellään myöhemmin tässä luvussa.

Ensimmäinen koodikirjaan vaikuttava tekijä on ympäristö, jota varten koodikirja rakennetaan. Ajatellaan, että koodikirja on hahmontunnistussovellukseen liitettävä moduuli. Tässä työssä koodikirja sisältää tietoa koiran piirteistä, jotka on ilmaistu piirrekuvaajien (vastaa aiemmin tässä insinöörityöraportissa esitettyä käsitettä piirrevektori) avulla. Yhtälailla koodikirja voidaan rakentaa tuoleista, lehmistä tai muista kohteista, joita halutaan tunnistaa.

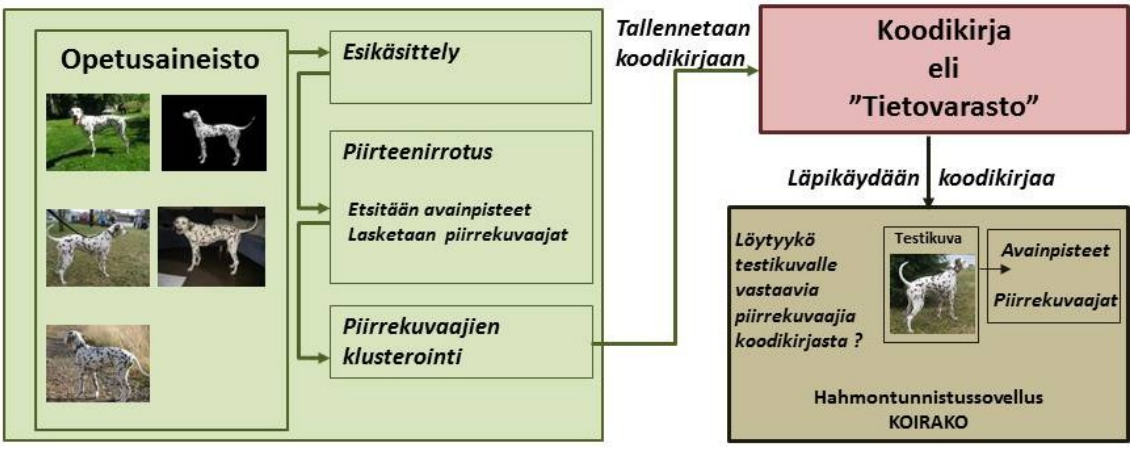
Seuraavaksi analysoidaan hahmontunnistussovelluksen käyttötarkoitus, joka yhdessä toteutuksessa käytettävien algoritmien kanssa asettavat ehtoja koodikirjan sisällön määrittämiseen. ISM-mallin koodikirjoissa [Leibe ym. 2008: 14–15] käytetään käsin segmentoitua opetusaineistoa, kun taas konstellaatiomenetelmässä [Fergus 2005b: 7] opetusmateriaali on segmentoimatonta. Myös opetusaineiston määrät vaihtelevat. Raskaampi ISM-malli [Leibe ym. 2008: 14–15] tarvitsee satoja kuvia ja konstellaatiomallissa [Fei-Fei ym. 2003a: 1134] pyritään käyttämään alle kymmentä opetuskuvaa.

Kolmas koodikirjaan vaikuttava tekijä on siihen tallennettava sisältö. Koodikirjaan tallennetaan yleensä vain avainpisteen kohdalta otettu kuvatilkku koko kuvan sijaan sekä opetuskuvan avainpisteet esitettyinä piirrekuvaajina. Näiden lisäksi ISM-menetelmän koodikirjaan tallennetaan edellisten lisäksi sekä kuvatilkkujen sijainti suhteessa objektin keskipisteeseen että kuvan skaala. BoVW-malli ei ota kantaa

kohteen osien eli piirteiden keskinäisiin suhteisiin ja menetelmä lähestyy tunnistusongelmaa muutenkin eri näkökulmasta. Sen tarkoituksena on tunnistaa monia kohdeluokkia, kun taas ISM keskittyy vain yhteen kohdeluokkaan kerrallaan. [Fergus ym. 2005: 380; Leibe ym. 2008: 7.]

Koodikirjan sisällön määrittäminen

Koodikirja on pohjimmiltaan listamuotoinen tietovarasto kuviin liittyvän tiedon tallentamiseen ja hakemiseen. Sellaisen toteuttamiseen soveltuvia tietorakenteita ovat esimerkiksi MATLABin solutaulukko sekä Javan suoritusympäristöihin usein sisältyvät listoja toteuttavat tietorakenneluokat. Tietorakenteisiin tallennettava tieto sekä tavat sen hyödyntämiseen riippuvat kuvien tunnistuksessa sovellettavista algoritmeista.



Kuva 16. Esimerkki visuaalisen koodikirjan muodostamisesta ja käytöstä.

Koodikirja (kuva 16) voidaan luoda eri menetelmillä, mutta sen sisällön määrittämiseen kuuluu

1. opetusaineiston esikäsittely
2. piirteenirrotus
    a. avainpisteiden havaitseminen
    b. piirrekuvaajien muodostaminen
3. piirrekuvaajien klusterointi.

Opetusaineiston esikäsittelyn tarpeellisuuden sekä esikäsittelyssä sovellettavat menetelmät määrää työn lopullinen tarkoitus. Tämän insinöörityöraportin esimerkeissä

koodikirjaan tallennettavat opetuskuvat ovat segmentoituja, koska ISM-menetelmässä tehdään niin. Segmentointi ei kuitenkaan ole välttämättömyys [Agarwal ym. 2004: 1478; Weber 2000: 3; Mikolajczyk & Schmid 2003: 257], ja koodikirjan sisällön muodostamisessa sovellettavien algoritmien pitäisi toimia, vaikka kuva sisältäisikin kohteen lisäksi taustan.

Jos kuvia ei esikäsitellä, koodikirjan sisällön muodostaminen alkaa piirteenirrotuksella, joka sisältää avainpisteiden *(interest points, keypoints)* havaitsemisen sekä piirrekuvaajien *(descriptors)* muodostamisen. Avainpisteet ovat kuvan pisteitä, jotka ovat muuttumattomia *(invariant)* skaalan *(scale)* ja kuvakulman vaihtuessa. Ne saadaan selville avainpisteidenhavaitsija-algoritmilla *(interest point detector)*, johon perehdytään luvuissa 6.1.1 ja 6.1.2. Sen jälkeen havaitsija-algoritmin edellä löytämien avainpisteiden perusteella muodostetaan kuvan ominaisuudet eli piirteet määrittelevä vektori – piirrekuvaaja [Leibe ym. 2008: 4–5; Seemann 2007: 44].

Vaiheet 1–2 toistetaan kaikille opetuskuville, minkä jälkeen on koossa suuri määrä avainpisteiden avulla etsittyjä ja piirrekuvaajilla kuvattuja opetuskuvien piirteitä. Seuraavaksi piirteet klusteroidaan, jolla tarkoitetaan samankaltaisten piirteiden ryhmittelyä omiin luokkiinsa. Klusteroidut piirteet hakuun soveltuvaan tietorakenteeseen tallennettuina muodostavat visuaalisen koodikirjan. [Leibe ym. 2008: 5, Seemann 2007: 44.]

Tunnistusvaiheessa testikuvan piirrekuvaajaa verrataan koodikirjaan tallennettuihin opetusaineiston piirrekuvaajiin [Leibe ym. 2008: 8–9]. Testimateriaalille tehtävät vaiheet riippuvat hahmontunnistussovelluksen käyttötarkoituksesta: Toisaalta Leiben Multi-cue ISM 1.4.1 -sovelluksen [Leibe 2008] tavoitteena on segmentoida syötteenä saatu kuva, joten siinä tapauksessa testimateriaalit ovat segmentoimattomia ja testimateriaalille tehdään vain vaiheet 2a–2b. Yleisesti ottaen testimateriaali käsitellään klusterointia lukuun ottamatta samoin kuin opetusaineistokin (vaiheet 1–2) [Bishop 2006: 2].

5.2 VOC-menetelmä

Felzenszwalb, Girshick, McAllester ja Ramanan [2009] esittelevät VOC-hahmontunnistussovelluksen teorian artikkelissa Object Detection with Discrminatively Trained Part Based Models [2010]. Teoriaa kutsutaan tässä työssä VOC-menetelmäksi ja siihen viitataan myös pelkällä VOC-nimityksellä. Menetelmällä ei ole virallista nimeä, joten siitä syystä se on lainattu sovelluksen kotisivuilta <http://www.cs.brown.edu/~pff/latent-release3/> ladattavan jakelupaketin nimestä voc-release3.1.tgz. Lyhenne VOC tulee sanoista Visual Object Classes.

VOC on toteutettu vastaamaan The PASCAL Visual Object Classes Challenge 2007 eli VOC 2007 -kilpailun [Everingham ym. 2007] haasteisiin. Kilpailuun ohjelmoitavan sovelluksen on tarkoitus luoda järjestelmä, joka tunnistaa kohteita monista kohdeluokista ja opetuskuvien, kuten myös tunnistettavien testikuvien, tulee vastata normaalia ympäristöä eli kuvien tulee olla segmentoimattomia. Kohdeluokkia VOC 2007-kilpailussa oli 20 ja ne koostuivat muun muassa ihmisistä, eläimistä, ajoneuvoista ja esineistä. Felzenszwalbin VOC-sovellus sai parhaimmat pisteet yhdeksästä kohdeluokasta ja toiseksi parhaimmat pisteet kahdeksasta kohdeluokasta [Felzenszwalb ym. 2010: 16].

VOC-menetelmä ja sen VOC-sovellus valittiin KOIRAKO-sovellukseen, koska

- se menestyi hyvin The PASCAL VOC 2007 -kilpailussa
- opetuskuvia ei tarvitse nimiöidä käsin
- opetusmateriaali (yli 1 000 opetuskuvaa) on valmiina ja saatavilla
- sovellus on hyvin kommentoitu ja sitä on mahdollista muokata
- sovellus vaatii vain pieniä muutoksia toimiakseen koiran tunnistamisessa
- sovellus piirtää kuvasta löydetyn kohteen ympärille tunnistuskehyksen *(bounding box)*.

VOC eroaa edellisessä luvussa tutustuttuihin koodikirjaan perustuviin menetelmiin siten, että opetusaineistosta ei luoda koodikirjaa, vaan jokaista opetuskuvaa kohden on yksi kuvaa vastaava XML-annotaatio *(XML annotation)* (kuva 17), jota käytetään opetusvaiheessa. XML on lyhenne sanoista Extensible Markup Language, joka on tiedon merkityksen kuvaamiseen tarkoitettu standardi [Extensible Markup Language (XML) 1.0 (Fift Edition)].

```
<annotation>
    <folder>VOC2007</folder>
    <filename>000036.jpg</filename>
    <source>
        <database>The VOC2007 Database</database>
        <annotation>PASCAL VOC2007</annotation>
        <image>flickr</image>
        <flickrid>340478761</flickrid>
    </source>
    <owner>
        <flickrid>jmp45fr</flickrid>
        <name>?</name>
    </owner>
    <size>
        <width>332</width>
        <height>500</height>
        <depth>3</depth>
    </size>
    <segmented>0</segmented>
    <object>
        <name>dog</name>
        <pose>Unspecified</pose>
        <truncated>0</truncated>
        <difficult>0</difficult>
        <bndbox>
            <xmin>27</xmin>
            <ymin>79</ymin>
            <xmax>319</xmax>
            <ymax>344</ymax>
        </bndbox>
    </object>
</annotation>
```

Kuva 17. Ote positiivisesta opetusaineistosta, vasemmalla kuva kohteesta ja oikealla sen XML-annotaatio, joka sisältää muun muassa kohteen ympäröivän tunnistuskehyksen koordinaatit. Kuva ja annotaatio ovat peräisin lähteestä Everingham ym. 2007.

VOCin opetusaineiston kuvia ei segmentoida, kuten Leiben [2008: 14–15] ISM-menetelmän opetusaineistolle on tehty. Sen sijaan XML-annotaatio sisältää kohteen rajaavan tunnistuskehyksen koordinaatit ja tunnistuskehyksen sisällölle tehdään piirteenirrotus, aivan kuten ISM:ssä segmentoidulle kuvallekin. VOCissa piirteenirrotus on toteutettu muokkaamalla HOG eli Histogram of Oriented Gradients -algoritmilla laskettuja HOG-piirteitä VOCiin sopivaksi [Felzenszwalb ym. 2010: 12–14]. Kun taas Leiben [2008: 19] artikkelissa DoG- ja SIFT-algoritmien yhdistelmällä tehty piirteenirrotus tuotti hyviä tuloksia. Edellä mainittuja HOG-, DoG- ja SIFT-algorimeihin perehdytään tarkemmin tämän insinöörityön piirteenirrotusta käsittelevässä luvussa 6.

Seuraavaksi tässä luvussa esitellään VOC-sovelluksen teoriaa aloittamalla sen rakennetta kuvaavalla hahmontunnistuskehyksellä (luku 5.2.1), jonka jälkeen luodaan katsaus opetus- ja tunnistusvaiheeseen (luvut 5.2.2 ja 5.2.3). Opetusvaiheen sisältämä mallin rakentaminen sekä tunnistusvaiheen tunnistusprosessi ovat liian laajoja esitettäväksi yksityiskohtaisesti tässä yhteydessä. Niissä käytettyjen menetelmien esitykset löytyvät Felzenszwalbin [Felzenszwalb ym. 2010] artikkelista: mallin rakentaminen luvusta

4 [Felzenszwalb ym. 2010: 8–10] ja tunnistusprosessi luvusta 3.2 [Felzenszwalb ym. 2010: 6].

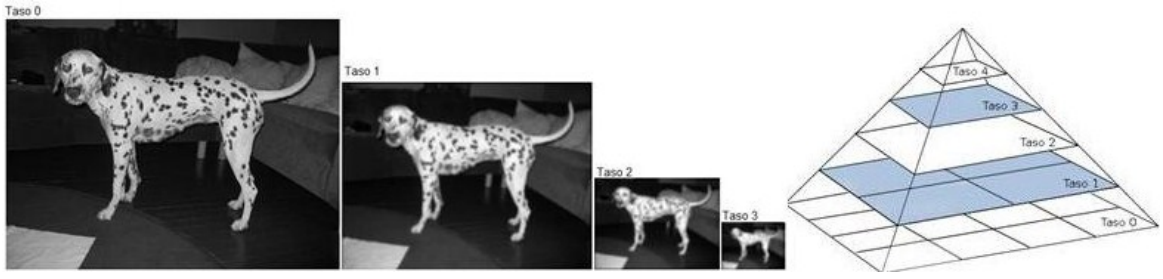
5.2.1 Hahmontunnistuskehys

VOC on monipuolinen hahmontunnistusmenetelmä, jonka vaiheiden ymmärtäminen on ollut haasteellista. VOCin teorian niin sanottua runkoa kutsutaan tässä insinöörityöraportissa hahmontunnistuskehykseksi, joka yhdessä opetus- ja tunnistusvaiheessa käytettävien algoritmien kanssa muodostavat VOCin. Tässä luvussa tarkastellaan VOCin hahmontunnistuskehyksen (jota kutsutaan tässä insinöörityöraportissa lyhyesti myös kehykseksi) pääpiirteitä.

Hahmontunnistuskehys rakentuu lineaarisista suodattimista *(filters)* sekä kuvapyramidin *(image pyramid)* ja kohteen piirrevektorit sisältävän piirrekartan *(feature map)* avulla muodostetusta piirrepyramidista *(feature pyramid)* [Felzenszwalb ym. 2010: 5].

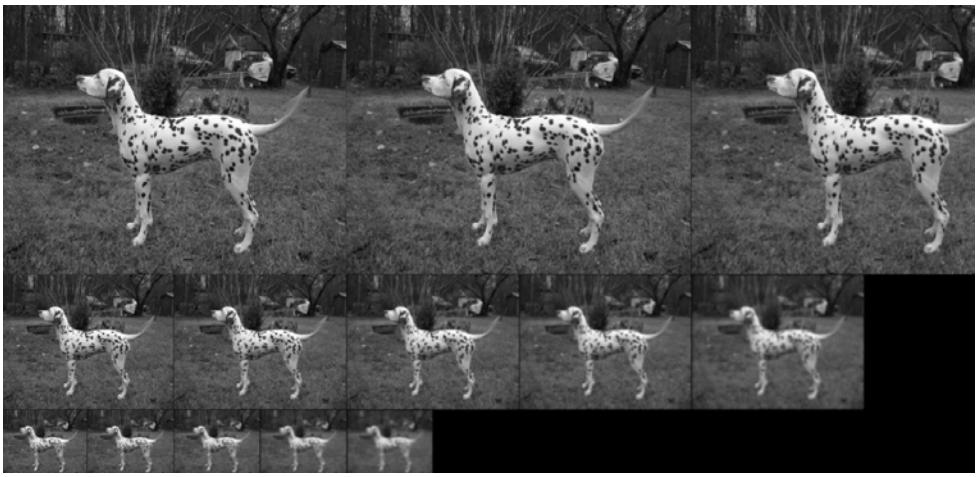
Kuvapyramidi

VOCin kehyksessä hyödynnettävä kuvapyramidi on tehokas tietorakenne. Kuvapyramidi sisältää kopioita alkuperäisestä kuvasta eli k-määrän kuvatasoja *(octave)*. Kuvapyramidissa kuvan pikselit ja resoluutio vähenevät asteittain ja symmetrisesti siirryttäessä kuvatasolta toiselle (kuva 18). Pikseleiden vähentyessä kuvan koko pienenee *(down-sampling)* ja resoluution pienentyessä kuvan yksityiskohdat sumenevat *(smoothing)*. Alkuperäinen kuva esitetään pyramidin juuressa ja siitä seuraavat kuvatasot tuotetaan rekursiivisesti antamalla edellisen tason kuva seuraavalle tasolle syötteenä. [Adelson ym. 1984: 34; Lindeberg 1994: 5.]



Kuva 18. Kuvassa oikealla havainnollistetaan kuvapyramidin ja sen tasojen rakennetta. Vasemmalla esimerkki toteutetusta kuvapyramidista, jossa neljä kuvatasoa: jokaisella tasolla (lukuun ottamatta alkuperäistä kuvaa) kuvan pikseleiden lukumäärää vähennetty ja kuvan resoluutio huonontunut. Lähdettä [Gonzales & Woods: 351] mukaillen.

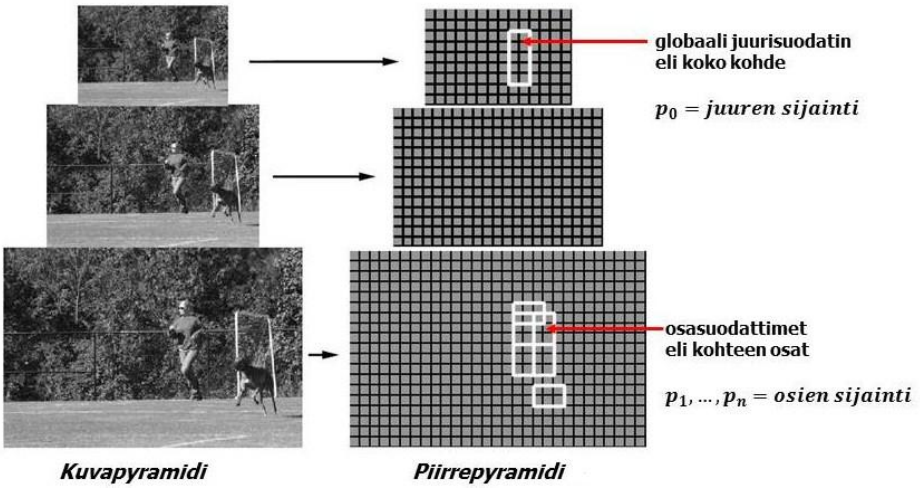
VOCin kuvapyramidin yhdellä kuvatasolla on useampi kuva. Yhden kuvatason sisältämien kuvien lukumäärää merkitään muuttujalla λ. VOCin opetusvaiheessa $\lambda = 5$ ja tunnistusvaiheessa $\lambda = 10$ [Felzenszwalb ym. 2010: 5]. Siirryttäessä kuvatasolta toiselle (pyramidissa ylöspäin) kuvan koko puolittuu, ja kuvatasoilla kuvakoon pysyessä samana, resoluutio vähenee. Kuva 19 havainnollistaa kolmea kuvatasoa, joista jokaisella on viisi kappaletta kuvia eli $\lambda = 5$ (lukuun ottamatta ylintä riviä, josta puuttuu kaksi kuvaa).



Kuva 19. Esimerkki kuvapyramidista, jossa resoluutio pienenee kuvatasoittain ja kuvan koko puolittuu siirryttäessä kuvatasolta toiselle. Ylinnä vasemmalla on alkuperäinen kuva. Jokaisella kuvatasolla pitäisi olla viisi kappaletta kuvia, mutta ensimmäisestä kuvatasosta (kuvassa ylinnä) puuttuu kaksi kuvaa.

Piirrekartta ja piirrepyramidi

VOC kehyksen rakentaminen alkaa kuvapyramidista luomisesta, jonka jälkeen kuvapyramidin kuvista muodostetaan piirrekartta. Käytännössä se tarkoittaa sitä, että jokaisesta kuvapyramidin kuvasta lasketaan piirteenirrotusalgoritmilla kohteen piirrevektorit, jotka kerätään alkioiksi piirrekartaksi kutsuttavaan matriisiin. Kuvassa 20 esiintyvä piirrepyramidi koostuu kuvapyramidin kuvista lasketuista piirrekartoista. [Felzenszwalb ym. 2010: 5.]



Kuva 20. Kuva havainnollistaa VOCin juuri- ja osasuodattimia, vasemmalla on analysoitavasta kuvasta rakennettu kuvapyramidi, jota käytetään piirrepyramidin muodostamiseen. Juurisuodatin sisältää koko kohteen ja osasuodattimet vastaavat kohteen osia. Lähdettä [Felzenszwalb ym. 2010: 5] mukaillen.

Suodattimet

Lisäksi hahmontunnistuskehykseen kuuluu suodattimet (kuva 20). Juurisuodatin *(root filter)* peittää koko kohteen ja osasuodattimet *(part filters)* kohteen osat. Juurisuodattimen sijainnin määrää tunnistuskehyksen koordinaatit. Osasuodattimet sijaitsevat piirrepyramidissa juurisuodatinta alempana, mikä tarkoittaa piirrekartan laskemisessa käytetyn kuvapyramidin kuvan olevan resoluutioltaan tarkempi verrattuna juurisuodattimen vastaavaan kuvapyramidin kuvaan. Felzenszwalbin mukaan se parantaa menetelmän suorituskykyä tunnistamisvaiheessa. [Felzenszwalb ym. 2010: 5.]

Pisteytysfunktio

Pisteytysfunktiota *(score of a hypothesis)* tarvitaan niin tunnistus- kuin opetusvaiheessa. Juurisuodattimille p_0 ja osasuodattimille $p_1, \ldots p_n$(n on osasuodattimien lukumäärä) lasketaan pistearvo *(score)* jokaisessa kuvan pisteessä sekä kuvapyramidin skaaloissa. Lasketut erilliset pistearvot summataan ja niille määräytyy niin sanottu deformaatiokustannus, joka riippuu jokaisen osasuodattimen suhteellisesta sijainnista juurisuodattimeen nähden. Pisteytysfunktio on muotoa

$$score(p_0, \ldots p_n) = \sum_{i=0}^{n} F_i' \cdot \phi(H, p_i) - \sum_{i=0}^{n} d_i \cdot \phi_d(dx_i, dy_i) + b\ , \qquad (38)$$

jossa suodattimien pistearvojen summa *(the data term)* on $\sum_{i=0}^{n} F_i' \cdot \phi(H, p_i)$, deformaatiokustannusten summa *(the spatial prior)* $\sum_{i=o}^{n} d_i \cdot \phi_d(dx_i, dy_i)$ ja b on bias-termi. [Felzenszwalb ym. 2010: 5.]

Suodattimen F ($w \times h$ kokoinen matriisi) pistearvo pisteessä p on

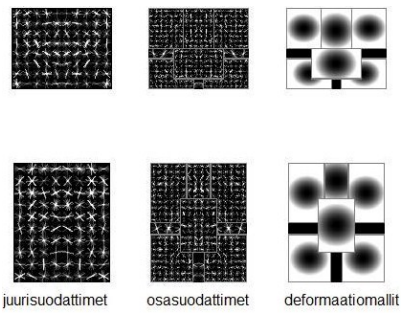
$$F' \cdot \phi(H, p, w, h)\,, \tag{39}$$

jossa F' on vaakavektori, joka on muodostettu suodattimesta F lukemalla matriisin F alkiot riveittäin ja asettamalla ne perätysten. Pystyvektori $\phi(H, p, w, h)$ koostuu piirrepyramidin H pisteessä p sijaitsevan $w \times h$ kokoisen ikkunan muokatuista HOG-piirteistä (katso luku 6.2) muodostettuna samaan tapaan kuin F' [Jäntti 2011]. Kaavan 39 piste $p = (x, y, l)$, jossa (x, y) määrittää sijainnin kuvassa ja l määrittää sijainnin piirrepyramidissa. [Felzenszwalb ym. 2010: 5.]

Mainittakoon vielä, että matemaattinen laskutoimitus kaavassa 39 on vaakavektorin F' ja pystyvektorin $\phi(H, p, w, h)$ pistetulo. Luvussa 4.2 kaavassa 25 esitelty diskriminanttifunktionhan muodostettiin painokerroinvektorin transpoosin w^T ja piirrevektorin x pistetulona, mikä vastaa samanlaista asettelua kuin kaavassa 39 esiintyy.

Komponenttimalli

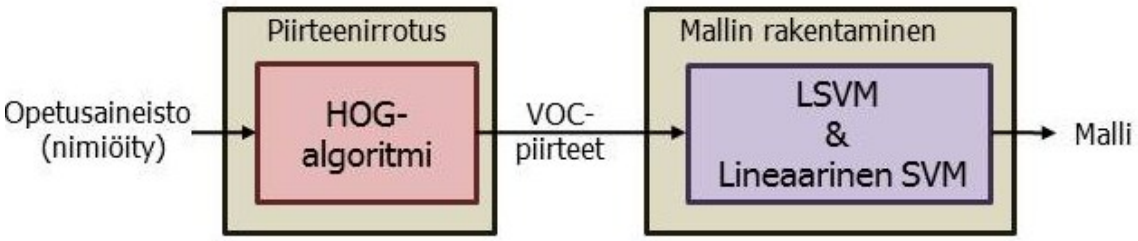
VOC komponenttimalli *(mixture model)*, joka huomioi kohteen eri kuvakulmat. Yhtä kuvakulmaa vastaa yksi komponentti. Esimerkkinä kaksi komponenttisesta mallista on kuva 21. Se kuvaa koiraa sivulta ja edestä katsottuna, lisäksi kuvasta 21 on nähtävissä juuri- ja osasuodattimet sekä deformaatiokustannus. Tarkempaa tietoa VOCin komponenttimallista lähteestä Felzenszwalb ym. 2010: 6–7.



Kuva 21. Tässä esimerkissä on kaksikomponenttisen koiran tunnistusmalli. Ensimmäiset kuvat havainnollistavat juurisuodattimia, joiden jälkeen osasuodattimet ja viimeisenä deformaatiokustannukset. Lähdettä [Felzenszwalb ym. 2009: visualizemodel.m] mukaillen.

5.2.2 Opetusvaihe

VOCin opetusvaiheessa käytettäviä algoritmeja havainnollistetaan kuvassa 22. Opetusvaihe koostuu opetusaineistolle tehtävästä piirteenirrotuksesta, josta saatavia VOC-piirteitä käytetään mallin rakentamiseen. Piirteenirrotus toteutetaan HOG-algoritmia muunnellen (luku 6.2) ja mallin rakentamiseen tarvitaan piilomuuttuja tukivektorikonetta *(latent support vector machine* eli *LSVM)* sekä lineaarista tukivektorikonetta *(linear SVM)*. Opetusvaiheen päätyttyä opetusaineistoa hyödyntämällä on muodostettu malli *(model)*, jota käytetään tunnistusvaiheen päätöksenteossa. [Felzenszwalb ym. 2010: 8, 12.]



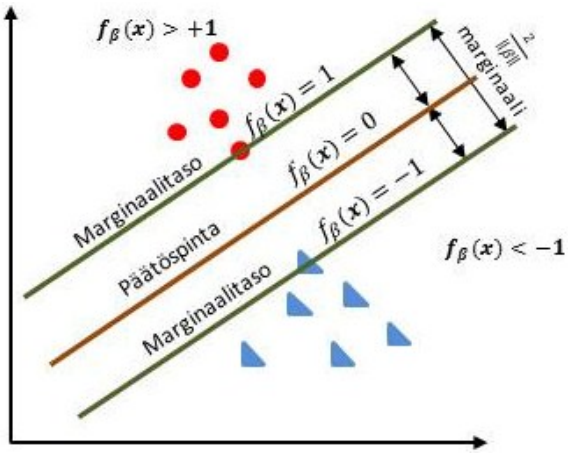
Kuva 22. VOCin opetusvaihe eli piirteenirrotuksessa ja mallin rakentamisessa käytetyt algoritmit.

Piilomuuttujat

Piilomuuttujat *(latent variables)* z ovat opetusvaiheessa vaikuttavia muuttujia, jotka auttavat löytämään parhaan lopputuloksen, mutta joiden todellista arvoa ei koskaan nähdä, ei edes opetusvaiheen aikana [LeCun ym. 2007: 211]. Piilomuuttujista voi lukea lisää lähteestä LeCun ym. 2007: 211–213. VOCissa kohteen osien eli osasuodattimien sijaintia käsitellään piilomuuttujina mallin rakentamisen aikana, koska osasuodattimien sijainteja ei ole nimiöity XML-annotaatioihin [Felzenszwalb ym. 2010: 2].

Tukivektorikone

Tukivektorikone muodostaa piirreavaruuden jakavan päätöspinnan siten, että sen kanssa yhdensuuntaisten *(orthogonal)* marginaalitasojen väliin jäävä marginaali on mahdollisimman suuri. Kuva 23 havainnollistaa kaksiluokkaista tapausta, missä piirreavaruus jaetaan kahteen päätösalueeseen. Marginaalitasoilla sijaitsevia piirrevektoreita sanotaan tukivektoreiksi *(support vectors).* [Schölkopf & Smola 2002: 11–17.]



Kuva 23. Esimerkki tukivektorikoneella löydetystä piirreavaruuden jakavasta päätöspinnasta, jossa funktio $f_\beta(x) = 0$, ja marginaalitasoista. [Schölkopf & Smola 2002: 11–17; SVM–Support Vector Machines, Optimum Separation Hyperplane]

Tukivektorikone ja VOC

VOCin artikkelin [Felzenszwalb ym. 2010: 2] merkintöjä käyttäen päätöspinnan $f_\beta(x) = 0$ sekä marginaalitasot $f_\beta(x) = 1$ ja $f_\beta(x) = -1$ toteuttaa funktio

$$f_\beta(x) = \beta \cdot \Phi(x)\,, \tag{40}$$

jossa β on suodatin, x on havainto eli kuva, $\Phi(x)$ on kuvasta laskettu piirrevektori ja laskutoimitus on suodattimen β ja piirrevektorin $\Phi(x)$ pistetulo. Kuvan 23 siniset kolmiot kuuluvat luokkaan 1, jolloin $y = -1$ ja kaavan 40 funktio $f_\beta(x)$ saa oikean päätöksen tapauksessa arvon $f_\beta(x) < 0$ ja väärän päätöksen tapauksessa $f_\beta(x) > 0$. Vastaavasti kuvan 23 punaiset pallot kuuluvat luokkaan 2, jolloin $y = 1$ ja kaavan 40 funktio $f_\beta(x)$ saa oikean päätöksen tapauksessa arvon $f_\beta(x) > 0$ ja väärän päätöksen tapauksessa $f_\beta(x) < 0$. [SVM–Support Vector Machines, Optimum Separation Hyperplane.]

Kaavassa 40 $f_\beta(x)$ riippuu lineaarisesti β:stä ja näin ollen opetusvaiheessa käytetään lineaarista tukivektorikonealgoritmia [Jäntti 2011]. Lineaarista tukivektorikonetta käytetään vain, kun $|Z(x_i)| = 1$ eli tapaukseen x_i liittyvien mahdollisten piilomuuttujien lukumäärän ollessa yksi. [Felzenszwalb ym. 2010: 8].

Piilomuuttujat z sisältyvät diskriminanttifunktioon

$$f_\beta(x) = \max_{z \in Z(x)} \{\beta \cdot \Phi(x, z)\}, \quad (41)$$

jossa $f_\beta(x)$:n riippuvuus β:sta on epälineaarinen [Felzenszwalb ym. 2010: 8; Jäntti 2011]. Epälineaarisuudesta johtuen lineaarisen tukivektorikoneen sijasta on menetelmäksi valittava piilomuuttuja tukivektorikone, joka esitellään seikkaperäisesti artikkelin [Felzenszwalb ym. 2010] luvuissa 4 ja 5.

Binäärisen luokitteluongelman ratkaiseminen tukivektorikoneella

Mallin parametrit β opetetaan nimiöidyllä opetusaineistolla $D = \{(x_1, y_1), \dots, (x_n, y_n)\}$, jossa $y_i \in \{-1,1\}$ ja $1 \leq i \leq n$. Opetusvaiheen päämääränä on minimoida kohdefunktio

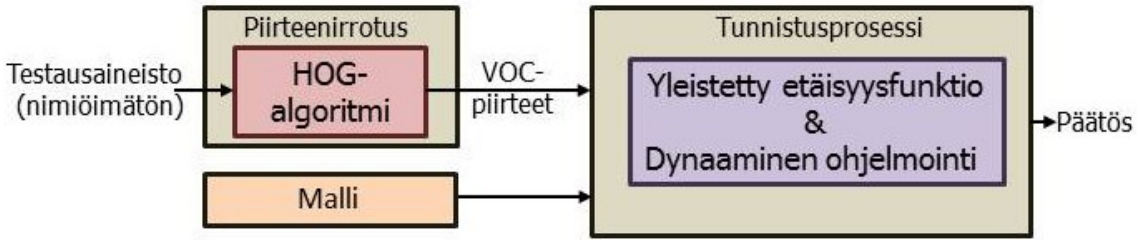
$$L_D(\beta) = \frac{1}{2}\|\beta\|^2 + C\sum_{i=1}^{n} max(0,1 - y_i f_\beta(x_i)) \ . \quad (42)$$

Kaavassa 42 oleva termi $\sum_{i=1}^{n} \max(0,1 - y_i f_\beta(x_i))$ viittaa luvussa 4.1.2 esitettyyn empiiriseen riskiin ja se sisältää hinge loss -kustannusfunktion (luku 4.1.1) sekä diskriminanttifunktion $f_\beta(x)$ kaavasta 41. Tästä voidaan päätellä VOC-teorian pohjautuvan diskriminatiiviseen todennäköisyysmalliin, koska sen kohdefunktio on (tässä insinöörityöraportissa esitetyn mukaan) saatu muodostamalla luokitteluongelmalle empiirinen riski [Felzenszwalb ym. 2010; Jäntti 2012]. Esitetty optimointiongelma on huonosti asetettu, siksi kaavan 42 kohdefunktio sisältää säännöllistämistermin $\frac{1}{2}\|\beta\|^2$ ja vakion C, joka säätelee säännöllistämistermin suhteellista painoarvoa. [Felzenszwalb ym. 2010: 8.]

5.2.3 Tunnistusvaihe

Tunnistusvaiheen algoritmit esitellään kuvassa 24. Kuten aikaisemmin tässä insinöörityössä on mainittu, niin nimiöimättömälle testausaineistolle tehdään opetusvaiheen piirteenirrotusta vastaavat vaiheet, jonka lopputuloksena saadaan HOG-piirteistä muokatut VOC-piirteet. VOC-piirteitä ja opetusvaiheessa muodostettua mallia tarvitaan tunnistusprosessissa, jonka keskeisiin menetelmiin kuuluvat yleistetty etäisyysfunktio *(generalized distance transforms)* ja dynaaminen ohjelmointi *(dynamic programming)*. Lisämateriaalia yleisestä etäisyysfunktiosta voi lukea lähteistä [Felzenszwalb & Hutten-

locher 2004; Felzenszwalb & Huttenlocher 2005: 17] ja dynaamisesta ohjelmoinnista lähteestä Bertsekas 2005: 2–51. [Felzenszwalb ym. 2010: 6.]



Kuva 24. VOCin tunnistusvaihe sisältäen piirteenirrotuksessa ja tunnistusprosessissa käytetyt menetelmät.

Kohteen tunnistaminen perustuu luvussa 5.2.1 esiteltyihin juuri- ja osasuodattimiin sekä niiden pistearvojen laskemiseen pisteytysfunktiolla (kaava 38). Korkeimman kokonaispistearvon saanut juurisuodatin valitaan tunnistuskehykseksi:

$$score(p_0) = \max_{p_1,\ldots,p_n} score(p_0, \ldots, p_n) \ . \tag{43}$$

Kokonaispistearvon lisäksi valintaan vaikuttaa osasuodattimien paras mahdollinen sijoittuminen, mikä ratkaistaan yleistetyllä etäisyysfunktiolla noudattaen dynaamista ohjelmointia. [Felzenszwalb ym. 2010: 6.]

Etäisyysfunktioita *(distance transforms)* käytetään hahmontunnistuksessa kuvien piirrevektoreiden vertailemiseen tunnistusprosessissa. Piirrevektoreiden vertailulla tehdään päätös, esimerkiksi sisältääkö havainto samoja piirrevektoreita, kuin mitä mallissa esiintyy. Binäärikuvassa (kaikkien pikseleiden arvo on joko 0 tai 1) etäisyysfunktio määrittelee jokaisen pikselin etäisyyden lähimpään pikseliin, jonka $arvo \neq 0$. VOCissa tunnistamisprosessissa käytetään yleistettyä etäisyysfunktiota, jossa binäärikuvan sijaan on niin sanottu oikea-arvoinen kuva. Binäärinen piirrekartta määrittelee jokaisessa pikselissä piirrevektorin olemassaolon tai poissaolon. Sen sijaan on hyödyllisempää määritellä kustannus piirrevektorille kuvan jokaisessa pikselissä, kuten kaavan 38 deformaatiokustannus tekee. Yleistetty etäisyysfunktio ottaa etäisyyksien määrittelemisen lisäksi piirteiden sijainnista aiheutuvat kustannukset (kutsutaan VOCissa deformaatiokustannuksiksi) huomioon. [Felzenszwalb & Huttenlocher 2004: 1–2; Felzenszwalb ym. 2010: 6.]

6 Piirteenirrotus

Tässä luvussa perehdytään piirteenirrotukseen tutkien kahta algoritmia. Ensimmäinen niistä on muun muassa ISM-menetelmässä käytetty Scale Invariant Feature Transform eli SIFT-algoritmi (luku 6.1) ja toinen on Histogram of Oriented Gradients eli HOG-algoritmi (luku 6.2), joka on muokattu VOCiin sopivaksi.

6.1 Scale Invariant Feature Transform -algoritmi

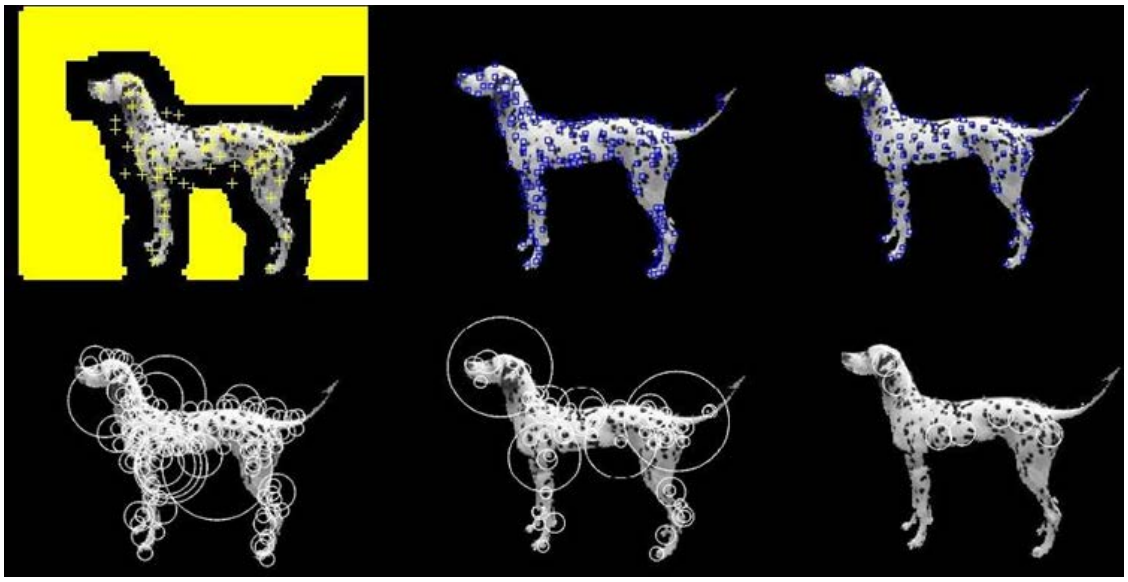
Avainpisteiden havaitsemisen ja piirteiden kuvaamisen yhteydessä esitetyt algoritmit ovat käytössä ISM-menetelmästä kertovassa artikkelissa [Leibe ym. 2008] sekä sitä vastaavassa Multi-Cue ISM 1.4.1 -sovelluksessa [Leibe 2008].

Kuvan paikalliset piirrekuvaajat lasketaan avainpisteiden avulla löydetyistä muuttumattomista avainpistealueista, kuten luvussa 5.1.2 määriteltiin. Piirrekuvaajien pitää olla yksilöllisiä ja samalla vakaita kuvakulman vaihteluille sekä mahdollisille avainpisteen havaitsijan tuottamille virheille. Niiden tulee kuvata löydettyjen kuvatilkkujen sisältö mahdollisimman tarkasti. [Mikolajczyk & Schmid 2003: 1.]

Yksinkertaisin piirrekuvaaja on kuvan pikseleistä muodostettu vektori, mutta tunnistustehtävässä laskennallinen kompleksisuus kasvaa sen ollessa moniulotteinen. Jakaumiin perustuvat piirrekuvaajat käyttävät hyödykseen histogrammia *(histogram)*, josta on nähtävissä kuvan pikseleistä laskettujen gradienttien *(gradients)* jakauma. Tämän typpisiin piirrekuvaajiin kuuluu Lowen [2004] kehittämä SIFT-algoritmi. [Mikolajczyk & Schmid 2005: 4.]

6.1.1 Avainpisteidenhavaitsija-algoritmeja

Kuva sisältää vakaita pisteitä, jotka pysyvät muuttumattomina kuvan skaalan vaihdellessa ja kuvakulman muuttuessa [Lowe 2004: 5]. Näiden avainpisteiden avulla hahmontunnistusjärjestelmä KOIRAKO voi tunnistaa testikuvassa olevan koiran, kun niitä verrataan ennalta muodostettuun koodikirjaan. Avainpisteidenhavaitsijat ovat algoritmeja, joiden tehtävänä on etsiä kuvasta vakaat avainkohdat.



Kuva 25. Avainpisteidenhavaitsija-algoritmien tuloksia kuvina. Ylärivi: DoG, SUSAN ja Harris. Alarivi: LoG, Harris-Laplace ja Gilles [Garcia 2007:test.m].

Havaitsija-algoritmeja on tarjolla monenlaisia, ja joitakin yleisimpiä on kokeiltu esimerkeissä käytettyyn opetuskuvaan. Kuvasta 25 on nähtävissä havaitut avainpisteet käyttäen algoritmeja DoG (Difference of Gaussian), SUSAN (Smallest Univalue Segment Assimilating Nucleus), Harris, LoG (Laplace of Gaussian), Harris-Laplace ja Gilles [Garcia 2007: test.m].

DoG-avainpisteidenhavaitsija-algoritmia on käytetty monissa hahmontunnistusprojekteissa [Lin ym. 2009: 1308; Leibe ym. 2008: 4; Seemann 2007: 71; Lowe 2004: 5]. Seuraavaksi tutustutaan algoritmiin tarkemmin.

6.1.2 Avainpisteiden havaitseminen: DoG-algoritmi

Tämä luku kuvaa DoG-algoritmin sekä avainpisteiden käsittelyn laaja-alaisemmin mukaillen David Lowen [2004] artikkelissa esitettyä SIFT-algoritmia, joka käsittää niin avainpisteiden havainnoinnin kuin piirteiden kuvaamisenkin. DoG-algoritmi on vasta osa avainpisteiden havainnointia. Sillä selvitetään avainpiste-ehdokkaat *(keypoint candidate)*, mutta sen lisäksi ehdokasjoukkoa on muokattava, jotta prosessin lopullinen päämäärä, eli kohteen tunnistaminen, onnistuisi parhaalla mahdollisella tavalla.

Avainpiste-ehdokkaiden määrittäminen

DoG eli Difference of Gaussian -funktio tunnistaa avainpisteet, jotka ovat muuttumattomia ottaen huomioon kuvan skaalan sekä kierron. DoG-funktio soveltaa skaala-avaruutta *(scale-space)* ja sieltä saadut minimi- ja maksimikohdat valitaan avainpiste-ehdokkaiksi. [Lowe 1999: 2.]

Seuraavaksi tarkastellaan prosessin matemaattista puolta [Lowe 2004: 5]. Skaala-avaruus määrittää kuvasta kohdat, jotka pysyvät muuttumattomina läpi kaikkien mahdollisten skaalojen. DoG-funktio hyödyntää skaala-avaruusteoriaa *(scale-space theory)* [Lindeberg 1994], jonka toteuttaa lauseke

$$L(x, y, \sigma) = G(x, y, \sigma) * I(x, y). \tag{44}$$

$L(x, y, \sigma)$ vastaa kuvan skaala-avaruutta ja σ-muuttuja vastaa kuvan skaalaa. Skaala-avaruus saadaan selville konvoloimalla kuva $I(x, y)$ Gaussin ytimellä $G(x, y, \sigma)$. Kaksiulotteisen Gaussin funktion *(gaussian function)* määrittää lauseke

$$G(x, y, \sigma) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}. \tag{45}$$

Kuvan avainpiste-ehdokkaat ovat funktion

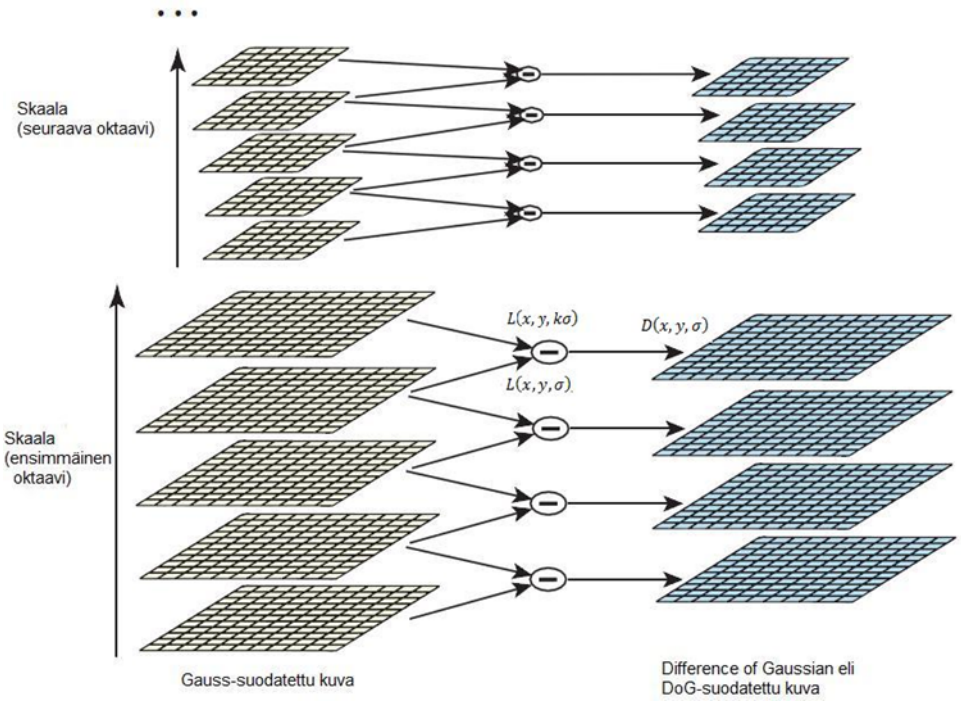
$$D(x, y, \sigma) = \big(G(x, y, k\sigma) - G(x, y, \sigma)\big) * I(x, y), \tag{46}$$

ääriarvoja *(extrema)*. Lauseke sisältää Gaussin funktioiden erotuksen eli DoG-funktion sekä DoG-funktion ja käsiteltävän kuvan $I(x, y)$ konvoluution.

Toisin sanoen ääriarvot eli kuvan minimi- ja maksimikohdat saadaan lausekkeesta

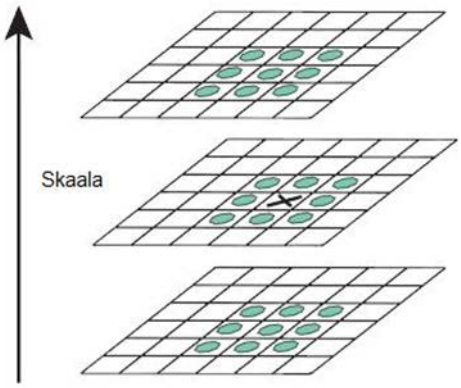
$$D(x, y, \sigma) = L(x, y, k\sigma) - L(x, y, \sigma), \tag{47}$$

joka laskee kahden lähekkäisellä skaalalla määritellyn kuvan $L(x, y, k\sigma)$ ja $L(x, y, \sigma)$ erotuksen, kuten kuva 26 havainnollistaa.



Kuva 26. Vasemmalla on Gaussin funktiolla konvoloidut skaala-avaruuden kuvat ja oikealla niistä saadut DoG-suodatetut kuvat $D(x, y, \sigma)$ [Lowe 2004: 6].

Skaala-avaruuden jokaiselle oktaaville eli kuvatasolle konvoloidaan alkuperäinen kuva Gaussin funktiolla (kaava 45), jotta saadaan kuvassa 26 esiintyvät vasemmanpuoleiset gauss-suodatetut kuvat. Kuvatason vierekkäiset gauss-suodatetut kuvat vähennetään toisistaan ja siten muodostetaan oikeanpuoleiset DoG-suodatetut kuvat. Jokaisen kuvatason jälkeen gauss-suodatettun kuvan pikselit vähennetään puoleen ja edellä mainittu prosessi toistetaan. Kaavoissa 46 ja 47 esiintyvä muuttuja k on skaalakerroin, jonka arvo on likimain yksi [Lowe 2004: 6–7].



Kuva 27. DoG-suodatetun kuvan ääriarvojen etsintä [Lowe 2004:7].

Seuraavaksi etsitään DoG-suodatetusta $D(x, y, \sigma)$ kuvasta paikalliset ääriarvot eli minimit ja maksimit. Mikä tarkoittaa sitä, että kuvassa 27 olevaa X-merkillä merkittyä pikseliä verrataan sen 26 naapuriin (kuvan 27 siniset ympyrät), joista 18 pikseliä on vierekkäisistä skaaloista ja kahdeksan pikseliä samasta skaalasta. Pikseli valitaan avainpiste-ehdokkaaksi, mikäli sen arvo on suurempi tai pienempi verrattuna sen naapureiden arvoihin.

Avainpiste-ehdokasjoukon karsinta

Edellä mainittujen menetelmien avulla kuvasta on löydetty avainpiste-ehdokkaat. David Lowe esittää artikkelissaan [Lowe 2004: 10–14] avainpiste-ehdokasjoukolle seuraavia toimenpiteitä:

- varmistetaan avainpisteiden oikea sijainti
- poistetaan alhaisen kontrastin omaavat avainpisteet
- poistetaan kohteen reunaviivoilla sijaitsevat avainpisteet.

Toimenpiteiden tarkoituksena on varmistaa lopulliset avainpisteet sekä niiden vakaus ja parantaa tunnistusprosessia. Avainpiste-ehdokasjoukolle suoritettavien toimenpiteiden matemaattiseen puoleen ei syvennytä tässä insinöörityöraportissa.

Avainpisteiden suunnan määritys piirrekuvaajia varten

Viimeisenä tehtävänä on laskea lopullisille avainpisteille tai tarkemmin sanottuna avainpisteiden ympärillä olevalle avainpistealueelle gradienttien histogrammi, jonka avulla avainpisteille määritellään suunta *(orientation)* [Lowe 2004: 13; Gonzales & Woods: 134–137; Jäntti 2009]. Avainpistealueen koko pikseleinä lasketaan seuraavasti

$$apAlue = (2 * round(3{,}5 * \sigma) + 1) * (2 * round(3{,}5 * \sigma) + 1), \qquad (48)$$

jossa $round(arg)$ pyöristää argumentin arg lähimpään kokonaislukuun ja σ-muuttuja vastaa skaalaa. Avainpistealueesta laskettua avainpisteen suuntaa tullaan käyttämään deskriptorin eli avainpisteestä muodostettavaan piirrekuvaajan määrittämiseen. Gradientti kertoo kuvan pikselin intensiteetin muutoksen suunnan, jossa muutos tapahtuu nopeinten ja gradienttien histogrammista nähdään kuvan avainpisteiden ympäristön pikseleistä laskettujen gradienttien jakauma.

Gradientti on vektori, jolla on suunta θ sekä magnitudi *(magnitude)* m ja Lowe laskee ne käyttäen kaavan 44 gauss-suodatettua kuvaa $L(x, y, \sigma)$ ottaen huomioon avainpisteen skaalan σ. Avainpistealueen kaikille pikseleille avainpisteen skaalassa σ lasketaan magnitudi yhtälöstä

$$m(x, y) = \sqrt{\big(L(x+1, y) - L(x-1, y)\big)^2 + \big(L(x, y+1) - L(x, y-1)\big)^2}, \quad (49)$$

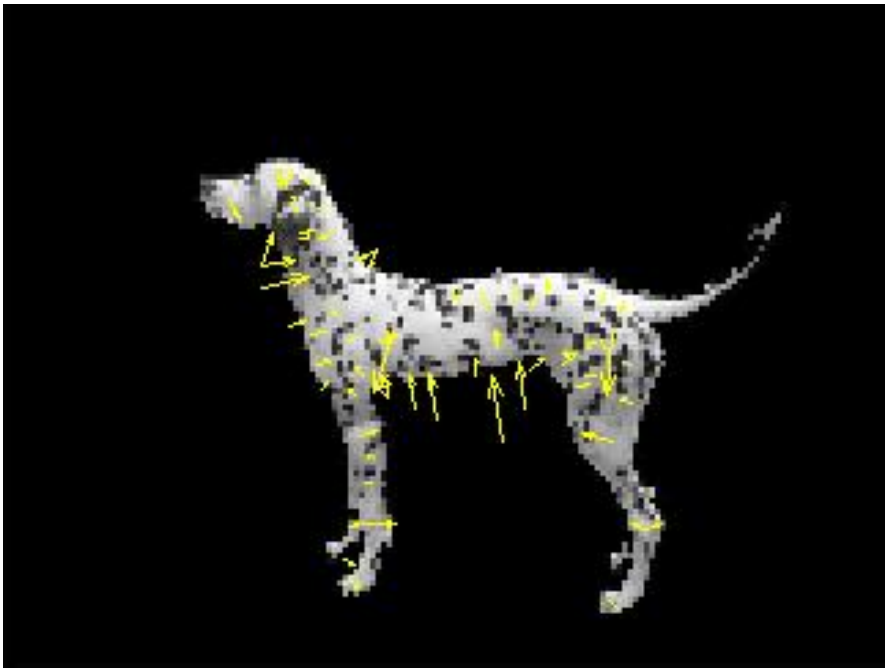
ja suunta yhtälöstä

$$\theta(x, y) = tan^{-1}\big(L(x, y+1) - L(x, y-1)\big)/\big(L(x+1, y) - L(x-1, y)\big). \quad (50)$$

Seuraavaksi muodostetaan suuntahistogrammi *(orientation histogram)* avainpistealueesta sisältävistä pikseleistä lasketuista gradientin suunnista (kaava 7). Histogrammin vaaka-akselina eli havaintojen arvona on suunta, joka on diskretoitu 10 asteen välein 360 asteesta 36:een ja pystyakselina eli havaintojen lukumääränä on avainpistealueen gradienttien suuntajakauma painotettuna gradienttien magnitudeilla ja kaksiulotteisella gaussin funktiolla, jonka hajonta σ on 1,5 kertaa avainpisteen skaala jaettuna oktaavin näytteistysluvulla.

Kaikki avainpistealueen pikselit käydään läpi ja näin saadaan täytettyä suuntahistogrammi. Jakaumahuippu eli suunta, jota on lukumäärältään ollut eniten avainpistealueessa vastaa avainpisteen vallitsevaa suuntaa *(dominant direction)*. Siihen verrataan muita löydettyjä suuntia ja kaikki suunnat, joita on ollut 80 % tai enemmän verrattuna jakaumahuippuun, merkitään avainpisteelle kuuluvaksi. Mikäli kyseiselle avainpisteelle tulee suuntahistogrammin perusteella enemmän kuin yksi suunta eli histogrammista löytyy jakaumahuipun lisäksi muita suuntia, luodaan ns. lisäavainpiste jokaiselle suunnalle erikseen ja sen sijainti sekä skaala määräytyvät alkuperäisestä avainpisteestä. Edellä mainitut toimenpiteet toteutetaan jokaiselle avainpisteelle. Avainpisteille on nyt

laskettu sijainti, skaala ja suunta (kuva 28), joita tarvitaan piirrekuvaajan muodostamiseen.

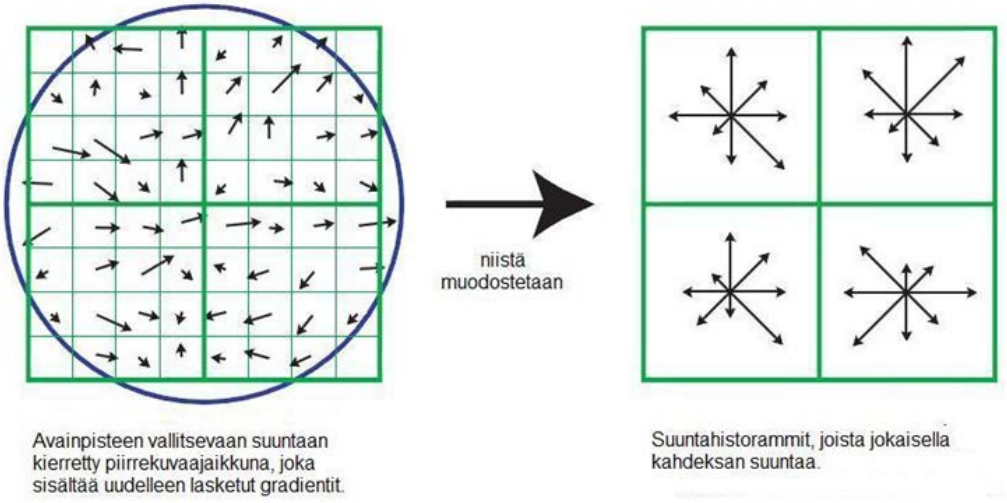


Kuva 28. DoG-algoritmilla lasketut lopulliset avainpisteet (keltaiset vektorit kuvastavat suuntaa ja skaalaa). Kuva avainpisteistä on saatu muokkaamalla Thomas El-Maraghin [2004: SIFTtutorial] MATLABilla ohjelmoimaa Lowen [2004] DoG-algoritmia.

Kuva 28 esittää DoG-algoritmin lopullisia avainpisteitä. Thomas El-Maraghi [2004: SIFTtutorial] on ohjelmoinut Lowen [2004] SIFT-algoritmin kokonaisuudessaan (sisältäen DoG-algoritmin) MATLAB-toteutukseksi. Se on hyvin kommentoitu ja helposti muokattavissa omiin käyttötarkoituksiin sekä sen avulla esimerkiksi DoG-algoritmin vaiheita on mukava tutkia. Seuraavassa luvussa perehdytään piirrekuvaajaan.

6.1.3 Piirrekuvaajien muodostaminen: SIFT-algoritmi

Piirrekuvaajan laskeminen aloitetaan edellisessä luvussa päädytystä tilanteesta. Suuntahistogrammin avulla selvitettyä avainpisteen vallitsevaa suuntaa käytetään kiertämään 16 x 16 pikselin kokoisen alueen eli piirrekuvaajaikkunan koordinaatistoa, jota tarvitaan piirrekuvaajan laskemiseen. Uuden kierretyn 16 x 16 piirrekuvaajaikkunan kierretyt pikselit eli näytepisteet eivät vastaa alkuperäisiä pikseleitä, koska näytepisteet eivät osu enää samoihin paikkoihin. Siitä syystä alkuperäiset pikseleiden arvotkaan eivät vastaa näytepisteiden arvoja. [Jäntti 2009; Lowe 2004: 15.]



Kuva 29. Esimerkki piirrevektorin muodostamisesta: Tässä esimerkissä kierretty piirrekuvaajaikkuna on 8 x 8 kokoinen näytepistejoukko ja siitä muodostettava piirrekuvaaja 2 x 2 kokoinen piirrematriisi. Todellisuudessa piirrekuvaaja on 4 x 4 kokoinen piirrematriisi, ja se lasketaan 16 x 16 kokoisesta piirrekuvaajaikkunasta. Lähdettä [Lowe 2004: 15] mukaillen.

Kierretyn piirrekuvaajaikkunan näytepisteille interpoloidaan intensiteetin arvot ja gradientti magnitudi m (kaava 49) sekä suunta θ (kaava 50) lasketaan uudestaan käyttäen näytepisteiden intensiteettien arvoja, mitä esittää kuvan 29 vasemmanpuoleinen ruudukko. Ruudukon solun sisällä oleva nuoli kuvaa gradienttia eli nuolen pituus vastaa magnitudia ja nuolen kärki osoittaa suuntaa. [Lowe 2004: 15–16; Jäntti 2010.]

Lisäksi jokaisen näytepisteen magnitudi painotetaan gaussin painofunktiolla, jonka koko saadaan laskettua kaavalla 48 ja jossa σ-muuttujan arvona käytetään puolta piirrekuvaajaikkunan leveydestä 16 x 16 näytepistettä, eli tällöin σ saa arvokseen 8. Toimenpiteen pyrkimyksenä on antaa vähemmän painoarvoa gradienteille, jotka sijaitsevat etäällä piirrekuvaajaikkunan keskustasta sekä välttää jyrkkiä muutoksia piirrekuvaajissa, joissa on pieniä vaihteluita ikkunan sijainnissa. Gaussin funktion painotusta havainnollistaa kuvaan 29 piirretty ympyränmuotoinen ikkuna. [Lowe 2004: 15–16; Jäntti 2010.]

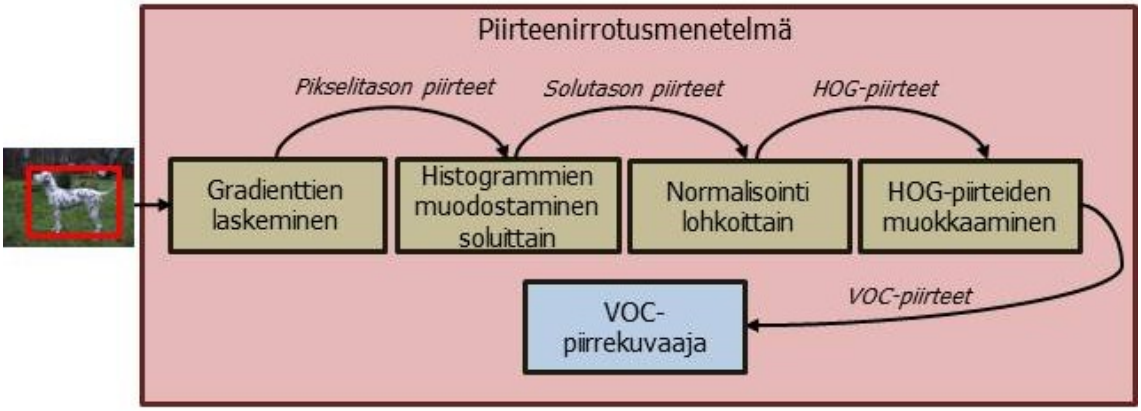
Kierretyn 16 x 16 -kokoisen piirrekuvaajaikkunan näytepisteiden gradienteista lasketaan suuntahistogrammit (kuva 29, oikeanpuoleinen ruudukko) siten, että jokainen muodostettu histogrammi sisältää gradientit 4 x 4 näytepisteen kokoisesta alialueesta. Tämä tarkoittaa sitä, että yhden avainpisteen kuvaamiseen tarvitaan yhteensä 16 suuntahistogrammia, jotka lasketaan avainpisteen vallitsevaan suuntaan kierretyn

piirrekuvaajaikkunan näytepisteiden gradienteista. Jokainen suuntahistogrammi sisältää kahdeksan suuntaa. Piirrekuvaaja on vektori, joka koostuu 128 elementistä eli 16 histogrammista ja niiden 8 suunnasta. [Lowe 2004: 15–16; Jäntti 2010.]

Seuraavaksi piirrekuvaaja normalisoidaan *(normalize)* yksikkövektoriksi *(unit vector)*, koska sillä se saadaan muuttumattomaksi valaistuksen affiineille muutoksille. Tämä tarkoittaa sitä, että muutos kuvan kontrastissa muuttaa jokaisen pikselin arvoa vakion $imCntrst$ verran ja muutos tapahtuu myös gradienteille muuttaen niiden arvoja saman verran. Normalisointi poistaa kontrastin muutokset, mutta se ei riitä poistamaan epälineaarisia valaistuksen vaikutuksia, jotka voivat johtua esimerkiksi kameran saturaatiosta. Se voi aiheuttaa suuria muutoksia joidenkin gradienttien magnitudeille, mutta vähemmän todennäköisesti gradienttien suunnille. Sen takia suurien gradienttien magnitudien vaikutus vähennetään asettamalla yksikkövektorin kynnysarvoksi 0,2, jonka jälkeen piirrevektori normalisoidaan uudestaan. Jokaisesta avainpisteestä muodostetaan 128-ulotteinen vektori eli piirrekuvaaja edellä esitettyjen vaiheiden mukaisesti. [Lowe 2004: 15–16; Jäntti 2010.]

6.2 VOC-menetelmän HOG-algoritmi

Histogram of Oriented Gradients eli HOG-algoritmi laskee kuvasta HOG-piirteet, joita muokataan ja jotka lopulta kerätään yhdeksi moniulotteiseksi VOC-piirrekuvaajaksi. SIFT-algoritmi (luku 6.1) laskee paikallisista avainpistealueista muodostettuja SIFT-piirrekuvaajia, kun taas HOG-piirteistä kootaan globaali piirrekuvaaja [Schiele 2010: 3]. Se tarkoittaa sitä, että SIFT-piirrekuvaaja sisältää tietoa vain yhdestä avainpiste-alueesta, eikä sen laskemiseen vaikuta muut avainpisteet. Sen sijaan VOC-piirrekuvaajan muodostamiseen on vaikuttanut koko kuvasta rajatun kohteen eli havaintoikkunan tietosisältö ja sen useat paikalliset toisiinsa kytköksissä olevat VOC-piirteet.



Kuva 30. VOCissa käytetyn HOG-algoritmin vaiheet [Dalal 2006: 22–23; Felzenszwalb ym. 2010: 12–14]

Dalalin esittämät HOG-piirteet ovat 36-ulotteisia, mutta VOC-menetelmässä käytetään 31-ulotteisia piirrevektoreita, sillä niiden on todettu parantavan tunnistusprosessin suorituskykyä verrattuna HOG-piirteisiin [Felzenszwalb ym. 2010: 12]. Kuvaan 30 on listattu piirrekuvaajan muodostamiseen käytetyn algoritmin vaiheet [Dalal 2006: 22–23; Felzenszwalb ym. 2010: 12–14].

Piirteiden nimeämisestä on mainittava sen verran, että kuten kuvasta 30 on nähtävissä, niin lopullinen piirrekuvaaja on nimetty etuliitteellä VOC. Tämä johtuu siitä, että tässä työssä painotetaan sitä, että alkuperäisiä HOG-algoritmilla luotuja HOG-piirteitä muokataan VOC-menetelmässä määritetyillä toimenpiteillä. HOG-piirteistä saadaan siis eri vaiheiden jälkeen VOC-piirteitä, jotka kootaan yhdeksi VOC-piirrekuvaajaksi. Felzenszwalbin ym. artikkelissa [Felzenszwalb ym. 2010] näistä piirteistä kerrotaan yleisesti nimellä HOG-piirteet, mutta tässä työssä on selkeämpää käyttää erilaista nimeämiskäytäntöä. Seuraavaksi esitellään VOC-piirrekuvaajan muodostamisen vaiheet.

6.2.1 Gradienttien laskeminen

HOG-piirteiden muodostaminen alkaa pikselitason piirteiden määrittämisellä, mikä tarkoittaa kuvan gradienttien suuntakulman ja magnitudin laskemista pikseleiden intensiteeteistä käyttämällä [-1, 0, 1] suodatinta x-akselin ja sen transpoosia y-akselin suuntaisesti [Dalal 2006: 37, 49; Felzenszwalb ym. 2010: 12]. Suodatinta tarvitaan osittaisderivaattojen approksimointiin, koska pikselit eivät ole jatkuvia muuttujia, kuten esimerkiksi funktion $f(x,y)$ muuttujat x ja y [Jäntti 2011]. Mikäli kuva on värillinen,

gradientti lasketaan jokaiselle värikanavalle erikseen ja valitaan niistä se, jonka gradientin neliö on suurin. SIFT-piirrekuvaajien määrittämiseen tarvitaan myös gradienttia ja laskukaavat löytyvät luvusta 6.1.2, magnitudi lasketaan kaavalla 49 ja suuntakulma kaavalla 50. Siitä huolimatta on tässäkin yhteydessä syytä tutustua kuvan pikselin gradientin laskemiseen.

Kuvan pikselin (x, y) intensiteetistä $I(x, y)$ lasketaan gradientti käyttäen osittaisderivaattaa x- ja y-suuntaan, jotka approksimoidaan numeerisesti edellä mainitulla [-1, 0, 1] suodattimella ja sen transpoosilla. $I(x, y)$:n osittaisderivaatta x-suuntaan on

$$\frac{\partial I(x,y)}{\partial x} = I(x+1, y) - I(x-1, y) \tag{51}$$

ja y-suuntaan

$$\frac{\partial I(x,y)}{\partial y} = I(x, y+1) - I(x, y-1). \tag{52}$$

Kaavat 51 ja 52 sisältävät suodatuksen ja osittaisderivaatan eli intensiteetin muutosnopeuden pikselissä (x, y) x- sekä y-suuntaan, joten gradientin arvo pikselissä (x, y) on

$$grad\big(I(x,y)\big) = \left(\frac{\partial I(x,y)}{\partial x}\right)\hat{e}_x + \left(\frac{\partial I(x,y)}{\partial y}\right)\hat{e}_y \tag{53}$$

jossa $\hat{e}_x$ on x-akselin ja $\hat{e}_y$ on y-akselin suuntainen yksikkövektori. Gradientin magnitudi saadaan yhtälöstä

$$|grad(I(x,y))| = \sqrt{\left(\frac{\partial I(x,y)}{\partial x}\right)^2 + \left(\frac{\big(\partial I(x,y)\big)}{\partial y}\right)^2} \tag{54}$$

ja sen suuntakulma

$$\varphi(x,y) = tan^{-1}\left(\frac{\frac{\partial I(x,y)}{\partial y}}{\frac{\partial I(x,y)}{\partial x}}\right). \tag{55}$$

Ensimmäiseksi lasketaan gradientin magnitudi. Tämän voi toteuttaa edellä esitellyllä kaavalla 54, mutta VOC-sovelluksessa se on toteutettu käyttäen suunnattua derivaattaa eli ottamalla derivaatta gradientin suuntaan, joka vastaa gradientin magnitudia. Derivaatta gradientin suuntaan saadaan hakemalla maksimiarvo kaavalla 56.

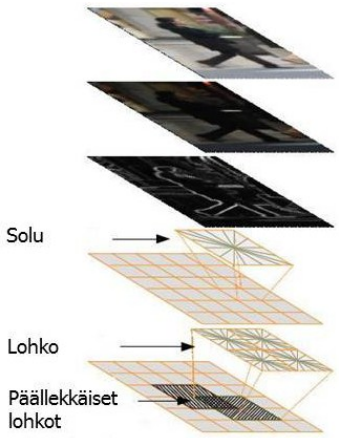
VOC-sovelluksen koodissa sitä vastaavaa muuttuja on nimeltään dot ja sen arvo saadaan yhtälöstä

$$dot = \frac{\partial I(x,y)}{\partial x} cos\big(\theta(x,y)\big) + \frac{\big(\partial I(x,y)\big)}{\partial y} sin\,\big(\theta(x,y)\big). \quad (56)$$

dotin arvo lasketaan 18 eri suuntaan eli 0 - 360 asteeseen 20 asteen välein ja valitaan niistä se, jossa muutosnopeus on suurin [Felzenszwalb ym. 2009: features.cc, rivit 97 – 107]. Maksimiarvon löytyessä täyttyy ehto $\varphi(x,y) = \theta(x,y)$, jossa φ on gradientilla laskettu suuntakulma ja θ on suunnatulla derivaatalla löydetty suuntakulma. Gradientin magnitudin $|grad(I(x,y))|$ (kaava 54), suuntakulman φ (kaava 55) ja derivaatan gradientin suuntaan eli dot (kaava 56) välisiä suhteita voidaan kuvata seuraavasti: Esitetään $dot(\theta(x,y))$ ja jos $\varphi(x,y) = \theta(x,y)$, niin siinä tapauksessa $dot = |grad(I(x,y))|$. [Jäntti 2011.]

6.2.2 Histogrammien muodostaminen soluittain

Toinen vaihe muistuttaa SIFT-piirrekuvaajan muodostamista, siinä kerätään pikselitason piirteet eli kuvan pikseleiden gradientit paikallisesti soluittain histogrammeihin. Tämän vaiheen lopputuloksena saadaan solutason piirre vektorina eli $C(i,j)$, jossa yhden vektorin sisältämä histogrammi kuvastaa yhden solun derivaatan suuntajakaumaa. Sen vaaka-akselilla on 20 asteen välein diskretoidut gradienttien suuntakulmat sekä pystyakselilla gradienttien magnitudiarvot.



Kuva 31. Kuvasta rajattu kohde eli havaintoikkuna jaetaan soluihin ja ne muodostavat lohkon [Dalal 2006: 23].

Havaintoikkuna (kuva 30) jaetaan soluihin, kuten kuvassa 31 esitetään. Solu koostuu pikseleistä ja niiden lukumäärä VOC-menetelmässä on 8 x 8 eli 64 pikseliä per solu [Felzenszwalb ym. 2009: featpyramid.m]. Kuvan koko on $w \times h$, solun sivun pituus k ja solutason piirrevektori $C(i,j)$, jossa $0 \leq i \leq \left\lfloor \frac{w-1}{k} \right\rfloor$ ja $0 \leq j \leq \left\lfloor \frac{h-1}{k} \right\rfloor$ [Felzenszwalb ym. 2010:12]. Jos $w = 64$, $h = 128$ ja $k = 8$, niin $i = \frac{64-1}{8} = 7{,}875 \sim 7$ ja $j = \frac{128-1}{8} = 15{,}875 \sim 15$ eli soluja i-suuntaan on 8 kappaletta ja j-suuntaan 16 kappaletta, joten tässä esimerkissä soluja on yhteensä 128.

Pikseleiden gradientit kootaan histogrammeihin ja se tapahtuu bilineaarisella interpoloinnilla [Gonzalez & Woods 2002: 273]. VOC-menetelmässä yhden pikselin gradientin magnitudi $|grad(I(x,y))|$ eli v vaikuttaa neljään solun histogrammiin seuraavasti. Se interpoloidaan neljään lähimpänä olevien solujen keskipisteisiin, mikä tarkoittaa sitä, että pisteessä (x,y) laskettu v saa uudet arvot v_1, v_2, v_3 ja v_4 neljässä eri pisteessä (x_1,y_1), (x_2,y_1), (x_1,y_2) ja (x_2,y_2), neljässä eri solussa s_1, s_2, s_3 ja s_4. Solusta s_1 muodostetaan histogrammi h_1, jonka ensimmäinen tapaus v_1 on pisteestä (x,y) pisteeseen (x_1,y_1) interpoloitu gradientin magnitudi v:n arvo. Huomattavaa on se, että v_1:llä on sama suuntakulma, kuin alkuperäisellä pikselin gradientin magnitudilla v sekä se, että v-muuttujan magnitudiarvo ei sisälly histogrammiin, vaan siihen kertyy vain interpoloidut magnitudiarvot. Samoilla toimenpiteillä kootaan solujen s_2, s_3 ja s_4 histogrammit h_2, h_3 ja h_4. [Jäntti 2011.]

6.2.3 Normalisointi lohkoittain

Kolmannessa vaiheessa solutason piirteestä $C(i,j)$ saadaan normalisoinnin jälkeen 36-ulotteinen HOG-piirre $H(i,j)$ ja edelleen yhden solun tietosisältö vastaa yhtä HOG-piirrettä [Felzenszwalb ym. 2010: 13]. Neljän solun soluryhmä muodostaa lohkon (kuva 31) ja yksi solu voi kuulua useampaan lohkoon, sillä havaintoikkunan lohkot menevät osittain päällekkäin.

Normalisointiarvot lasketaan yhtälöstä

$$N_{\delta,\gamma}(i,j) = (\|C(i,j)\|^2 + \|C(i+\delta,j)\|^2 + \|C(i,j+\gamma)\|^2 + \|C(i+\delta,j+\gamma)\|^2)^{\frac{1}{2}}, \quad (57)$$

jossa $\delta,\gamma \in \{-1,1\}$. Kaava 57 mittaa lohkosta niin kutsuttua gradienttienergiaa, jolla normalisoidaan solutason piirre eli $C(i,j)/\,N_{\delta,\gamma}(i,j)$ ja jonka maksimiarvo rajoitetaan operaattorilla T_α eli $T_\alpha\big(C(i,j)\big) = C(i,j)$, kun $C(i,j) \leq \alpha$ ja $T_\alpha\big(C(i,j)\big) = \alpha$, kun $C(i,j) > \alpha$. α:n rajoitusarvo on 0,2 ja sitä käytetään myös Dalalin HOG-menetelmän L2-Hys -normalisointimenetelmässä [Dalal 2006: 39] sekä Lowen SIFTissä [Lowe 2004: 16]. Gradienttienergiat määritellään neljän solun piirrevektoreista eli laskettavana olevasta solusta (piirrevektori $C(i,j)$) sekä sen kolmesta naapurisolusta ja näin ollen saadaan neljä normalisointiarvoa, joilla suoritetaan laskettavana olevan piirrevektorin $C(i,j)$ normalisointi maksimiarvoja rajoittaen. Laskutoimitusta kuvaa yhtälö

$$H(i,j) = \begin{pmatrix} T_\alpha\big(C(i,j)/N_{-1,-1}(i,j)\big) \\ T_\alpha\big(C(i,j)/N_{+1,-1}(i,j)\big) \\ T_\alpha\big(C(i,j)/N_{+1,+1}(i,j)\big) \\ T_\alpha\big(C(i,j)/N_{-1,+1}(i,j)\big) \end{pmatrix}, \quad (58)$$

jossa $H(i,j)$ on tämän vaiheen lopputulos eli solun (i,j) HOG-piirre [Felzenszwalb ym. 2010:13].

Havainnollistetaan kaavoissa 57 ja 58 käytettävien solujen piirrevektorien sijaintia suhteessa toisiinsa sijoittamalla kaavaan 57 $i = 1$, $j = 2$, $\delta = -1$ ja $\gamma = -1$,

$$N_{-1,-1}(1,2) = (\|C(1,2)\|^2 + \|C(1-1,2)\|^2 + \|C(1,2-1)\|^2 + \|C(1-1,2-1)\|^2)^{\frac{1}{2}} \quad (59)$$

$$= (\|C(1,2)\|^2 + \|C(0,2)\|^2 + \|C(1,1)\|^2 + \|C(0,1)\|^2)^{\frac{1}{2}}.$$

Jatketaan testaamista arvoilla $i = 1$, $j = 2$, $\delta = 1$ ja $\gamma = -1$ eli

$$N_{+1,-1}(1,2) = (\|C(1,2)\|^2 + \|C(2,2)\|^2 + \|C(1,1)\|^2 + \|C(2,1)\|^2)^{\frac{1}{2}}, \quad (60)$$

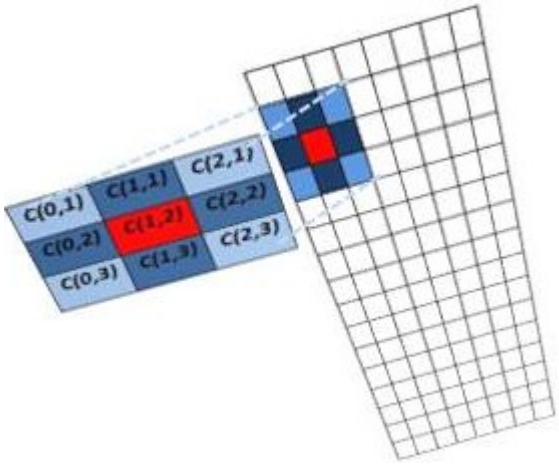
$i = 1$, $j = 2$, $\delta = 1$ ja $\gamma = 1$ eli

$$N_{+1,+1}(1,2) = (\|C(1,2)\|^2 + \|C(2,2)\|^2 + \|C(1,3)\|^2 + \|C(2,3)\|^2)^{\frac{1}{2}} \quad (61)$$

ja $i = 1$, $j = 2$, $\delta = -1$ ja $\gamma = 1$ eli

$$N_{-1,+1}(1,2) = (\|C(1,2)\|^2 + \|C(0,2)\|^2 + \|C(1,3)\|^2 + \|C(0,3)\|^2)^{\frac{1}{2}}. \quad (62)$$

Kuten kaavoista 59–62 ja kuvasta 32 voidaan todeta, jokainen normalisointiarvo lasketaan yhden lohkon piirrevektoreista ja piirrevektorin $C(1,2)$ HOG-piirteeseen $H(1,2)$ (kaava 58) vaikuttaa sen kahdeksan naapurisolua.



Kuva 32. Esimerkki soluihin jaetusta havaintoikkunasta, jossa kuvan koko 64 x 128 pikseliä ja solun sivun pituus 8 pikseliä, joten soluja 8 x 16 eli 128 kappaletta. Punaisella merkatun solun piirrevektorin $C(1,2)$ normalisointiin ja HOG-piirteen $H(1,2)$ laskemiseen vaikuttaa kahdeksan naapurisolun (neljän lohkon) piirrevektorit, jotka merkattu kuvaan sinisellä, niistä tummansiniset havainnollistavat päällekkäisiä soluja.

Tähän vaiheeseen esitelty HOG-piirre on 36-ulotteinen ja se vastaa Dalalin väitöskirjan HOG-algoritmilla [Dalal 2006:37–41] luotua HOG-piirrettä suuriltaosin. Seuraavaksi perehdytään sen jatkokäsittelyyn, joka on ominaista juuri VOCissa käytetylle HOG-algoritmille.

6.2.4 HOG-piirteiden muokkaaminen

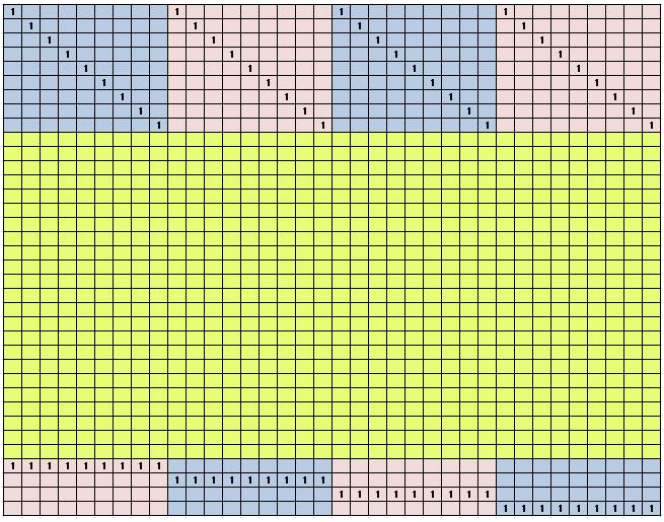
Viimeisessä vaiheessa tähdätään HOG-piirteen ulottuvuuden alentamiseen, joka tekee opetusvaiheessa luodusta mallista yksinkertaisemman sekä nopeuttaa opetus- ja tunnistusvaiheessa käytettäviä algoritmeja. Felzenszwalb on tutkinut HOG-piirteitä projektissaan ja on tehnyt niille pääkomponenttianalyysin eli PCAn [Jolliffe 2002; Shlens 2009], josta selvisi, että 36-ulotteiset HOG-piirteet voidaan esittää 11-ulotteisina säilyttäen saman suoristuskyvyn tunnistusvaiheessa. Artikkelissa kuitenkin todettiin, että PCA-menetelmällä toteutettu HOG-piirteen ulottuvuuden alentaminen on laskennallisesti raskas ja sen sijasta VOCissa käytetäänkin analyyttistä projektiota, jolla saadaan 36-ulotteista HOG-piirteistä 13-ulotteisia. [Felzenszwalb ym. 2010:13–14.]

Analyyttinen projektio on Felzenszwalbin VOCiin kehittämä menetelmä, joka on PCAta yksinkertaisempi ulottuvuuden alentamistekniikka [Felzenszwalb ym. 2010:14]. Sen käyttökelpoisuutta on perusteltu PCA-analyysilla, joka asettaa ulottuvuuksien minimirajaksi 11, koska sen alle mentäessä piirteistä katoaa oleellista tietosisältöä. Analyyttisella projektio -menetelmällä HOG-piirteistä saadaankin 13-ulotteisia, mikä on tarpeeksi lähellä PCAn antamaa lukua ja näin ollen käyttökelpoinen, lisäksi sen tunnistusvaiheen suorituskyky pysyy samana verrattuna niin 36-ulotteisiin HOG-piirteisiin kuin 11-ulotteisiin PCA-käsiteltyihin piirteisiin. [Felzenszwalb ym. 2010: 13–14; Jäntti 2012.]

VOC-artikkelissa [Felzenszwalb ym. 2010] kerrotaan sekä analyyttisen projektion että PCAn vaikutuksista 36-ulotteiseen HOG-piirteeseen, mutta käytännössä VOCissa on 108-ulotteisia piirrevektoreita, joista saadaan analyyttisella projektiolla 31-ulotteisia [Felzenszwalb ym. 2010: 14]. Niitä varten selvitetään käsitteet contrast sensitive- ja contrast insensitive -piirteet, jotka yksinkertaisuudessaan ovat kuvan gradienttien laskemisessa käytettyjä suuntakulmia. Kuten gradienttien laskeminen ja histogrammien muodostaminen soluittain -vaiheissa jo mainittiin, niin histogrammi sisältää kuvan solujen gradienttien jakauman ja sen muodostamiseen tarvitaan gradientin suuntakulmaa. Dalalin mukaan on kaksi vaihtoehtoa, histogrammin vaaka-akseli voidaan jakaa joko 0–360 asteeseen tai 0–180 asteeseen [Dalal 2006:38]. VOCissa HOG-piirteet lasketaan käyttäen molempia Dalalin mainitsemia suuntakulmavaihtoehtoja 20 asteen

välein, contrast sensitive -piirteitä eli $H_{s18}(i,j)$ on 18 kappaletta ja contrast insensitive -piirteitä eli $H_{s9}(i,j)$ 9 kappaletta [Felzenszwalb ym. 2010: 14].

Alkuperäinen 36-ulotteinen HOG-piirre esitetään yleensä 9×4 matriisina, joka sisältää kaavan 58 mukaan neljä normalisointia jokaista 9 suuntaa kohden eli perinteisessä tapauksessa käytetään vain $H_{s9}(i,j)$ piirteitä. VOCissa näiden edellä mainittujen kaavan 58 mukaan laskettujen piirteiden lisäksi lasketaan 18 suunnalla samaa kaavaa 58 käyttäen $H_{s18}(i,j)$ piirteet ja näin saadaan VOC-piirteet $F(i,j)$, jotka siis sisältävät $H_{s18}(i,j)$ sekä $H_{s9}(i,j)$ ja joilla on yhteensä 27 suuntaa sekä neljä normalisointia. VOC-piirteet ovat tässä vaiheessa 108-ulotteisia eli $4 * (9 + 18) = 108$. [Felzenszwalb ym. 2010: 13–14.]



Kuva 33. Esimerkki 36-ulotteisen HOG-piirteen projektiomatriisista, jossa näkyy missä kohdin matriisia ykköset sijaitsevat, tyhjät ruudut vastaavat nollia [Jäntti 2012]. HOG-piirteet kootaan pystyvektoriin ja kerrotaan se tällä matriisilla, siten piirteet saadaan projisoitua 13-ulotteisiksi.

Analyyttisella projektiomatriisilla kerrotaan 108-ulotteisista VOC-piirteistä $F(i,j)$ muodostettu pystyvektori ja siten saadaan alennettua VOC-piirteet $F(i,j)$ 31-ulotteisiksi vektoreiksi $G(i,j)$. Sitä havainnollistetaan kuvilla 34 ja 35, joissa uudet piirrekomponentit muodostetaan 36-ulotteisesta HOG-piirteestä sekä kuvalla 33, joka esittää 36-ulotteisen HOG-piirteen projektiomatriisia. Nämä esimerkit on tehty 36-ulotteisella HOG-piirteellä 108-ulotteisen VOC-piirteen sijaan, jotta esitys olisi selkeämpi. [Felzenszwalb ym. 2010:13–14.]

Uusi p1	Uusi p2	Uusi p3	Uusi p4	Uusi p5	Uusi p6	Uusi p7	Uusi p8	Uusi p9
H_{11}	H_{21}	H_{31}	H_{41}	H_{51}	H_{61}	H_{71}	H_{81}	H_{91}
H_{12}	H_{22}	H_{32}	H_{42}	H_{52}	H_{62}	H_{72}	H_{82}	H_{92}
H_{13}	H_{23}	H_{33}	H_{43}	H_{53}	H_{63}	H_{73}	H_{83}	H_{93}
H_{14}	H_{24}	H_{34}	H_{44}	H_{54}	H_{64}	H_{74}	H_{84}	H_{94}

$$\sum_{j=1}^{4} H_{ij}, 1 \leq i \leq 9$$

Kuva 34. Uudet piirrekomponentit lasketaan summaamalla vektoreita yhteen. Tässä esimerkissä summataan vektorit, joilla on sama suuntakulma ja saadaan yhteensä yhdeksän uutta piirrekomponenttia (p1–p9).

Kuvissa 34 ja 35 esiintyvä H_{ij} on HOG-piirre, jossa muuttuja i kuvaa suuntakulmaa ja muuttuja j normalisointikerrointa. $Uusi\,p1$ -niminen piirrekomponentti (kuva 34) on summa H_{11}-, H_{12}-, H_{13}- ja H_{14}-piirteistä, joissa on neljä erilaista normalisointikerrointa $1 \leq j \leq 4$ ja yksi kiinteä suuntakulma $i = 1$. Kertomalla HOG-piirteistä koottu pystyvektori projektiomatriisilla (kuva 33), lasketaan samalla tavalla summaamalla loput piirrevektorit eli yhteensä yhdeksän uutta piirrettä $p1$–$p9$. Piirrekomponentti $Uusi\,p10$ summataan kiinteällä normalisointikertoimella $j = 1$ muuttuvien suuntakulmien eli $1 \leq i \leq 9$ ylitse. Sama pätee piirteisiin $p11$–$p13$, kuvan 35 mukaisesti. Tällä tavalla laskettuja piirteitä on neljä kappaletta. Näin saadaan muodostettua $9 + 4 = 13$ uutta piirrekomponenttia $p1$–$p13$. [Felzenszwalb ym. 2010:13–14; Jäntti 2012.]

H_{11}	H_{21}	H_{31}	H_{41}	H_{51}	H_{61}	H_{71}	H_{81}	H_{91}	Uusi p10
H_{12}	H_{22}	H_{32}	H_{42}	H_{52}	H_{62}	H_{72}	H_{82}	H_{92}	Uusi p11
H_{13}	H_{23}	H_{33}	H_{43}	H_{53}	H_{63}	H_{73}	H_{83}	H_{93}	Uusi p12
H_{14}	H_{24}	H_{34}	H_{44}	H_{54}	H_{64}	H_{74}	H_{84}	H_{94}	Uusi p13

$$\sum_{i=1}^{9} H_{ij}, 1 \leq j \leq 4$$

Kuva 35. Tässä esimerkissä summataan vektorit, joilla on sama normalisointikerroin ja saadaan yhteensä neljä uutta piirrekomponenttia (p10–p13).

Edellä esitetyn analyyttisen projektion seurauksena VOC-piirteet koostuvat 31 piirrekomponentista, joista 27 vastaa suuntakulmia ja neljä kuvaa lohkon solujen gradienttienergiaa [Felzenszwalb ym. 2010: 14]. Tässä luvussa kerrottujen vaiheiden jälkeen

ne kootaan yhdeksi suureksi vektoriksi, jota tässä työssä kutsutaan VOC-piirrekuvaajaksi (kuva 30) [Felzenszwalb ym. 2010: 5].

7 KOIRAKO-sovelluksen toteutus

Tässä luvussa käsitellään luvussa 2 esiteltyyn määrittelyyn sekä sitä seuraavien lukujen teoriasisältöön pohjautuen toteutettua KOIRAKO-sovellusta. KOIRAKOn teknisessä toteutuksessa (luku 7.1) perehdytään: nimeämiskäytäntöihin (luku 7.1.1), esitellään KOIRAKOn rakenne ja pakettitasolta moduulitasolle siirtyen (luvut 7.1.2–7.1.4) ja lopuksi kuvataan KOIRAKOn käyttäytymistä käyttötapauksittain (7.1.5). VOC-sovelluksen käyttö ja sen kehitystyö sekä hyödyntäminen KOIRAKOssa esitellään luvussa 7.2, jonka jälkeen luodaan katsaus KOIRAKOn käyttämiseen (luku 7.3). Ajanpuutteen vuoksi sovellusta ei ole testattu, mutta joitakin sovelluksessa havaittuja virheitä ja puutteita luetellaan luvussa 7.4.

7.1 Tekninen toteutus

KOIRAKO on MATLABin päälle rakennettu hahmontunnistussovellus, joka on toteutettu MATLAB-ohjelmointikielellä. Sen kehitysympäristönä on käytetty Linuxille tarkoitettua 64-bittistä MATLAB-ohjelmiston 7.11.0 (R2010b) -versiota.

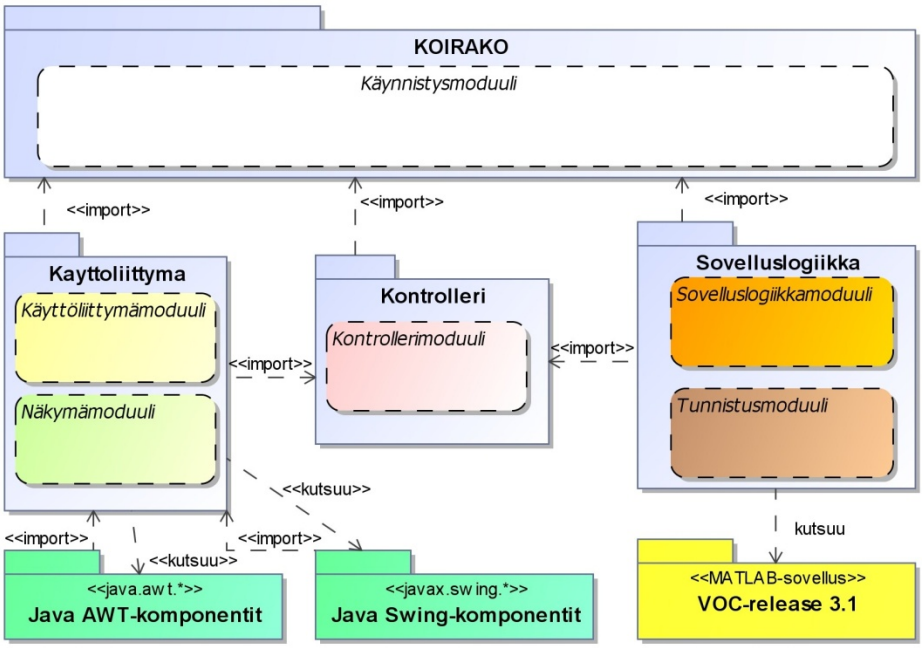
KOIRAKO on työasemasovellus, joka on suunniteltu MVC-arkkitehtuurin *(Model-View-Controller)* mukaisesti, eli se sisältää käyttöliittymän ja sovelluslogiikan sekä niitä yhdistävän kontrollerin, joka välittää tietoa käyttöliittymästä sovelluslogiikalle sekä päinvastoin. Tämän ansiosta sovelluksen rakenne on selkeä ja sen osat on pystytty ohjelmoimaan erikseen. Tämä on erityisen tärkeää, koska KOIRAKOn sovelluslogiikka hyödyntää VOC-sovellusta, joka on käytännössä erillinen sovelluspaketti.

KOIRAKO on suunniteltu oliopohjaisesti, mutta siihen kuuluu proseduraalisesti ohjelmoitu VOC-sovellus, jonka funktioita kutsutaan KOIRAKOsta. KOIRAKO-sovelluksen suunnitteluvaiheessa on pohdittu toteutukseen liittyviä ratkaisuja, joiden rajaavina tekijöinä ovat MATLAB-ohjelmassa toimiva VOC sekä Swing- ja Abstract Windows Toolkit-eli AWT-käyttöliittymäkirjastojen [The Java™ Tutorials: About the JFC and Swing;

Abstract Window Toolkit (AWT)] avulla toteutettu graafinen käyttöliittymä. Niiden perusteella KOIRAKOn tekniselle toteutukselle oli valittavana kaksi lähestymisvaihtoehtoa: joko kutsua Swing-komponentteja MATLAB-koodista tai kääriä VOC-sovelluksen koodi Java-luokkiin ja rakentaa sitä hyödyntävä sovellus kokonaan Javalla. Jälkimmäinen vaihtoehto on toteutettavissa MATLAB-ohjelmistoon liitettävällä MATLAB Builder JA:lla [MATLAB Builder JA for Java language–Deploy MATLAB code as Java Classes], jonka avulla MATLABilla tehdyt funktiot kääritään Java-luokkien sisään, jolloin ne käyttäytyvät kuin muutkin Java-luokat.

Tätä insinöörityötä toteutettaessa lähestymistavaksi valittiin Swing-komponenttien kutsuminen MATLAB-koodista, koska oli mielenkiintoista nähdä miten se käytännössä tapahtuu ja onko niiden käyttäminen yhtä vaivatonta kuin Javalla ohjelmoitaessa. MATLAB-funktioiden kutsumista Java-ohjelmasta ei tutkittu tarkemmin, koska erikseen hankittavaa MATLAB Builder JA -ohjelmistoa ei ollut mahdollista hankkia.

Kuva 36 havainnollistaa KOIRAKOn luokkien jakoa käyttöliittymä-, kontrolleri- ja sovelluslogiikkapaketteihin sekä sovelluksen käynnistävään käynnistyspakettiin. Lisäksi kuvasta on nähtävissä KOIRAKOon liittyvät ulkopuoliset komponentit: VOC-sovellus sekä Swing- ja AWT-käyttöliittymäkirjastot.



Kuva 36. KOIRAKO-sovelluksen paketti- ja moduulikaavio.

Seuraavaksi kuvataan KOIRAKOssa käytettyjä nimeämiskäytäntöjä, minkä jälkeen tutustutaan moduuleihin luokkatasolla. KOIRAKOn luokkakaavio on liitteessä 4 ja sen ohjelmakoodit löytyvät liitteistä 5–14.

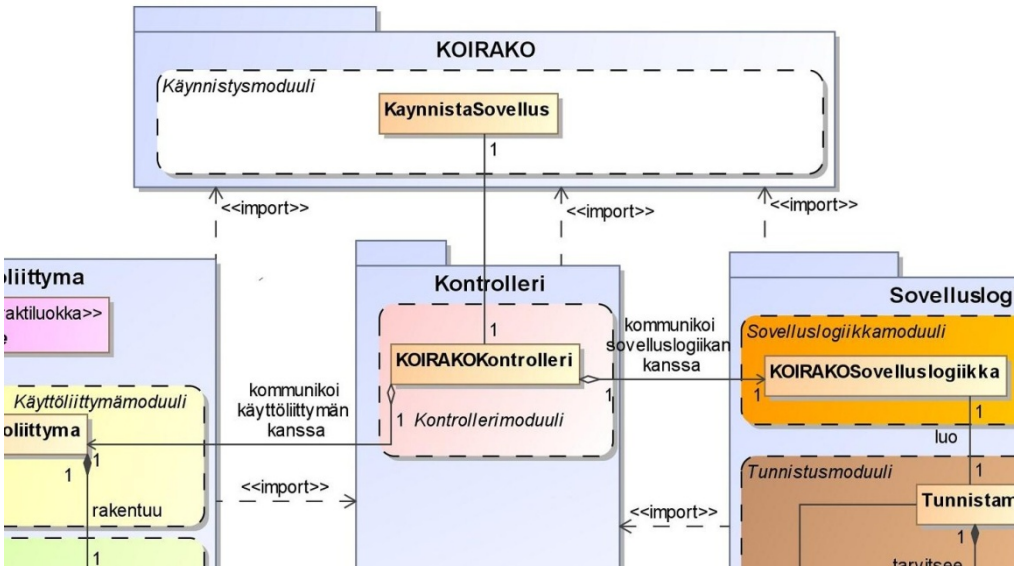
7.1.1 Nimeämiskäytännöt

MATLAB-ohjelmointikielessä ei määritellä muuttujille tietotyyppiä toisin kuin Javassa. Siitä syystä KOIRAKOn luokkakaaviossa on sovellettu UML:n syntaksia, jotta oleellinen tieto välittyisi dokumentin lukijalle parhaiten. Lisäksi muuttujat on pyritty nimeämään siten, että nimestä kävisi ilmi muuttujan tyyppi, sillä se helpottaa ohjelmoimista.

Edellä mainitusta esimerkkinä tekstiä sisältävät muuttujat, jotka on nimetty *muuttujannimiStr*. Swing-komponentit on nimetty *muuttujannimiBtn,* jossa Btn-pääte tarkoittaa JButton-komponenttia ja vastaavasti Lbl-pääte vastaa JLabel-komponenttia. Luokkakaavioiden muuttujiin on merkitty tietotyypit, mikäli muuttuja on toiseen luokkaan viittaava olio, Swing- tai AWT-komponentti. Tämä ei ole UML:n kuvauskieleen kuuluva rakenteellinen menettely, mutta se parantaa KOIRAKOn luokkakaavion tietosisällön välittymistä lukijalle.

7.1.2 KOIRAKO- ja kontrolleripaketit

KOIRAKO-paketin suhteita muihin paketteihin havainnollistaa kuva 37 (täydellinen luokkakaavio on liitteessä 4), siihen tuodaan muut paketit: Käyttöliittymä, Kontrolleri ja Sovelluslogiikka.

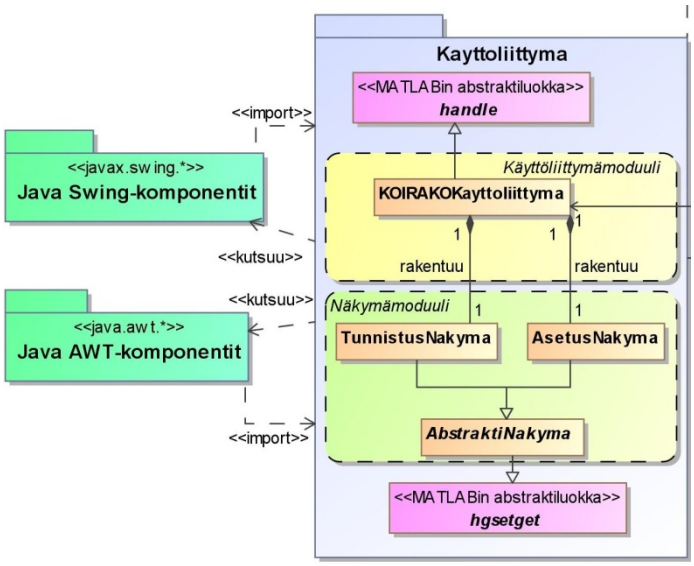


Kuva 37. KOIRAKO-paketti sisältää sovelluksen käynnistämiseen tehdyn luokan ja kontrolleripaketti vastaa tiedon välittämisestä käyttöliittymästä sovelluslogiikkaan ja päinvastoin.

Kontrollerimoduuli kommunikoi niin käyttöliittymä- kuin sovelluslogiikkamoduulin kanssa Sen päätehtävät ovat käyttöliittymämoduulin pyyntöjen välittäminen sovelluslogiikkamoduulin toteutettaviksi sekä päivityspyyntöjen lähettäminen käyttöliittymälle, mikäli sovelluslogiikalta toteutettavana olleet käyttöliittymän pyynnöt niin vaativat.

7.1.3 Käyttöliittymäpaketti

Käyttöliittymäpaketti (kuva 38) sisältää käyttöliittymä- ja näkymämoduulit ja siihen kuuluu MATLABin kaksi abstraktia luokkaa: *handle* ja *hgsetget.* Lisäksi käyttöliittymäpaketin luokat kutsuvat Swing- ja AWT-kirjastojen komponentteja.



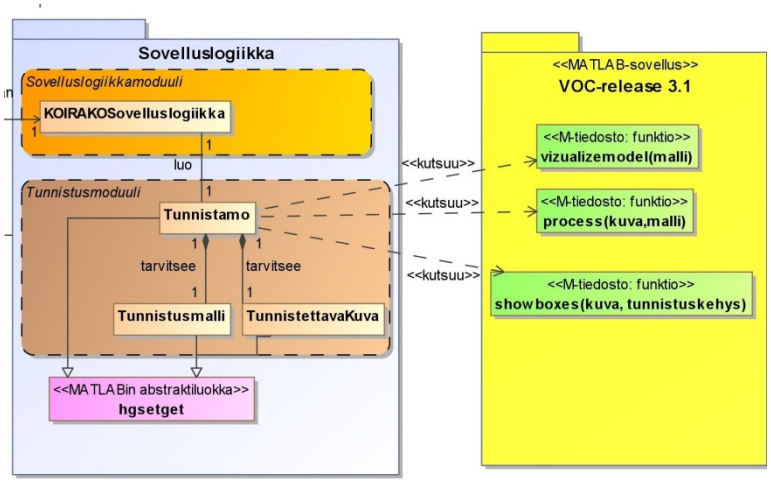
Kuva 38. Käyttöliittymäpaketti koostuu käyttöliittymä- ja näkymämoduuleista. Lisäksi käyttöliittymä- ja näkymämoduulin luokat kutsuvat Swing- ja AWT-kirjaston komponentteja.

Kuva 39 havainnollistaa KOIRAKOn käyttöliittymäpaketin luomaa pääikkunaa sovelluksen käynnistyttyä. Kuvaan on merkitty kaikki näkymän Swing-kirjaston JPanel-komponentit, joilla ne esiintyvät ohjelmakoodissa. Lisäksi kuvassa näkyy JMenuBar-komponentilla toteutettu valikkopalkki.

Kuva 39. Kuva KOIRAKOn aloitusnäkymästä, johon on merkattu käyttöliittymän sisältämät paneelit eli Swing-kirjaston JPanel-komponentit sekä JMenuBar-komponentti eli valikkopalkki.

7.1.4 Sovelluslogiikkapaketti ja VOC-sovellus

Kuvassa 38 esiintyvä sovelluslogiikkapaketti sisältää sovelluslogiikka- ja tunnistusmoduulit. Kontrollerimoduuli lähettää sovelluslogiikkamoduulille toimintapyyntöjä ja KOIRAKOSovellulogiikka-luokka hallinnoi tunnistusmoduulia Tunnistamo-luokan kautta.

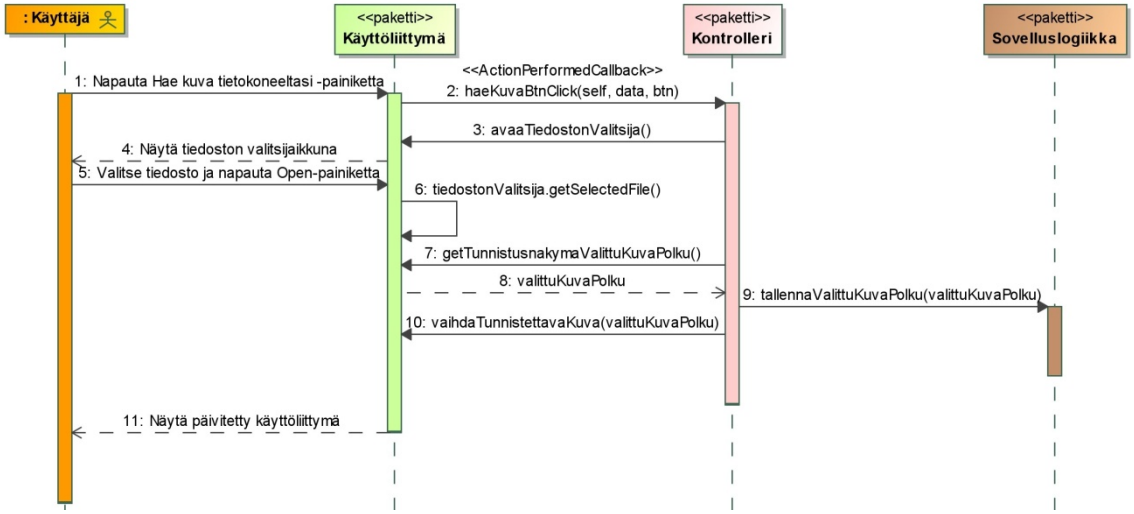


Kuva 40. Sovelluslogiikan moduuli ja luokat sekä VOC-sovelluksen funktiot.

Luokka Tunnistamo kutsuu KOIRAKOSovelluslogiikka-luokan pyynnöstä VOC-sovelluksen funktioita.

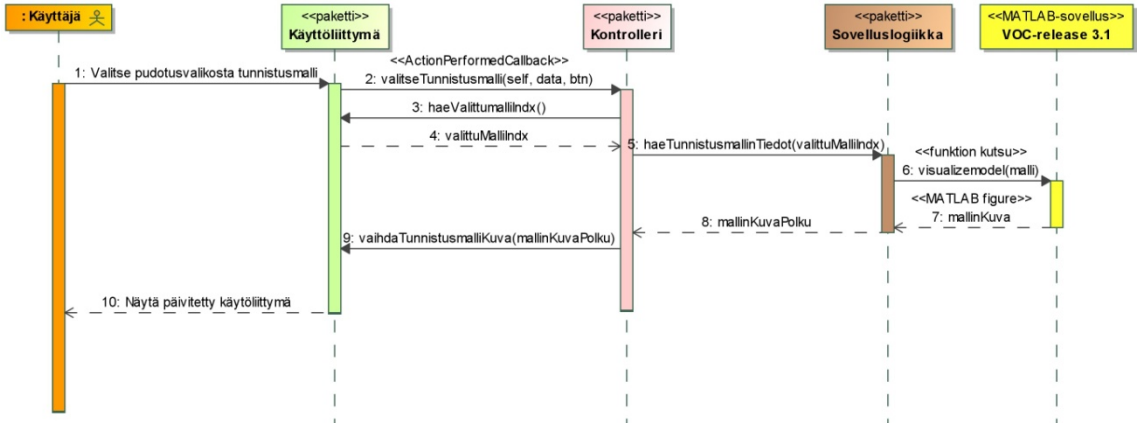
7.1.5 Sovelluksen käyttäytyminen

Tässä luvussa esitellään sekvenssikaaviot (kuvat 41–44) seuraaville käyttötapauksille: Valitse analysoitava kuva (K1), Valitse tunnistusmalli (K2), Analysoi kuva (K3) sekä Tallenna tunnistustulos (K4). Sekvenssikaaviot kuvaavat käyttäjän ja KOIRAKO-sovelluksen käyttöliittymän välistä vuorovaikutusta sekä siitä aiheutuvaa sovelluksen pakettien välistä kutsuliikennettä.



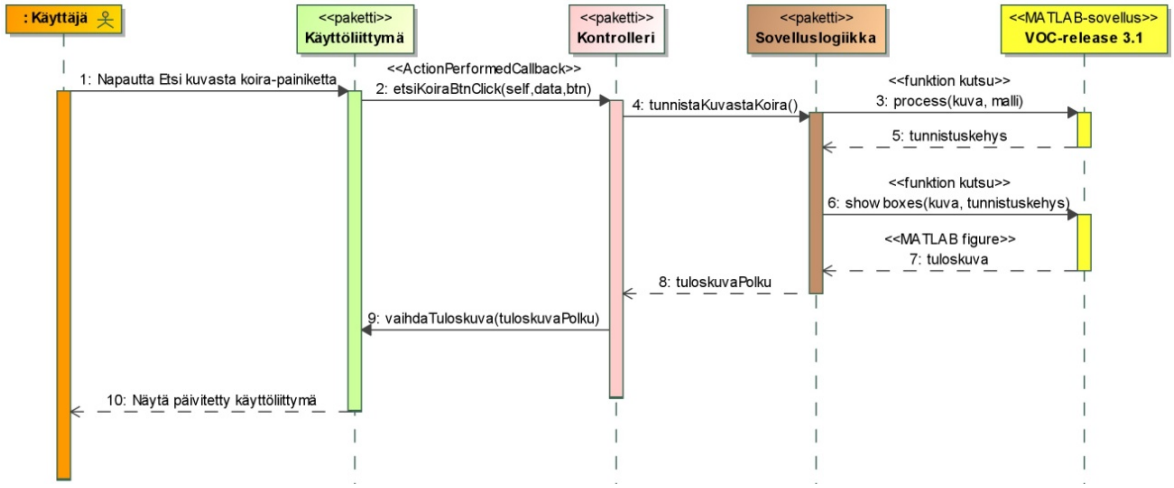
Kuva 41. Käyttötapauksen Valitse analysoitava kuva (K1) sekvenssikaavio.

Kuva 41 esittää taulukossa 1 kirjattuun käyttötapaukseen Valitse analysoitava kuva liittyvän sekvenssikaavion. Käyttötapaukseen (K1) osallistuvat käyttäjän lisäksi käyttöliittymä, kontrolleri ja sovelluslogiikka.



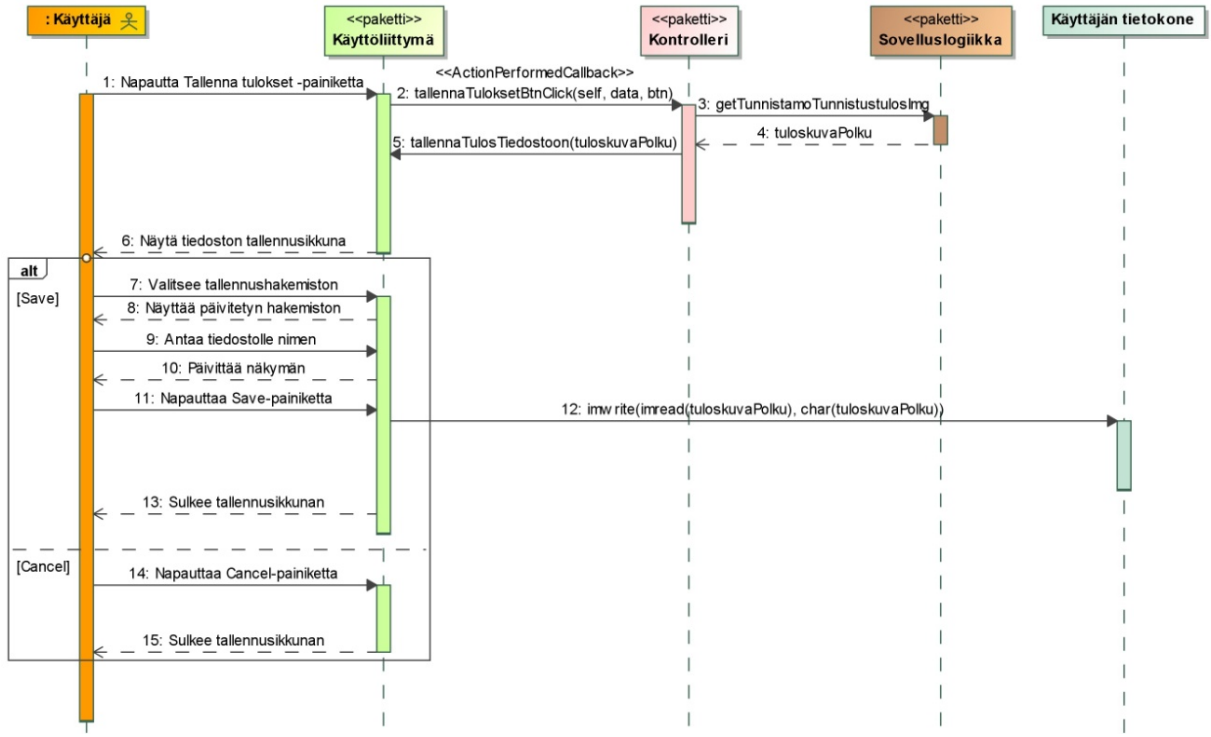
Kuva 42. Käyttötapauksen Valitse tunnistusmalli (K2) sekvenssikaavio.

Kuva 42 esittää taulukossa 2 kirjattuun käyttötapaukseen Valitse tunnistusmalli liittyvän sekvenssikaavion. Käyttötapaukseen (K2) osallistuvat käyttäjän lisäksi käyttöliittymä, kontrolleri ja sovelluslogiikka sekä VOC-sovellus.



Kuva 43. Käyttötapauksen Analysoi kuva (K3) sekvenssikaavio.

Kuva 43 esittää taulukossa 3 kirjattuun käyttötapaukseen Analysoi kuva liittyvän sekvenssikaavion. Käyttötapaukseen (K3) osallistuvat käyttäjän lisäksi käyttöliittymä, kontrolleri ja sovelluslogiikka sekä VOC-sovellus.



Kuva 44. Käyttötapauksen Tallenna tunnistustulos (K4) sekvenssikaavio.

Kuva 44 esittää taulukossa 4 kirjattuun käyttötapaukseen Analysoi kuva liittyvän sekvenssikaavion. Käyttötapaukseen (K3) osallistuvat käyttäjän lisäksi käyttöliittymä, kontrolleri ja sovelluslogiikka sekä mahdollisesti käyttäjän tietokone. Liite 3 sisältää KOIRAKO-sovelluksella simuloidut käyttötapaukset K1–K4.

7.2 VOC-sovellus

VOC-sovellus on toteutettu MATLAB-ohjelmalla ja se sisältää myös C/C++ -ohjelmointikielillä tehtyjä ohjelman osia, jotka parantavat suorituskykyä. Ohjelmisto on testattu Linux ja Mac OS X -käyttöjärjestelmissä. Mainittakoon vielä, että VOC-sovellukseen on rajoituksettomat käyttöoikeudet vedoten ohjelmiston COPYING-tiedostoon [Felzenszwalb ym. 2009: COPYING].

Kuva 45. Onnistunut tunnistustapaus: koira tunnistettu, sen ympärille on piirretty tunnistuskehys. Alkuperäinen kuva lähteestä [Jäntti].

VOC-sovellus tunnistaa kohteita edellä mainituista kohdeluokista. Sovelluksessa voidaan käyttää valmiiksi opetettua tunnistusmallia tai vaihtoehtoisesti käynnistetään opetusvaihe, jonka loputtua sovellus on luonut tunnistusmallin opetettavasta kohteesta. Tunnistusmallia käytetään tunnistusprosessissa. Onnistuneessa tunnistamistapauksessa sovellus piirtää tunnistuskehyksen havaitun kohteen ympärille, kuten kuva 45 havainnollistaa.

7.2.1 Käyttö

VOC-sovellus [Felzenszwalb ym. 2009] ei sisällä erillistä käyttöliittymää vaan ohjelmaa käytetään MATLABin komentokehotteesta käsin. Pakattu sovellus ladataan Felzenszwalbin www-sivuilta <http://people.cs.uchicago.edu/~pff/latent-release3/> ja se pure-

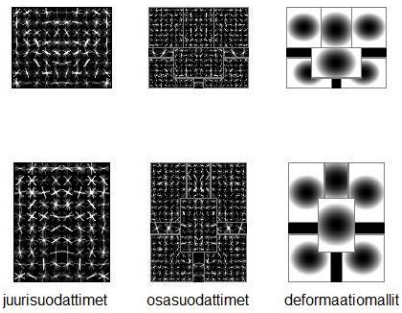
taan omavalintaiseen hakemistoon. Sen jälkeen käynnistetään MATLAB, jonka työhakemistoksi asetetaan jakelupakettia purettaessa luotu hakemisto voc-release3.1. Seuraavaksi suoritetaan MATLABin komentoikkunassa *compile.m*-tiedosto, joka kääntää VOCin C++-kieliset apufunktiot. Hakemistosta voc-release3.1 löytyy README-tiedosto, joka sisältää ohjeet VOCin käyttöön. Valmiiksi luodut tunnistusmallit löytyvät hakemistosta voc-release3.1/VOC2007 [Felzenszwalb ym. 2009: README.]

Opetusvaihe

VOCissa [Felzenszwalb ym. 2009: README] tunnistusmallin luominen toteutetaan seuraavasti:

1. Lataa ja pura tiedostot 2007 PASCAL VOC devkit ja test data VOC 2007 kilpailun [Everingham ym. 2007] www-sivuilta hakemistoon voc-release3.1.
2. Muokkaa tiedoston globals.m sisältämien muuttujien *cachedir, tmpdir* ja *VOCdevkit* hakemistopolut vastaamaan omaa hakemistorakennettasi.
3. Käännä kääntäjällä g++ *learn.cc* - Linuxin komentotulkissa: *g++ -03 -o learn learn.cc*
4. Käynnistä MATLAB.
5. Suorita *compile* MATLABin komentoikkunasta.
6. Aloita tunnistusmallin luominen funktiolla *pascal*. Esimerkki kaksikomponenttisesta koiran tunnistusmallin opettamisesta: *pascal('dog', 2);.*

Opetusvaiheen päätyttyä VOC-sovellus on tehnyt tiedoston *dog_final.mat*, joka ladataan tunnistusprosessissa MATLABiin.



Kuva 46. VOC-sovelluksen funktio *visualizemodel* palauttaa tunnistusmallin kuvan. Tässä esimerkissä on kaksikomponenttisen koiran tunnistusmalli. Lähdettä [Felzenszwalb ym. 2009: visualizemodel.m] mukaillen.

VOCin funktio *visualizemodel* palauttaa parametrina annettua tunnistusmallia havainnollistavan kuvan (kuva 46). Tätä funktiota kutsutaan KOIRAKOsta, kun käyttäjä valitsee tunnistusmallin (käyttötapaus Valitse tunnistusmalli (K2)).

Tunnistusvaihe

Kuvassa 47 on esimerkki VOCin käytöstä tunnistusvaiheessa. Kuvassa olevat komennot kirjoitetaan MATLABin komentoikkunaan: Tunnistusmalli ja analysoitava kuva ladataan MATLABiin *load*- ja *imread*-funktioilla, jossa funktio *load* lataa ja tallentaa tunnistusmallin *model* nimiseen structure-tyyppiseen muuttujaan. Tunnistusprosessi käynnistetään *process*-funktiolla, jolle annetaan parametreina sekä analysoitava kuva, tunnistusmalli että kynnysarvo *(threshold)*. Funktio palauttaa havaittuja kohteita vastaavien tunnistuskehyksien koordinaatit ja kokonaispistearvot (katso luku 5.2.3) muuttujaan *bbox*. Tunnistustuloksen näyttää *showboxes*-funktio, jonka parametreina ovat tunnistuskehykset sisältävä *bbox*-muuttuja sekä analysoitava kuva. [Felzenszwalb ym. 2009: README, process.m, detect.m, showboxes.m.]

```
esimerkki:
> load VOC2007/dog_final.mat;   % koiran tunnistusmalli, opetettu PASCAL 2007 opetusaineistolla
> im = imread('000034.jpg');    % testikuva
> bbox = process(im, model, 0); % käynnistää tunnistusprosessin
> showboxes(im, bbox);          % näyttää tunnistustuloksen
```

Kuva 47. Esimerkki VOC-sovelluksen käytöstä: Käyttäjä valitsee mallin ja analysoitavan kuvan, sovellus käynnistää tunnistusprosessin ja palauttaa tunnistustuloksen kuvana. Lähdettä [Felzenszwalb ym. 2009: README] mukaillen.

Funktio *process* kutsuu funktioita *detect*, *getboxes*, *nms* ja *clipboxes,* joiden suorittamisen jälkeen tunnetaan analysoitavan kuvan tunnistuskehys tai -kehykset. Funktio *detect* sisältää varsinaisen tunnistusprosessin ja palauttaa matriisin, jonka neljä ensimmäistä saraketta sisältävät tunnistuskehysten koodinaatit ja viimeisestä sarakkeesta selviää kokonaispistearvo (katso luku 5.2.3). Kynnysarvo rajaa tunnistusprosessissa löydetyt kehykset kokonaispistearvojen perusteella siten, että *detect*-funktion palauttamasta matriisista ei löydy annettua kynnysarvoa pienempiä kokonaispistearvoja.

7.2.2 Kehitystyö – VOC-sovelluksen testaaminen

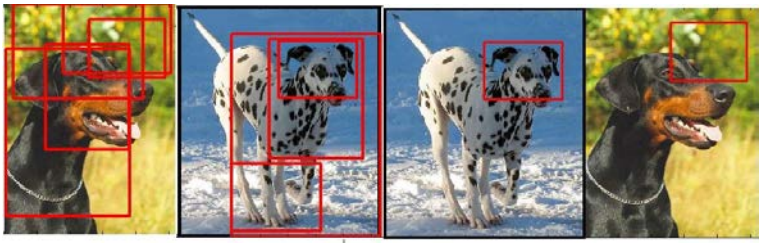
Tässä luvussa kuvataan tämän insinöörityöprojektin vaihe, jossa tutkittiin VOC-sovelluksen käyttökelpoisuutta yhden tämän insinöörityön tavoitteen saavuttamisessa. Tutkimus toteutettiin empiirisesti eli kokeiltiin käytännössä, mitä VOC-sovelluksella voi tehdä ja olisiko se hyödynnettävissä KOIRAKOn sovelluslogiikan pohjana.

VOC-sovellusta testattiin erilaisilla tunnistusmalleilla kutsuen *process* -funktiota parametrien arvoja vaihdellen. Hahmontunnistusmenetelmän valinta tehtiin perustuen kokemusperäiseen testaustyöhön ja siitä saatuihin tunnistustuloksiin. Tunnistusmalleja kokeiltiin erilaisille testausaineistolle aluksi pienellä kuvamäärällä (noin 20 kuvan sarjoja), vaihdellen muun muassa kuvan kokoa ja tunnistusmallin sisältämää kynnysarvoa, koska niin sanotuilla perusasetuksilla VOC ei löytänyt testikuvissa olevia koiria. Kokeilutyön kautta VOCin ohjelmakoodi tuli tutuksi ja tunnistustulosten parantuessa löydettiin ratkaisu, jolla päästiin koirien tunnistamisen kannalta mahdollisimman hyvään lopputulokseen.

Löydettyä ratkaisua voidaan arvostella siitä, että se on tehty ilman teoreettista tietämystä VOC-menetelmästä. VOC-menetelmä sisältää insinööriopintoihin sisältyvään kurssitarjontaan nähden vaikeita matemaattisia yksityiskohtia, joten tämän insinöörityön laajuudessa koiran tunnistamiseen liittyviä ongelmia ei pystytty ratkaisemaan teorialähtöisesti. Vasta VOC-sovelluksen varmistuttua käyttökelpoiseksi KOIRAKOssa, ryhdyttiin selvittämään perusteellisesti VOCiin liittyvää teoriaa. Lisäksi jokaisessa tunnistustesteissä käytetyssä kuvassa oli koira, eikä VOC-sovellusta testattu muunlaisilla kuvilla.

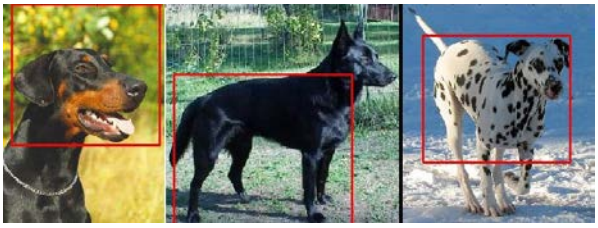
18.2.2011 Ensimmäiset tunnistustestit

Kohdeluokasta koira luotiin kaksi- ja neljäkomponenttiset tunnistusmallit. Opetusaineistona käytettiin VOC 2007 -kilpailumateriaalia [Everingham ym. 2007] (käytetty kaikissa tunnistusmallien opetusvaiheissa), joka sisältää kuvat ja niiden XML-annotaatiot. Ensimmäisissä tunnistustesteissä ei saatu toivottua tunnistustulosta eli muuttuja *bbox,* joka sisältää havaitut kohteet, palautti tyhjän matriisin ja täten tunnistustuloskuva oli alkuperäinen, eikä sisältänyt tunnistuskehystä. Funktiota *process* testattiin eri kynnysarvoilla, jotka olivat väliltä $-1{,}04 \leq kynnysarvo \leq 0$.



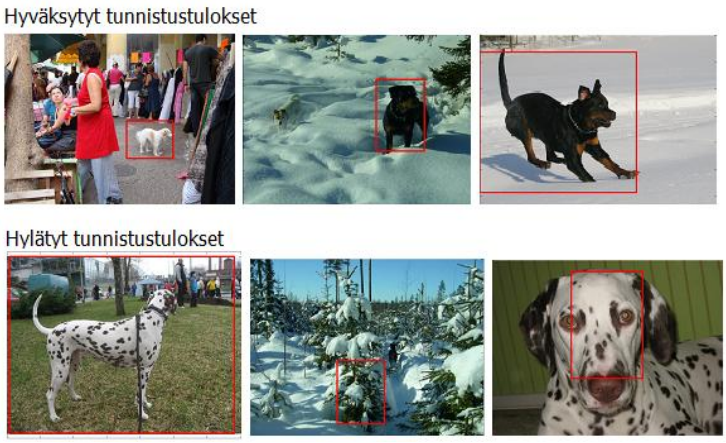
Kuva 48. Tunnistustuloksia 18.2.2011 kaksikomponenttisella tunnistusmallilla eri kynnysarvoja käyttäen.

Kuvissa 48 ja 49 on ensimmäisiä tunnistustuloksia. Niiden $kynnysarvo < 0$. Kaksikomponenttisella (kuva 48) tunnistusmallilla saadut tulokset sisälsivät enemmän kehyksiä verrattuna neljäkomponenttisen (kuva 49) tunnistusmallin tuloksiin.



Kuva 49. Tunnistustuloksia 18.2.2011 neljäkomponenttisella tunnistusmallilla.

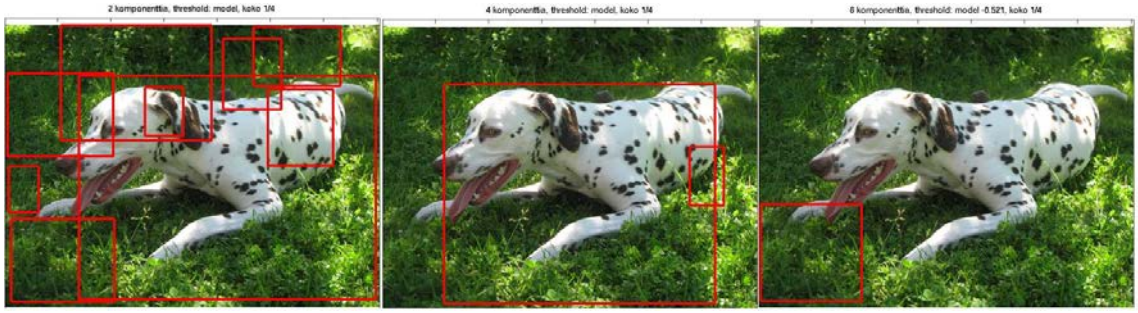
Hyväksytyksi tunnistustulokseksi määriteltiin sellainen tunnistuskehys, joka peittää kohteesta silmämääräisesti vähintään 70 %. Lähes koko kuvan rajaava tunnistuskehys luokiteltiin hylätyksi tunnistustulokseksi. Esimerkkejä hyväksytyistä ja hylätyistä tunnistustuloksista on kuvassa 50.



Kuva 50. Esimerkkejä hyväksytyistä ja hylätyistä tunnistustuloksista. Alkuperäiset kuvat lähteistä [Jäntti; Könönen; Kainulainen; Kallio].

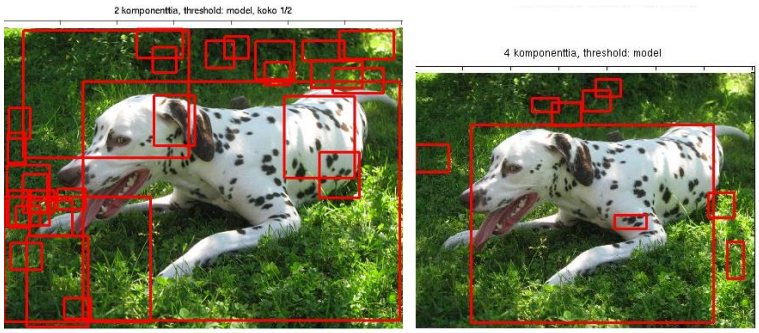
21.2.2011 Tunnistustestejä: kuvan koko, kynnysarvo ja tunnistusmalli

Seuraavaksi tutkittiin analysoitavan kuvan koon, kynnysarvon ja tunnistusmallin vaikutuksia tunnistustuloksiin.



Kuva 51. Tunnistustuloksia 21.2.2011. Kuvan koko ¼ alkuperäisestä ($1200\ x\ 901$ pikseliä) ja kynnysarvo tunnistusmallista. Vasemmalta lueteltuina: kaksikomponenttinen (mallin kynnysarvo $-1{,}0343$), neljäkomponenttinen (mallin kynnysarvo $-0{,}7669$) ja kuusikomponenttinen (mallin kynnysarvo $-0{,}521$).

Kuva 51 sisältää tunnistusprosessin tuloksia tunnistustesteistä, joissa kuvan koko oli ¼ alkuperäisestä kuvasta ($1200\ x\ 901$ pikseliä) ja kynnysarvona käytettiin tunnistusmallin sisältämää arvoa, joka kaksikomponenttisessa mallissa on $-1{,}0343$, neljäkomponenttisessa $-0{,}7669$ ja kuusikomponenttisessa $-0{,}521$. Kuva 52 havainnollistaa tunnistustestejä, joissa kuvan koko puolitettiin alkuperäisestä kuvasta ja tunnistusmallien (testattu vain kaksi- ja neljäkomponenttisilla tunnistusmalleilla) kynnysarvot olivat samat kuin kuvassa 51 käytetyissä tunnistusprosesseissa.



Kuva 52. Tunnistustuloksia 21.2.2011. Kuvan koko ½ alkuperäisestä (1200 x 901 pikseliä) ja kynnysarvo tunnistusmallista. Vasemmalta lueteltuina: kaksikomponenttinen (mallin kynnysarvo -1,0343) ja neljäkomponenttinen (mallin kynnysarvo: -0,7669).

Kuvan koon vaikutuksen huomaa, kun vertaa kuvasarjojen 51 ja 52 kuvia. Suuremmat kuvat (kuva 52) sisältävät huomattavasti enemmän tunnistuskehyksiä, minkä lisäksi kuvakoon kasvaminen pidentää tunnistusprosessin kestoa. Tästä tehtiin johtopäätös,

että analysoitavan kuvan koko muutetaan vastaamaan likimain opetusaineiston sisältämien kuvien kokoa (noin $500x500$ pikseliä). Vaikka kuvan koon suurentuessa analysoitavaan kuvaan tuli enemmän kehyksiä, oli suurin kuvista löytyvä tunnistuskehys samanlainen molemmissa kuvissa (vertaa kuvien 51 ja 52 neljäkomponenttisia tunnistustuloksia).

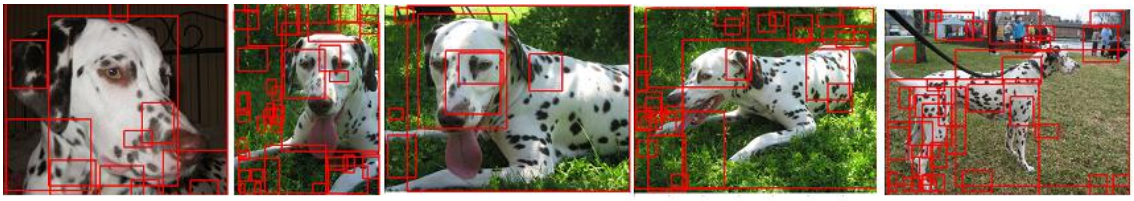
Nro	Pvm	Tunnistus-mallin kompo-nenttien lkm	Analysoitavan kuvan koko alkuperäisestä	Kynnysarvo	Tunnistuskehyksiä sisältävien kuvien lkm	Hyväksytty tunnistus lkm (100% = 20 kpl)	Kommentit	Hyväksytyt tunnistukset %
1	21.2.2011	2	1/4	0	0	0	Tunnistuskehyksiä 0 kpl	0 %
2	21.2.2011	2	1/4	-0,5	0	0	Tunnistuskehyksiä 0 kpl	0 %
3	21.2.2011	2	1/4	Malli: -1,0343	20	15	Tunnistuskehyksiä 1-14kpl/tulos	75 %
4	21.2.2011	2	1/2	0	0	0	Tunnistuskehyksiä 0 kpl	0 %
5	21.2.2011	2	1/2	-0,5	0	0	Tunnistuskehyksiä 0 kpl	0 %
6	21.2.2011	2	1/2	Malli: -1,0343	20	16	Tunnistuskehyksiä 8-40 kpl/tulos	80 %
7	21.2.2011	3	1/4	0	0	0	Tunnistuskehyksiä 0 kpl	0 %
8	21.2.2011	3	1/4	-0,25	1	1	Tunnistuskehyksiä 1kpl/tulos	5 %
9	21.2.2011	3	1/4	-0,5	3	2	Tunnistuskehyksiä 1kpl/tulos	10 %
10	21.2.2011	3	1/4	-0,75	17	7	Tunnistuskehyksiä 1-5kpl/tulos	35 %
11	21.2.2011	3	1/4	Malli: -0,9721	19	15	Tunnistuskehyksiä 1-14kpl/tulos	75 %
12	21.2.2011	4	1/4	0	0	0	Tunnistuskehyksiä 0 kpl	0 %
13	21.2.2011	4	1/4	-0,5	15	5	Tunnistuskehyksiä 1-3kpl/tulos	25 %
14	21.2.2011	4	1/4	Malli: -0,7669	19	13	Tunnistuskehyksiä 1-4kpl/tulos	65 %
15	21.2.2011	4	1/2	0	0	0	Tunnistuskehyksiä 0 kpl	0 %
16	21.2.2011	4	1/2	-0,5	17	5	Tunnistuskehyksiä 1-5kpl/tulos	25 %
17	21.2.2011	4	1/2	Malli: -0,7669	20	9	Tunnistuskehyksiä 1-20kpl/tulos	45 %
18	21.2.2011	6	1/2	0	3	1	Tunnistuskehyksiä 1 kpl/tulos	5 %
19	21.2.2011	6	1/2	-0,25	4	3	Tunnistuskehyksiä 1 kpl/tulos	15 %
20	21.2.2011	6	1/2	-0,75	20	7	Tunnistuskehyksiä 1-9kpl/tulos	35 %
21	21.2.2011	6	1/2	Malli: -0,7669	16	6	Tunnistuskehyksiä 1-3kpl/tulos	30 %

Kuva 53. Yhteenveto 21.2.2011 suoritetuista testaustuloksista, joissa kaikissa testausmateriaalina 20 kappaletta kuvia koirasta ja alkuperäisen kuvan koko on lähellä 1200 x 901 pikselin kokoluokkaa.

Tunnistustestejä tehtiin yhteensä 21 eri kokoonpanolla, joissa jokaisessa käytettiin samaa 20 kuvan sisältämää testiaineistoa [Könönen 2009–2012]. Testien tulokset esitetään kuvassa 53. Prosentuaalisesti parhaimmat tulokset (hyväksyttyjen tunnistusten lukumäärä/testikuvien lukumäärä) 65–80 % saatiin testeistä 3, 6, 11 ja 14, ja jokaisessa vaihtoehdossa käytettiin tunnistusmallin sisältämää kynnysarvoa. Testissä 14 on kuitenkin näistä neljästä vaihtoehdosta selkeimmät tuloskuvat, sillä ne sisältävät vain 1–4 tunnistuskehystä per yksi tuloskuva. Testissä 3 tuloskuva sisältää tunnistuskehyksiä 1–14 kappaletta, testissä 6 peräti 8–40 kappaletta ja testissä 11 tunnistuskehyksiä on 1–14 kappaletta.

Kuva 54. Testin 14 hyväksytyt tunnistustulokset, kriteerinä hyväksymiseen oli se, että yksi tunnistuskehys peittää kohteesta silmämääräisesti enemmän kuin 70%.

Testin 14 hyväksytyt tunnistustulokset esitellään kuvassa 54, ja vertailukohteena muutamia testin 16 tuloskuvia esitellään kuvassa 55.



Kuva 55. Testin 16 tunnistustuloksia vertailukohteena kuvassa 53 oleviin tuloksiin. Erona näissä kuvan koko, käytetty komponenttimalli ja kynnysarvo.

Tehdyistä tunnistustesteistä pääteltiin, että lähes aina suurin tunnistuskehys peittää kohteesta enemmän kuin 70 %. Siitä syystä päädyttiin tekemään funktio, joka etsii VOCin antamista tunnistustuloksista suurimman kehyksen.

1.3.2011 Tunnistustestejä: suurin tunnistuskehys

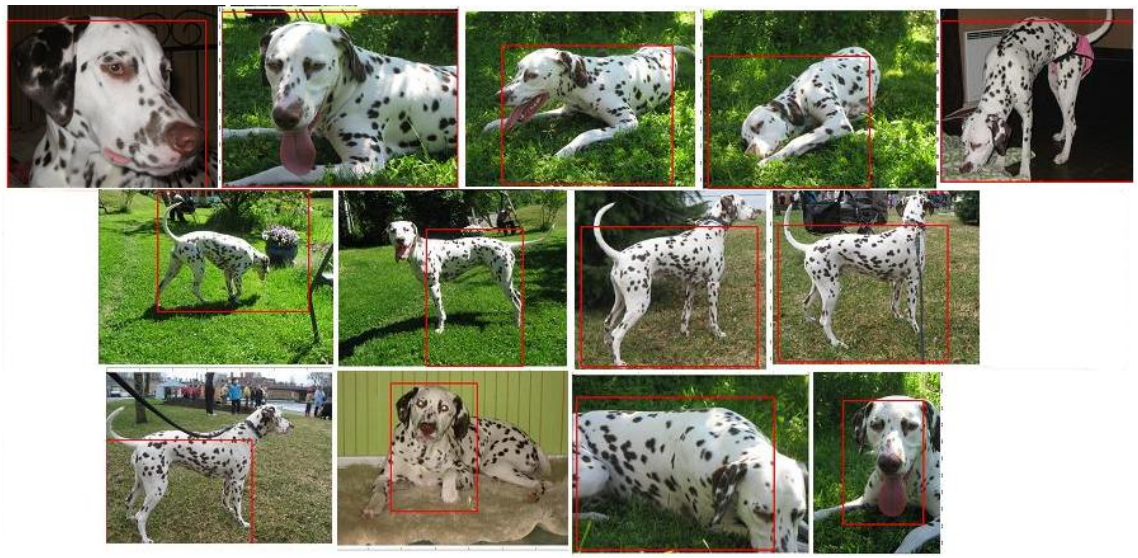
Viimeisessä vaiheessa VOC-sovellusta testattiin suuremmalla aineistolla hyödyntäen tunnistamisprosessia varten tehtyä *findMaxSizeBbo*-funktiota (kuva 56), jota kutsutaan VOCin tunnistusprosessin käynnistävän *process*-funktion lopusta. *findMaxSizeBbo* etsii parametrinaan saamasta matriisista suurimman tunnistuskehyksen.

```
1  function [ bbo ] = findMaxSizeBbo( bbox )
2
3  % Finds max size bounding box and return it.
4  % 1.3.2011 Marika Könönen
5
6  [row, column] = size(bbox);
7  sizesOfBbos   = zeros(1,row);
8  bbo           = zeros(1,column);
9  xLength       = zeros(1,1);
10 yLength       = zeros(1,1);
11
12 for i=1:row
13     xLength          = bbox(i,3) - bbox(i,1);
14     yLength          = bbox(i,4) - bbox(i,2);
15     sizesOfBbos(1,i) = xLength * yLength;
16 end;
17
18 [C,indexOfMaxBboRow] = max(sizesOfBbos);
19
20 for j=1:column
21     bbo(indexOfMaxBboRow,j) = bbox(indexOfMaxBboRow,j);
22 end;
```

```
1  function bbox = process(image, model, thresh)
2
3  % bbox = process(image, model, thresh)
4  % Detect objects that score above a threshold, return bonding boxes.
5  % If the threshold is not included we use the one in the model.
6  % This should lead to high-recall but low precision.
7
8  if nargin < 3
9    thresh = model.thresh
10 end
11
12 boxes = detect(image, model, thresh);
13 bbox = getboxes(model, boxes);
14 bbox = nms(bbox, 0.5);
15 bbox = clipboxes(image, bbox);
16
17 % M.Könönen lisännyt oman funktion
18 bbox = findMaxSizeBbo(bbox);
```

Kuva 56. Funktio findMaxSizeBbo (vasemmalla) etsii parametrina saadusta taulukosta suurimman tunnistuskehyksen. VOCin process-funktio kutsuu findMaxSizeBbo:ta (oikealla rivi 18).

Funktion *findMaxSizeBbo* vaikutusta esitellään kuvassa 57, jonka havainnollistamat testit toteutettiin samoilla asetuksilla, kuin kuvan 54 testit (testinumero 14). Tuloksien perusteella todettiin löydetyn ratkaisun olevan tarpeeksi hyvä tämän insinöörityön kokonaisuuteen ja tavoitteisiin nähden.



Kuva 57. Tunnistustulokset testin 14 mukaisilla asetuksilla käyttäen funktiota findMaxSizeBbo.

Viimeiseksi testattiin *findMaxSizeBbo*-funktiota uudella testiaineistolla.

1.3.2011 Tunnistustestejä: uusi testiaineisto

Funktion *findMaxSizeBbo* vaikutusta VOC-sovelluksella saatuihin tunnistamistuloksiin testattiin kaksi- ja neljäkomponenttisilla tunnistusmalleilla käyttäen mallien sisältämiä

kynnysarvoja (katso kuva 53). Testiaineisto koostui kolmesta eri kuvasarjasta [Fei-Fei ym. 2003b: dalmatian; Kainulainen; Kallio], joissa oli yhteensä 393 kuvaa. Haastavin ja kuviltaan monipuolisin oli Katja Kallion 303 kappaleen kuva-arkisto, jolla testattaessa hyväksytyt tunnistustulokset jäivät alle 50 %.

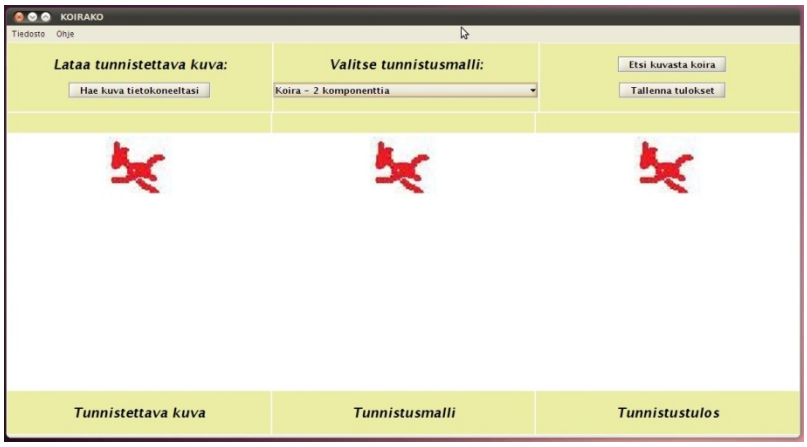
Nro	Pvm	Mallin komp. lkm	Kynnysarvo	Testikuvien lkm	Hyväksytty tunnistus lkm	Kuvasarja	Hyväksytyt tunnistukset %
1	1.3.2011	2	Malli: -1,0343	67	53	101 Object Categories: dalmatian	79 %
2	1.3.2011	4	Malli: -0,7669	67	45	101 Object Categories: dalmatian	67 %
3	1.3.2011	2	Malli: -1,0343	303	132	Katja Kallio: kuva-arkisto	44 %
4	1.3.2011	4	Malli: -0,7669	303	146	Katja Kallio: kuva-arkisto	48 %
5	1.3.2011	2	Malli: -1,0343	23	20	Heini Kainulainen: kuva-arkisto	87 %
6	1.3.2011	4	Malli: -0,7669	23	15	Heini Kainulainen: kuva-arkisto	65 %
	yht.	**2**		**393**	**205**		**52,16 %**
	yht.	**4**		**393**	**206**		**52,42 %**

Kuva 58. 1.3.2011 suoritettujen tunnistustestien yhteenveto. Testiaineisto sisälsi yhteensä 393 kuvaa, jotka koostuivat kolmesta eri kuvasarjasta [Fei-Fei ym. 2003b: dalmatian; Kainulainen; Kallio].

Tehtyjen tunnistustestien yhteenveto (kuva 58) paljastaa, että näissä testeissä yhteistulokset eivät eronneet kaksi- ja neljäkomponenttisien tunnistusmallien välillä: kaksikomponenttisen mallin kokonaistunnistusprosentti on 52,16 ja neljäkomponenttisen 52,42 %, eroa on siis vain 0,26 prosenttiyksikköä.

7.3 Käyttöohje

KOIRAKO vaatii toimiakseen Linux-käyttöjärjestelmän, MATLAB-ohjelmistosta vähintään version 7.6 t ja VOC-sovelluksen version 3.1 [Felzenszwalb ym. 2009]. Ohjeet KOIRAKOn asentamiseen löytyvät liitteestä 2, joka sisältää muun muassa muutoksia VOCin funktioihin ja KOIRAKOn hakemistopolkuihin.



Kuva 59. KOIRAKO-sovelluksen pääikkuna sovelluksen käynnistyttyä.

KOIRAKOn käyttöliittymä sisältää yhden näkymän (kuva 59), josta sen kaikki toiminnot tapahtuvat. Hae kuva tietokoneeltasi -painike avaa tiedoston valintaikkunan, josta valitaan tunnistettava kuva. Tunnistamisessa käytettävä malli valitaan pudotusvalikosta, tunnistusprosessi käynnistetään Etsi kuvasta koira -painikkeella ja tuloskuvan tallennetaan Tallenna tulokset -painikkeella. KOIRAKOn käyttöä käyttötapauksittain (K1–K4) on simuloitu liitteessä 3.

7.4 Jatkokehityksestä

Tässä luvussa luetellaan tunnettuja KOIRAKO-sovelluksen käyttöliittymään ja ohjelmakoodiin liittyviä puutteita sekä esitellään ehdotuksia sovelluksen jatkokehittämiseksi.

Kovakoodatut hakemistopolut

KOIRAKO-sovelluksen ohjelmakoodissa on hakemistopolkuja sisältäviä muuttujia, joiden arvot on upotettu ohjelmakoodiin. Näiden muuttujien arvot tulisi lukea asetustiedostosta, ja hakemistopolkujen vaihtamisen tulisi olla mahdollista sovelluksen käyttöliittymästä käsin.

Käyttöliittymäkomponentit tuhoavien desktruktorien puuttuminen

Mikäli KOIRAKO suljetaan ja käynnistetään uudestaan MATLABia sulkematta, luodaan sovelluksen käyttöliittymän komponenteista uudet ilmentymät jo olemassa olevien lisäksi. Tämä johtuu luultavasti siitä, ettei handle-luokan periviin alaluokkiin kuuluvia destruktoreita ole toteutettu.

Tuntemattomien tiedostotyyppien puutteellinen käsittely

Sovelluksen ohjelmakoodi ei osaa käsitellä virhetilannetta, joka syntyy käyttäjän valitessa tiedostonvalintaikkunasta kohteen, joka ei ole tunnetun muotoinen kuva. Tätä kirjoittaessa käyttäjä voi valita tiedostonvalintaikkunasta minkä tahansa tiedoston tai hakemiston ilman, että sovellus reagoi siihen mitenkään.

MATLABin abstraktin hgsetget-luokan periytyminen

hgsetget-luokka sisältää erityiset getter- ja settermetodit, joita käyttämällä saadaan kutsuttua toisen luokan muuttujia. Ongelmana on kuitenkin se, että niiden käyttäminen vaatii viittauksen kohteena olevan luokan muuttujien GetAccess-parametrin arvoksi public. KOIRAKO-sovelluksen ohjelmoinnissa hgsetget-luokkaa on käytetty siksi, että MATLABin oliopohjaiseen ohjelmointiin tutustuessa luultiin sen käytön olevan pakollista, mikä myöhemmin osoittautui vääräksi tiedoksi. Vaihtoehtona hgsetget-luokan metodien käyttämiselle voidaan tehdä omat getter- ja setter-metodit, jolloin GetAccess-parametrin arvo voisi olla private, joka sen kuuluukin olla. Ohjelmakoodia tulisi muuttaa siten, ettei mikään luokka perisi MATLABin abstraktia hgsetget-luokkaa, koska täten luokkien muuttujien GetAccess-parametrin arvo voisi olla private.

MATLABin pakettimäärittelyt

KOIRAKO-sovelluksen paketti- ja moduulikaaviosta on nähtävissä, että sovellus koostuu paketeista, jotka on jaettu moduuleihin. Käytännössä tämä tapahtuu MATLABissa siten, että luodaan hakemisto, jonka nimi alkaa plusmerkillä eli käyttöliittymäpaketin hakemiston nimi voisi olla +käyttöliittymä [Scoping Classes with Packages], ja kyseiseen hakemistoon sijoitetaan kaikki käyttöliittymäpakettiin kuuluvat luokat. Pakettimäärittelyt kuitenkin puuttuvat KOIRAKOn toteutuksesta.

Tiedostonvalinta- ja -tallennusikkunoiden Cancel-painikkeet

Tiedostonvalinta- ja -tallennusikkunoiden Cancel-painikkeiden toimintoja ei ole ohjelmoitu, siksi niiden napauttaminen aiheuttaa virhetilanteen MATLABissa.

Pudotusvalintalista

KOIRAKOn käynnistyessä pääikkunan Valitse tunnistusmalli-pudotusvalintalistan oletusvalintana on *Koira - 2 komponenttia*, mutta tunnistusnäkymässä ei ole sen kuvaa vaan

KOIRAKOn logo. Pudotusvalintalistan oletusvalinnan tulisi olla joko tyhjä tai esimerkiksi teksti *<valitse malli>.*

Tiedostonvalinta- ja -tallennusikkunoiden tekstit

Käyttöliittymän JFileChooser-komponentin tekstit ovat englanninkieliset. Ne pitäisi vaihtaa suomenkielisiksi, jotta sovelluksen ulkoasu säilyisi yhdenmukaisena ja käytettävyys paranisi.

Valikkopalkin toiminnot

Sovelluksen valikkopalkin toiminnot Tiedosto-valikon Hae kuva, Tallenna ja Sulje sovellus eivät ole käytettävissä, koska ohjelmakoodi on jäänyt keskeneräiseksi.

Hahmontunnistuksen rajoittuminen yhteen koiraan kerralla

VOC-sovellukseen tehty funktio *findMaxSizeBbo* etsii tunnistusprosessissa suurimman tunnistuskehyksen ja palauttaa kehyksen. Funktio rajoittaa tunnistusprosessin tuottamaa tulosta siten, että tuloksena näytetään vain yksi tunnistuskehys, vaikka VOC-sovellus paikantaisi useamman koiran kuvasta.

Muiden kohdeluokkien kohteiden tunnistaminen

KOIRAKOa käytettäessä tunnistusprosessissa käytettävä tunnistusmalli valitaan pudotusvalintalistasta. Muuttamalla käyttöliittymää siten, että tunnistusmallin sisältävän tiedoston voi valita tietokoneelta, olisi myös muiden kohteiden kuin koirien tunnistaminen ja paikantaminen mahdollista. Sen toteuttamiseen tarvitaan vain tunnistusmalli ja muutama muutos KOIRAKOn käyttöliittymän ohjelmakoodissa.

8 Olio-ohjelmointi MATLABilla

MATLAB *(Matrix Laboratory)* on vektori- ja matriisilaskennan kehitysympäristö sekä siinä käytettävä ohjelmointikieli, joka mahdollistaa nopean numeerisen laskennan esimerkiksi tieteellisiin ja teknisiin tehtäviin [MATLAB – The Language Of Technical Computing]. MATLABilla ohjelmoimiseen liittyy useita ratkaisuja, jotka saattavat vaikuttavat kummallisilta, jos on aloittanut ohjelmoinnin Java-kielellä.

MATLAB-ohjelmointikielessä olio-ohjelmointi ominaisuuksia parannettiin huomattavasti MATLABin versiossa 7.6 vuonna 2008, kun siihen oli mahdollista määritellä luokka, metodit ja luokkamuuttujat [Programming, MATLAB Version 7.6 (R2008a)]. Sitä ennen versiosta 6 alkaen luokkaoliot olivat teknisesti struktuureita ja ne määriteltiin alihakemistona, joka sisälsi luokan luonti- ja metodifunktiot [Apiola & Laine]. Esimerkiksi luokka nimeltä *pallo* toteutettiin luomalla alihakemisto *@pallo*.

Tässä luvussa esitellään olio-ohjelmointia, kuten esimerkiksi luokkien, muuttujien ja metodien määrittely MATLAB-ympäristössä.

8.1 Luokka-, muuttuja- ja metodilohkot

Seuraavaksi esitellään lyhyesti luokka-, muuttuja ja metodilohkojen määrittelyt MATLABissa.

Luokkalohko (classdef)

Luokka luodaan MATLABissa m-tiedostoon, joka on MATLABin oma tiedostopääte. Tiedosto katsotaan luokaksi, kun sinne on määritelty classdef-niminen lohko. M-tiedoston ensimmäiselle riville kirjoitetaan *classdef UudenLuokanNimi* ja lohko suljetaan sanalla *end*. Classdef-lohko koostuu properties-, methods- ja events-alilohkoista. Kuvassa 60 on esimerkki luokasta nimeltä KOIRAKOSovelluslogiikka, ja se sisältää kahdenlaisia properties-alilohkoja sekä methods-alilohkon. [Classdef Block.]

```
1   classdef KOIRAKOSovelluslogiikka
2       properties (Constant)
4       end
5       properties (SetAccess = private, GetAccess = public)
7       end
8       methods
34      end
35  end
```

Kuva 60. MATLABin luokkamäärittelyt m-tiedostossa. Vihreällä on merkattu classdef-lohkon alku ja loppu, keltaisella properties-alilohko ja punaisella methods-alilohko.

Luokkaa voidaan kutsua MATLAB-ohjelmiston komentoikkunasta suoraan luokan nimellä, jolloin luokasta luodaan olio, jota pystyy tarkastelemaan MATLABin työtilassa *(workspace)*. MATLAB tallentaa olion muuttujaan nimeltä ans.

Muuttujalohkot (properties)

MATLABin olio-ohjelmoinnin syntaksissa muuttujat määritellään properties-lohkoissa [Property Attributes]. Esimerkiksi, jos luokassa on näkyvyysalueeltaan private- ja public-määritteisiä muuttujia, ne pitää jakaa omiin properties-lohkoihin. Toiselle lohkolle tulee attribuutiksi *Access=private* ja toiselle *Access=public* (kuva 60). MATLABin properties-määrittelyn erikoisuus on se, että Access-attribuutin sijaan voidaan määritellä erikseen SetAccess- ja GetAccess-ominaisuudet.

Metodilohkot (methods)

Myös metodilohkoille määritellään attribuutit aivan kuten properties-lohkoillekin. MATLABin metodien oletusnäkyvyys on public eli julkinen [Class Methods]. Siinä tapauksessa lohkolle ei anneta parametreja. AbstraktiNakyma-luokka on abstrakti luokka, jonka kuvissa esiintyvät TunnistusNakyma- sekä AsetusNakyma-luokat perivät. Se sisältää attribuuteiltaan kahden tyyppisiä metodeja: julkisia sekä staattisia abstrakteja. Luokan metodit muokkaavat käyttöliittymän ulkoasua, kuten esimerkiksi paneelien värejä ja otsikoiden fontteja.

Jokaisen metodin ensimmäisenä parametrina on viittaus, joka viittaa kyseisen metodin omaan luokkaan. Parametrin voi nimetä miten vain ja tämän työn ohjelmakoodissa sitä kutsutaan nimellä *self*. MATLABIssa ei periaatteessa ole muita parametrittomia metodeja kuin konstruktori ja staattiset metodit, joita voidaan kutsua ilman luokan ilmentymää. Ei-staattiset metodit sisältävät aina parametrin self. Sen avulla päästään käsiksi luokan muuttujiin (kuten esimerkiksi *self.tulosLbl)* ja voidaan kutsua luokan ei-staattista metodia *self.setUlkoasuPaneeli().* Huomattavaa on, että metodin määrittelyssä tulee mainita *self,* mutta metodia kutsuttaessa se jätetään pois.

8.2 MATLABin hgsetget-luokka

AbstraktiNakyma-luokka perii MATLABin abstraktin *hgsetget*-luokan ja näin sen metodit on käytettävissä myös TunnistusNakyma- ja AsetusNakyma-luokissa. *hgsetget*-luokalla on neljä metodia: *set*, *get*, *setdisp* sekä *getdisp* ja se on edellä mainitun *handle*-luokan aliluokka [hgsetget]. Metodilla *set* asetetaan muuttujan arvon ja *get*-metodi palauttaa sen. *hgsetget*-luokan *get*-metodin käyttäminen vaatii toimiakseen muuttujan GetAc-

cess-näkyvyysalueen julkiseksi. KOIRAKO-sovelluksen suunnitteluvaiheessa oletettiin, että getter pitää tehdä *hgsetget*-luokan *get*-metodilla. Myöhemmin huomattiin, että getterin toteutuksen voi tehdä itse, jolloin GetAccess attribuutin arvo saa olla *private*. Sovelluksen toteutus sisältää kuitenkin *hgsetget*-luokan get-metodeita, jotka pakottavat GetAccess-attribuutin arvoksi public.

8.3 Abstraktit luokat

Kuvassa 61 määritellään abstrakti luokka ja sen perivä aliluokka. Yliluokka peritään käyttämällä < merkkiä. Aliluokan metodeissa yliluokan metodeja kutsuttaessa kutsujono on pitkä, koska MATLABin syntaksi vaatii aliluokan metodin nimeksi saman, kuin mitä kutsuttava yliluokan metodi on. Se estää laittamasta kaikkien yliluokan metodikutsuja yhteen metodiin.

```
1   classdef AbstraktiNakyma
2       methods
16      end
17      methods(Abstract, Static)
20      end
21
22  end
```

```
1   classdef AsetusNakyma < AbstraktiNakyma
2       properties (Constant)
17      end
18      properties (SetAccess = private, GetAccess = public)
35      end
66  end
```

Kuva 61. Kuvassa vasemmalla on abstrakti luokka ja oikealla sen aliluokka. Periytyminen toteutetaan classdef-rivillä pienempi kuin -merkillä.

Esimerkkikoodi 1 havainnollistaa yliluokan metodin kutsumista TunnistusNakyma-luokasta.

```
classdef TunnistusNakyma < AbstraktiNakyma
  :
  methods
    function setUlkoasu(self)
        self.setUlkoasuPaneeli();
        self.setUlkoasuKuvaPaneeli();
        self.setUlkoasuOtsikkoFontti();
    end
    function setUlkoasuPaneeli(self)
        setUlkoasuPaneeli@AbstraktiNakyma(self.vasenpaneeli);
        setUlkoasuPaneeli@AbstraktiNakyma(self.keskipaneeli);
        setUlkoasuPaneeli@AbstraktiNakyma(self.oikeapaneeli);
    end
    function setUlkoasuKuvaPaneeli(self)
        setUlkoasuKuvaPaneeli@AbstraktiNakyma(self.vasenKuvapaneeli);
        setUlkoasuKuvaPaneeli@AbstraktiNakyma(self.keskiKuvapaneeli);
        setUlkoasuKuvaPaneeli@AbstraktiNakyma(self.oikeaKuvapaneeli);
    end
    function setUlkoasuOtsikkoFontti(self)
        setUlkoasuOtsikkoFontti@AbstraktiNakyma(self.ladattuKuvaLbl);
        setUlkoasuOtsikkoFontti@AbstraktiNakyma(self.malliLbl);
        setUlkoasuOtsikkoFontti@AbstraktiNakyma(self.tulosLbl);
    end
  end
end
```

Esimerkkikoodi 1. Yliluokan metodien kutsuminen aliluokasta.

MATLABin oliopohjaisen ohjelmoinnin käytettävyyttä parantaisi se, ettei yliluokan metodien käyttäminen vaatisi kutsuvalta metodilta samaa nimeä. Siinä tapauksessa kaikki yliluokan metodikutsut voisi sisältyä yhteen metodiin, kuten esimerkkikoodi 2 esittää. Huomattavaa on, että esimerkkikoodi 2 on vain teoreettinen parannusehdotus ja se tuottaa virhetilanteen MATLABissa.

```
function setUlkoasu(self)
    setUlkoasuPaneeli@AbstraktiNakyma(self.vasenpaneeli);
    setUlkoasuPaneeli@AbstraktiNakyma(self.keskipaneeli);
    setUlkoasuPaneeli@AbstraktiNakyma(self.oikeapaneeli);
    setUlkoasuKuvaPaneeli@AbstraktiNakyma(self.vasenKuvapaneeli);
    setUlkoasuKuvaPaneeli@AbstraktiNakyma(self.keskiKuvapaneeli);
    setUlkoasuKuvaPaneeli@AbstraktiNakyma(self.oikeaKuvapaneeli);
    setUlkoasuOtsikkoFontti@AbstraktiNakyma(self.ladattuKuvaLbl);
    setUlkoasuOtsikkoFontti@AbstraktiNakyma(self.malliLbl);
    setUlkoasuOtsikkoFontti@AbstraktiNakyma(self.tulosLbl);
  end
```

Esimerkkikoodi 2. Parannusehdotus MATLABin tapaan sitoa yliluokan metodikutsu aliluokan metodin nimeen, vertaa esimerkkikoodi 1.

8.4 Callback-toiminnot

KOIRAKOKayttoliittyma-luokka perii MATLABin abstraktin *handle*-luokan, jota tarvitaan sovelluksen toimintojen kannalta tärkeässä asetaKontrolleri-metodissa. Tämän metodin toteutukseen löydettiin apua Undocumented Matlab -sivustolta [Uicontrol callbacks], jossa neuvotaan Javan käyttöliittymäkomponenttien ja MATLABin käyttämistä yhdessä. MATLABin kehittäjän MathWorksin käyttöohjeista ei löydy vastaavia tietoja. Esimerkkikoodi 3 havainnollistaa handle-objektin käyttöä ActionPerformedCallback-toiminnon asettamisessa.

```
haeKuvaBtn = javax.swing.JButton('Hae kuva');
haeKuvaBtnHndl  = handle (haeKuvaBtn, 'CallbackProperties');
set(haeKuvaBtnHndl, 'ActionPerformedCallback', @kontrolleri.haeKuvaBtnClick);
```

Esimerkkikoodi 3. Callback-toimintojen asettaminen.

set-funktion käyttö ilman *handle*-objektia (esimerkkikoodi 3) saattaa aiheuttaa pahoja muistivuotoja, mikä vuoksi haeKuvaBtn-muuttujaa ei kannata laittaa suoraan funktion *set* parametriksi [Uicontrol callbacks]. Se määrittelee KOIRAKOKontrolleri-luokan metodin nimen, jota kutsutaan, kun esimerkiksi haeKuvaBtn-muuttujan viittamaa kompo-

nenttia eli painiketta napautetaan. Edellä mainittujen metodien lisäksi KOIRAKOKayttoliittyma-luokka sisältää gettereitä ja settereitä sekä muita metodeita, joilla päivitetään käyttöliittymää KOIRAKOKontrolleri-luokan callback-metodien toimesta.

9 Yhteenveto

Tämä insinöörityö oli lähtöisin kiinnostuksesta hahmontunnistukseen ja käyttöliittymäohjelmointiin sekä halusta oppia ymmärtämään molempia osa-alueita. Insinöörityön tavoitteena oli perehtyä hahmontunnistukseen ja löytää menetelmä, jota voitaisiin käyttää toteutettavassa hahmontunnistussovelluksessa. Insinöörityö koostui hahmontunnistuksen ja koneoppimisen teorian opiskelusta sekä ohjelmoinnista MATLAB-ympäristössä. Lisäksi tutkittiin ja testattiin VOCia, mikä auttoi soveltamaan opittua teoriaa käytännössä.

Insinöörityön tuloksena määriteltiin, suunniteltiin ja toteutettiin graafisen käyttöliittymän sisältämä hahmontunnistussovellus KOIRAKO, joka tunnistaa ja paikantaa digitaalisesta kuvasta koiran. Se saa syötteenä analysoitavan kuvan ja palauttaa tunnistustuloksen kuvana, jossa onnistuneesti tunnistettu kohde on rajattu tunnistuskehyksellä. KOIRAKO toteutettiin olioperusteisesti MATLAB-kielellä, sen käyttöliittymä koottiin Swing-kirjaston komponenteista ja sen sovelluslogiikasta luotiin yhteys VOC-sovellukseen kuvan analysointiin tarvittavien funktioiden osalta. Lisäksi VOCin ohjelmakoodia muokattiin KOIRAKOn käyttötarpeiden mukaisiksi.

Ensimmäisessä vaiheessa tutkittiin hahmontunnistuksen käsitteitä ja menetelmiä, joiden myötä esiin tulleisiin hahmontunnistusalgoritmeihin syvennyttiin tarkemmin tavoitteena ymmärtää koneoppimisen oppimis- ja tunnistusvaiheita alusta loppuun asti. Siinä samalla löydettiin VOC-sovellus, joka osoittautui erinomaiseksi valinnaksi monipuolisen ja haasteellisen teorian johdosta.

Ongelmaksi muodostui työn laajuus, sillä jo perusalgoritmeja tutkittaessa, siinä ohessa täytyi perehtyä lisäksi muun muassa matematiikkaan (lineaarialgebraa, todennäköisyysteoriaa ja päätösteoriaa), digitaaliseen kuvankäsittelyyn ja digitaaliseen signaalinkäsittelyyn. Lisäksi MATLAB vektori- ja matriisilaskentoineen oli täysin tuntematon

ympäristö, johon syventyminen vei aikaa, unohtamatta MATLABin olio-ohjelmointiin perehtymistä. Haasteena olivat myös alan artikkelit, joita työn alkuvaiheessa oli lähes toivotonta ymmärtää peruskäsitteiden ollessa tuntemattomia. Hahmontunnistukseen ja koneoppimiseen tutustumisen myötä työskentely helpottui ja tutkimustyön edetessä artikkeleista löytyi yhä enemmän insinöörityössä hyödynnettäväksi soveltuvaa tietoa.

KOIRAKO-toteutuksessa hyödynnetyn VOC-sovelluksen testaaminen tapahtui kokemusperäisesti, koska hahmontunnistuksen teoriaan perehtyminen ei ollut KOIRAKOn suunnittelu- ja toteutusaikana vielä siinä vaiheessa, että olisi osattu tehdä kehityspäätöksiä VOCin suhteen. VOCia testattiin MATLABissa ja saavutettiin tämän insinöörityön laajuuteen nähden kelvolliset tulokset, joiden perusteella VOC valittiin osaksi KOIRAKOn sovelluslogiikkaa. Vasta KOIRAKO-sovelluksen ohjelmoimisen jälkeen päästiin jatkamaan VOCin teorian sisältävien algoritmien käsittelyä pääsemättä kuitenkaan tutkimaan VOCin teoriaa täysin loppuun asti tämän insinöörityöraportin valmistumisajankohdan puitteissa.

Niin KOIRAKOssa kuin VOC-sovelluksessa riittää paranneltavaa tämän insinöörityön aiheen osalta. KOIRAKOn ohjelmakoodia tulisi järkeistää esimerkiksi poistamalla kovakoodatut ohjelmistopolut ja VOC-sovelluksen käyttöä KOIRAKOssa tulisi tutkia opitun teorian pohjalta.

Lähteet

Abstract Window Toolkit (AWT). Verkkosivu. Oracle.
<http://docs.oracle.com/javase/1.4.2/docs/guide/awt/>. Luettu 31.3.2011.

Adelson, E.H. – Anderson, C.H. – Bergen, J.R. – Burt, P.J. – Ogden, J.M. 1984. Pyramid methods in image processing. RCA Engineer, 29-6, Nov/Dec. Verkkodokumentti. <https://alliance.seas.upenn.edu/~cis581/wiki/Lectures/Pyramid.pdf>.

Agarwal, Shivani – Awan, Aatif – Roth, Dan. 2004. Learning to Detect Objects in Images via a Sparse, Part-Based Representation. IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 26, no. 11, s. 1475–1490.

Apiola, Heikki – Laine, Marko. MATLAB-opas, MATLABin uudet piirteet. Verkkosivu. Helsingin yliopisto. <http://www.helsinki.fi/~mjlaine/matlab/uutta.html>. Päivitetty 14.6.2008. Luettu 24.5.2012.

Bertsekas, Dimitri P. 2005. Dynamic Programming and Optima Control. Third edition. USA: Athena Schientific.

Bishop, Christopher M. 2006. Pattern Recognition and Machine Learning. USA: Springer.

Chen, Pai-Hsuen – Lin, Chih-Jen – Schölkopf Bernhard. 2005. A Tutorial on v-Support Vector Machines. Applied Stochastic Models in Business and Industry, vol. 21, no. 2, s. 111–136. Verkkodokumentti.
<http://www.csie.ntu.edu.tw/~cjlin/papers/nusvmtutorial.pdf>.

Classdef Block. Verkkosivu. The MathWorks.
<http://www.mathworks.se/help/techdoc/matlab_oop/brqy3km-6.html>.
Luettu 11.5.2012.

Class Methods. Verkkosivu. The MathWorks.
<http://www.mathworks.com/help/techdoc/matlab_oop/f1-13781.html>.
Luettu 16.3.2011.

Dalal, Naveet. 2006. Finding People in Images and Videos. PhD thesis. France: Institut National Polytechique de Grenoble. Verkkodokumentti.
<http://lear.inrialpes.fr/pubs/2006/Dal06/Dalal-phd06.pdf>.

El-Maraghi, Thomas F. 2004. SIFT algorithm in MATLAB.

Extensible Markup Language (XML) 1.0 (Fift Edition). Verkkodokumentti. W3C.
<http://www.w3.org/TR/REC-xml/>. Päivitetty 21.3.2008. Luettu 7.6.2011.

Estimaatti. Tilastokeskus. Verkkodokumentti.
<http://www.stat.fi/virsta/tkeruu/kasitteet/estimaatti1077/index.html>.
Päivitetty 27.1.2006. Luettu 20.5.2011.

Everingham, M. – Van Gool. L. – Williams, C. K. – Winn, J. – Zisserman, A. 2007. "The PASCAL Visual Object Classes Challenge 2007 (VOC2007)". Verkkodokumentti. <http://pascallin.ecs.soton.ac.uk/challenges/VOC/voc2007/>.

Fei-Fei, L. – Fergus, R. – Perona, P. 2003a. A Bayesian approach to unsupervised one-shot learning of object categories. Proceedings of the 9th International Conference on Computer Vision, Nice, France, s. 1134–1141.

Fei-Fei, Li – Andreetto, Marco – Ranzato, Mark'Aurelio. 2003b. The Caltech-101 Object Categories. Verkkosivu. <http://www.vision.caltech.edu/feifeili/Datasets.htm>.

Felzenszwalb, P. – Girshick, R. – McAllester D. – Ramanan, D. 2010. Object Detection with Discriminatively Trained Part Based Models. IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 32, no. 9, s. 1627–1645. Verkkodokumentti. <http://people.cs.uchicago.edu/~pff/papers/lsvm-pami.pdf>.

Felzenszwalb, P. – Girshick, R. B. – McAllester, D. 2009. Discriminatively Trained Deformable Part Models, Release 3, MATLAB-toteutus. Verkkodokumentti. <http://people.cs.uchicago.edu/~pff/latent-release3/>.

Felzenszwalb, P. – Huttenlocher, D. 2004. Distance Transforms of Sampled Functions. Cornell University CIS. technical report. Verkkodokumentti. <http://www.cs.cornell.edu/~dph/papers%5Cdt.pdf>.

Felzenszwalb, P. – Huttenlocher, D. 2005. Pictorial Structures for Object Recognition. International Journal of Computer Vision, vol. 61. no. 1.

Fergus, Robert. 2003. Constellation model code, MATLAB-toteutus. Verkkodokumentti. <http://cs.nyu.edu/~fergus/constellation.html>. Luettu 10.11.2010.

Fergus, Robert. 2005a. Star model code, MATLAB-toteutus. Verkkodokumentti. <http://cs.nyu.edu/~fergus/cvpr05_code.html>. Luettu 10.11.2010.

Fergus, Robert. 2005b. Visual Object Category Recognition. PhD thesis. UK: University of Oxford.

Fergus, R. – Perona, P. – Zisserman, A. 2005. A Sparse Object Category Model for Efficient Learning and Exhaustive Recognition. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, San Diego, vol. 1, s. 380–387.

Garcia, Vincent. 2007. Keypoint Extraction, MATLAB-toteutus. Verkkosivu. <http://www.mathworks.com/matlabcentral/fileexchange/authors/2892>.

Gonzalez, Rafael C. – Woods, Richard E. 2002. Digital Image Processing. Second edition. US, New Jersey: Pearson Education.

Gupta, Maya R. – Chen, Yi hua. 2010. Theory and Use of the EM Algorithm. Foundations and Trends in Signal Processing, vol. 4, no. 3, s. 223–296. Verkkodokumentti. <http://www.ee.washington.edu/research/guptalab/publications/EMbookChenGupta2010.pdf>.

hgsetget. Verkkosivu. The MathWorks. <http://www.mathworks.com/help/techdoc/ref/hgsetget.html>. Luettu 30.4.2011.

Holmström, Lasse. 2010. Tilastollinen hahmontunnistus. Verkkodokumentti. Oulun yliopisto: Matemaattisten tieteiden laitos. <http://cc.oulu.fi/~llh/HT2010/Luentomoniste.pdf>.

Jolliffe, I.T. 2002. Principal Component Analysis. Second edition. US, New York: Springer.

Jäntti, Timo-Pekka. 2010–2012. Luennot, keskustelut ja sähköpostit.

Kainulainen, Heini. Valokuva-arkistot: Lenni & Pablo.

Kallio, Katja. Valokuva-arkistot: Jade, Pastori & muut.

Koistinen, Ari. 2005. Joukkojen mahtavuudesta. Solmu 3/2005, s. 15–17. Verkkodokumentti. <http://solmu.math.helsinki.fi/2005/3/solmu31.pdf>.

Kämäräinen, Joni. 2003. Teräksen vakuumikäsittelyn visuaalinen laadunarviointi. Verkkodokumentti. <http://personal.lut.fi/users/joni.kamarainen/downloads/publications/dithesis.pdf>.

Kääriäinen, Matti. 2008a. Johdatus koneoppimiseen, viikko 1. Verkkodokumentti. Helsingin yliopisto: Tietojenkäsittelytieteen laitos. <http://www.cs.helsinki.fi/group/joko/viikko1.pdf>. Luettu 3.4.2012.

Kääriäinen, Matti. 2008b. Johdatus koneoppimiseen, viikko 2. Verkkodokumentti. Helsingin yliopisto: Tietojenkäsittelytieteen laitos. <http://www.cs.helsinki.fi/group/joko/viikko3.pdf>. Luettu 24.4.2012.

Könönen, Marika. 2009–2011. Valokuva-arkistot:Pala.

Leibe, Bastian. 2008. Multi-Cue ISM 1.4.1, toteutus. Verkkosivu. <http://www.vision.ee.ethz.ch/~bleibe/code/ism.html>.

Leibe, Bastian – Leonardis, Aleš – Schiele, Bernt. 2008. Robust Object Detection with Interleaved Categorization and Segmentation, International Journal of Computer Vision, vol. 77, no. 1/3, s. 259–289. Verkkodokumentti. <http://www.vision.ee.ethz.ch/~bleibe/papers/leibe-interleaved-ijcv07final.pdf>.

LeCun, Y. – Chopra, R. – Hadsell, R. – Ranzato, M. – Huang, F. 2007. Energy-Based Models. In Predicting Structured Data, G. Bakir, T. Hofman, B. Schölkopf, A. Smola, B. Taskar, S. V. N. Vishwanathan (eds), p.191–216. London, England: The MIT Press.

Leino, Yrjö. 2004. Matlab-paketti huonosti asetettujen tehtävien regularisointiin. @CSC, 1/2004, s. 3–5. < http://www.csc.fi/blogit/atcsc-arkisto/2004/1>.

Lin, Chengzhu – Li, Shaozi – Su, Songzhi. 2009. Image Classification Using Adapted Codebook. IEEE International Symposium on IT in Medicine & Education, vol.1, s. 1307–1312.

Lindeberg, Tony. 1994. Scale-space theory: A basic tool for analysing structures at different scales. Journal of Applied Statistics, 21(2), 1994, pp. 225–270. Verkkodokumentti. < ftp://ftp.nada.kth.se/CVAP/scsp/papers/scsptheory-review.jas94.pdf>.

Liukkonen, Jukka. 2007. Yleistetyt funktiot eli distribuutiot. Solmu 1/2007, s. 25–29. Verkkodokumentti. <http://solmu.math.helsinki.fi/2007/1/solmu37.pdf>

Lowe, David G. 1999. Object Recognition from Local Scale-Invariant Features. In International Conference on Computer Vision, Corfu, Greece, pp.1150–1157. Verkkodokumentti. <http://people.csail.mit.edu/torralba/courses/6.870/papers/sift_iccv99.pdf.>

Lowe, David G. 2004. Distinctive Image Features from Scale-Invariant Keypoints. International Journal of Computer Vision, 60(2), s. 91–110. Verkkodokumentti. <http://www.cs.ubc.ca/~lowe/papers/ijcv04.pdf>.

MATLAB Builder JA for Java language–Deploy MATLAB code as Java Classes. Verkkosivu. The MathWorks. <http://www.mathworks.com/products/javabuilder/>. Luettu 31.1.2011.

MATLAB – The Language Of Technical Computing. Verkkosivu. The MathWorks. <http://www.mathworks.com/products/matlab/>. Luettu 1.4.2011.

Mellin, Ilkka. 2010a. Todennäköisyyslaskenta, osa 2: Satunnaismuuttujat ja todennäköisyysjakaumat: Peruskäsitteet. Verkkodokumentti. < https://noppa.aalto.fi/noppa/kurssi/mat-1.2600/luennot/Mat-1_2600_satunnaismuutujat_ja_todennakoisyysjakaumat.pdf >.

Mellin, Ilkka. 2010b. Todennäköisyyslaskenta, osa 2: Satunnaismuuttujat ja todennäköisyysjakaumat: Kertymäfunktio. Verkkodokumentti. < https://noppa.aalto.fi/noppa/kurssi/mat-1.2600/luennot/Mat-1_2600_kertymafunktio.pdf>.

Mellin, Ilkka. 2010c. Todennäköisyyslaskenta, osa 2: Satunnaismuuttujat ja todennäköisyysjakaumat: Moniulotteiset satunnaismuuttujat ja jakaumat. Verkkodokumentti. <https://noppa.aalto.fi/noppa/kurssi/mat-1.2600/luennot/Mat-1_2600_moniulotteiset_satunnaismuuttujat_ja_jakaumat.pdf>.

Mellin, Ilkka. 2010d. Todennäköisyyslaskenta, osa 2: Satunnaismuuttujat ja todennäköisyysjakaumat: Jakaumien tunnusluvut. Verkkodokumentti. <https://noppa.aalto.fi/noppa/kurssi/mat-1.2600/luennot/Mat-1_2600_tunnusluvut.pdf>.

Mellin, Ilkka. 2010e. Todennäköisyyslaskenta, osa 1: Todennäköisyys ja sen laskusäännöt: Todennäköisyyden peruslaskusäännöt. Verkkodokumentti. <https://noppa.aalto.fi/noppa/kurssi/mat-1.2600/luennot/Mat-1_2600_peruslaskusaannot.pdf>.

Mikolajczyk, Krystian – Schmid, Cordelia. 2003. A performance evaluation of local descriptors, in Proceedings of the Conference of Computer Vision and Pattern Recognition, USA, s. 257–264.

Mikolajczyk, Krystian – Schmid, Cordelia. 2005. A performance evaluation of local descriptors, IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 27, no. 10, s. 1615–1630.

Mitchell, Tom M. 1997. Machine Learning. USA:McGraw Hill.

Mitä on hahmontunnistus? Verkkosivu. Suomen Hahmontunnistustutkimuksen Seura ry. < http://www.hatutus.org/files/hatutus_esite.pdf>. Luettu 5.8.2010.Verkkosivu.

Niemann, Heinrich. 2003. Klassifikation von Mustern. 2. überarbeitete Auflage im Internet. Verkkodokumentti. <http://www5.informatik.unierlangen.de/fileadmin/Persons/NiemannHeinrich/klassifikation-von-mustern/m00-www.pdf>.

Perronnin, Florent. 2008. Universal and Adapted Vocabularies for Generic Visual Categorization, IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 30, no. 7.

Pesonen, Martti E. 2012. Lineaarialgebra. Verkkodokumentti. <www.joensuu.fi/matematiikka/kurssit/Lineaarialgebra/Kurssimateriaali/LAText/LinAlgMoniste2011ab.pdf>. Luettu 1.12.2011.

Programming, MATLAB Version 7.6 (R2008a). Verkkosivu. The MathWorks.<http://www.mathworks.com/help/techdoc/rn/brjk1q5-1.html#brjk1q5-4>. Luettu 4.4.2011.

Property Attributes. Verkkosivu. The MathWorks. < http://www.mathworks.com/help/techdoc/matlab_oop/brjjwby.html>. Luettu 16.3.2011.

Russell, B.C. – Torralba, A. – Freeman, W.T. 2005. LabelMe: The open annotation tool. Verkkosivu. MIT, Computer Science and Artificial Intelligence Laboratory. <http://labelme.csail.mit.edu/guidelines.html>. Luettu 18.4.2012.

Russell, B.C. – Torralba, A. – Murphy, K.P. – Freeman, W.T. 2008. LabelMe: a database and web-based tool for image annotation. International Journal of Computer Vision, vol. 77, no. 1–3, s. 157–173. Verkkodokumentti. <http://people.csail.mit.edu/brussell/research/AIM-2005-025-new.pdf>.

Schiele, Bernt. 2010. Histogram of Oriented Gradients (HOG) & Support Vector Machines (SVM). Verkkodokumentti. <https://tahiti.mis.informatik.tu-darmstadt.de/public/cv_ss10_nopass/cv-ss10-0510-hog-svm-v0.pdf>. Luettu 9.6.2011.

Schölkopf, Bernhard – Smola, Alexander J. 2002. Learning with Kernels. London, England: The MIT Press.

Scoping Classes with Packages. Verkkosivu. The MathWorks. < http://www.mathworks.com/help/techdoc/matlab_oop/brfynt_-1.html>. Luettu 15.4.2011.

Seemann, Edgar. 2007. Pedestrian Detection in Crowded Street Scenes, PhD thesis. Germany:Technische Universität Darmstadt. Verkkodokumentti. <http://www.pedestrian-detection.com/>.

Seppänen, Tapio. 2012. Hahmontunnistus ja neuroverkot. Verkkodokumentti. Oulun yliopisto: Tietotekniikan osasto. <http://www.ee.oulu.fi/research/tklab/courses/521497S/pruju/Chapter2.pdf>. Luettu 7.4.2012.

Shlens, Jonathon. 2009. A Tutorial on Principal Component Analysis. Verkkodokumentti. <http://www.snl.salk.edu/~shlens/pca.pdf>. Luettu 21.1.2012.

SVM–Support Vector Machines, Optimum Separation Hyperplane. Verkkosivu. <http://www.support-vector-machines.org/SVM_osh.html>. Luettu 28.2.2012.

The Java™ Tutorials: About the JFC and Swing. Verkkosivu. <http://download.oracle.com/javase/tutorial/uiswing/start/about.html>. Luettu 31.3.2011.

Tohka, Jussi. 2009. Johdatus hahmontunnistukseen, Tampereen teknillinen yliopisto: Signaalinkäsittelylaitos. Verkkodokumentti. <http://www.cs.tut.fi/~jupeto/jht_lectnotes.pdf>.

Uicontrol callbacks. Verkkosivu. <http://undocumentedmatlab.com/blog/uicontrol-callbacks/>. Luettu 7.4.2011.

Object Management Group–UML. Verkkosivu. Object Management Group. <http://www.uml.org/>. Päivitetty 14.2.2011. Luettu 25.5.2011

Luku 9, Uskottavuuden ominaisuuksia. Verkkodokumentti. <http://mtl.uta.fi/tilasto/mttapu/Luku9.pdf>. Päivitetty 27.2.2007. Luettu 19.5.2011.

Vapnik, Vladimir N. 2000. The Nature of Statistical Learning Theory. Second edition. US: Springer.

Verhagen, C. J. D. M. 1975. Some general remarks about pattern recognition; its definition; its relation with other disciplines; a literature survey. Pattern Recongnition, vol. 7, iss. 3, s. 109–116.

Weber, M. 2000. Unsupervised Learning of Models for Object Recognition. PhD thesis. Pasadena, CA: California Institute of Technology.

Multi-Cue ISM: Asennusohjeet (Ubuntu 10.04 LTS)

Lataa Multi-Cue ISM-sovellus

www-osoitteesta: http://www.vision.ee.ethz.ch/~bleibe/code/ism.html.

1. Luo symboliset linkit code- ja codebooks-hakemistoille

```
ln -s $PWD/code ~/code
ln -s $PWD/codebooks ~/codebooks
```

Huomio: $PWD/code tilalle voi laittaa normaalin polun alkaen omasta työhakemistosta ja varmistaa että linkki toimii.

Huomio: codebooks-hakemisto on hyvä tallentaa ism-feb08-hakemistoon.

2. Tarkasta onko koneeseen asennettu tcsh/csh-komentotulkkia, jos ei asenna

```
sudo apt-get install tcsh
```

tai

```
sudo apt-get install csh
```

3. Vaihda komentotulkki

```
which csh
which tcsh
```

4. Mene sen jälkeen codebooks-hakemistoon ja aja prepare.sh

```
./prepare.sh
```

5. Mene päähakemistoon, jossa on start.sh -tiedosto ja aja se, jotta näet puuttuvat tiedostot ja kirjastot

```
./start.sh
```

puuttuvat tiedostot esimerkiksi,

```
sudo apt-get install libqt3-mt
sudo apt-get install libqt3-mt
sudo apt-get install libstdc++6
```

6. Ubuntu 10.04 -versio käyttää c++ kirjastoa libstdc++6, mutta tässä sovelluksessa tarvitaan libstdc++5:sta. Lataa libstdc++5 osoitteesta: http://packages.debian.org/lenny/i386/libstdc++5/download ja aloita automaattinen asennus.

7. Käynnistä sovellus

```
./start.sh
```

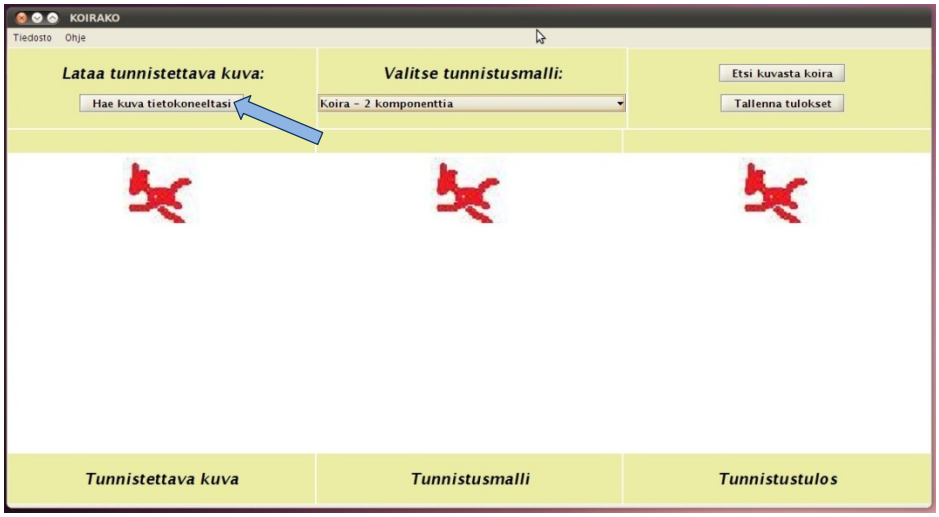
KOIRAKO: Asennusohjeet

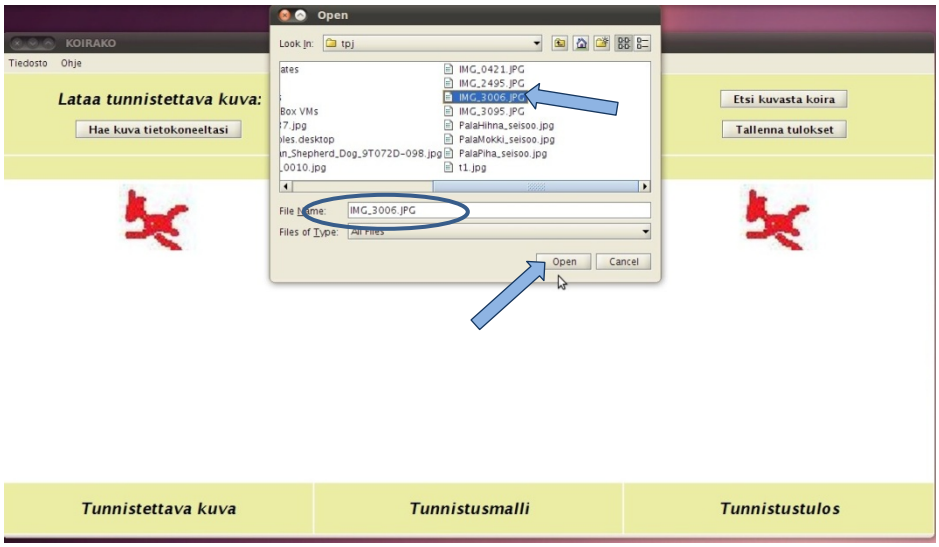
- Lisää KOIRAKO-hakemisto samalle tasolle hakemistorakenteeseen kuin VOC-release 3.1
    - esimerkiksi
    - …/work/KOIRAKO
    - …/work/voc-release3.1
- KOIRAKO-hakemisto näyttää seuraavalta
    - /KOIRAKO/kuvat
    - /KOIRAKO/tulokset
    - /KOIRAKO/AbstraktiNakyma.m
    - /KOIRAKO/AsetusNakyma.m
    - /KOIRAKO/TunnistusNakyma.m
    - /KOIRAKO/KOIRAKOKayttoliittyma.m
    - /KOIRAKO/KOIRAKOKontrolleri.m
    - /KOIRAKO/KOIRAKOSovelluslogiikka.m
    - /KOIRAKO/Tunnistamo.m
    - /KOIRAKO/TunnistettavaKuva.m
    - /KOIRAKO/Tunnistusmalli.m
    - /KOIRAKO/KaynnistaSovellus.m

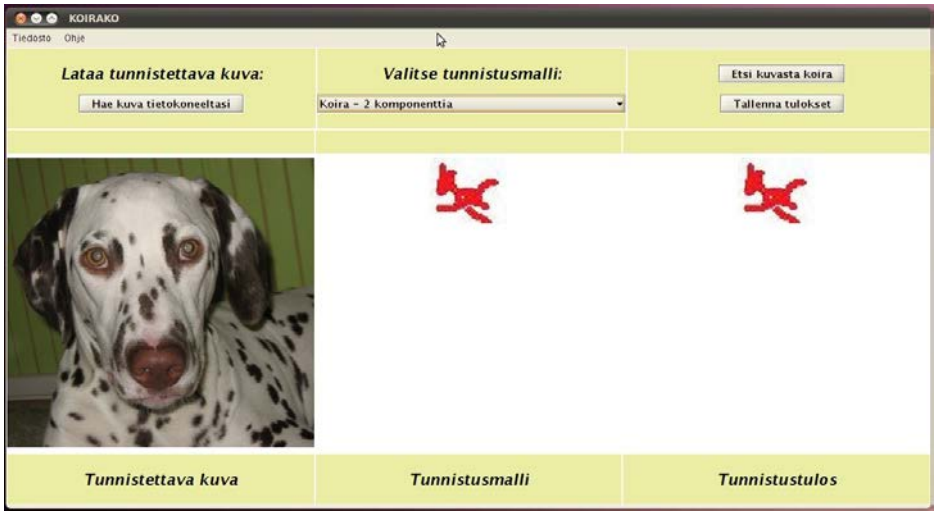
- Lisää voc-release3.1-hakemistoon seuraavat hakemistot ja tiedostot (ne löytyvät Lisaa-VOChakemistosta)
    - /voc-release3.1/mallit/
    - /voc-release3.1/process.m
    - /voc-release3.1/findMaxSizeBbo.m
- KOIRAKO-sovelluksen seuraaviin tiedostoihin tulee muuttaa polut, jotta ne vastaavat oman koneen
hakemistorakennetta. Tiedostot löytyvät KOIRAKO-hakemistosta.
    - TunnistusNakyma.m
        - rivi 7, **polku**-niminen muuttuja
    - KOIRAKOKayttoliittyma.m
        - rivi 21, **imgUrl**-niminen muuttuja
    - KOIRAKOSovelluslogiikka.m
        - rivi 3, **VOCPolku**-niminen muuttuja
        - self.setValittumalliKuvaPolku(…)rivi 50, **malliPolku**-niminen muuttuja
        - rivi 56 load …
        - rivi 66 self.setValittumalliKuvaPolku(…)
        -
- rivi 72 load …
- rivi 82 self.setValittumalliKuvaPolku(…)
- rivi 87 load …
- rivi 97 self.setValittumalliKuvaPolku(…)
- rivi 102 **tuloskuvanPolku**-niminen muuttuja

KOIRAKO: Käyttötapausten simulointi

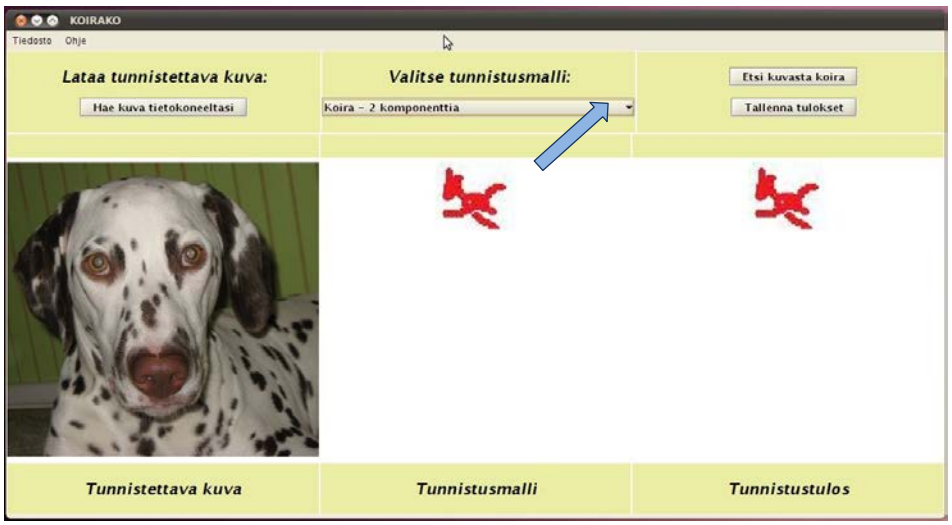
Käyttötapaus Valitse analysoitava kuva (K1)

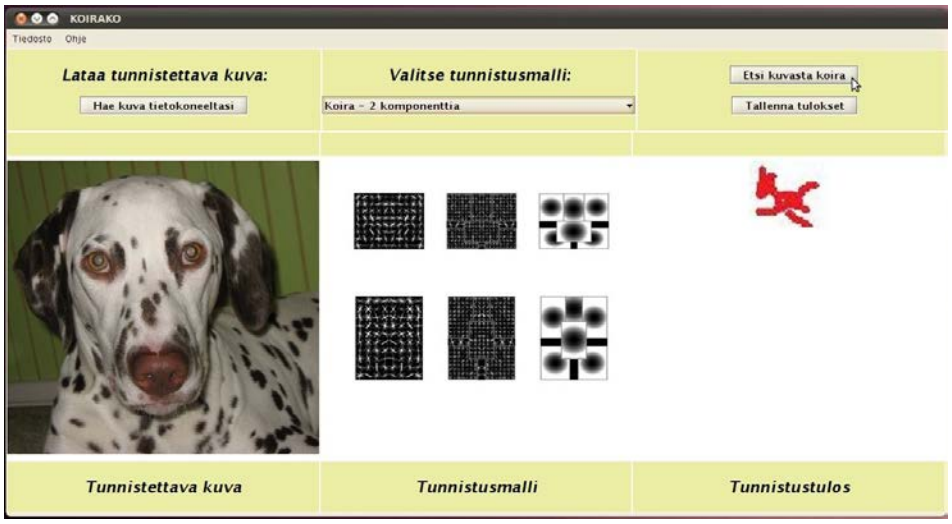




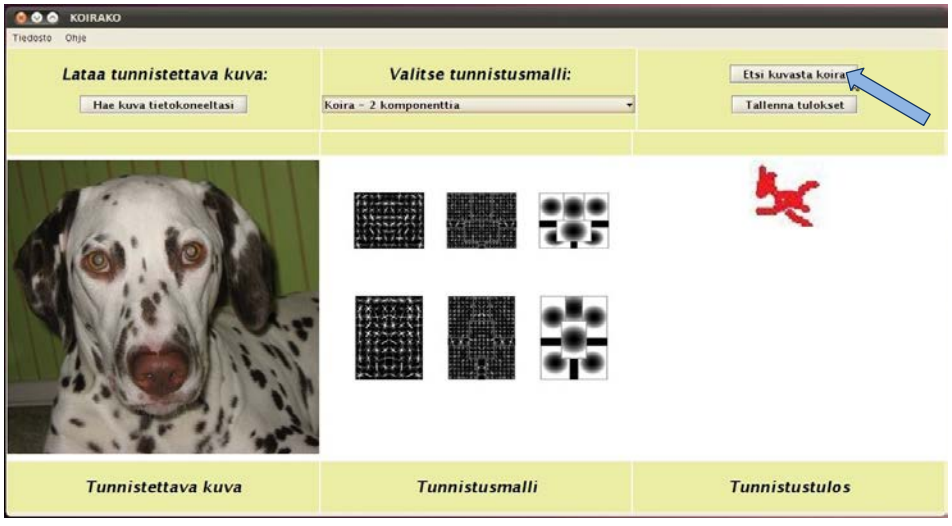


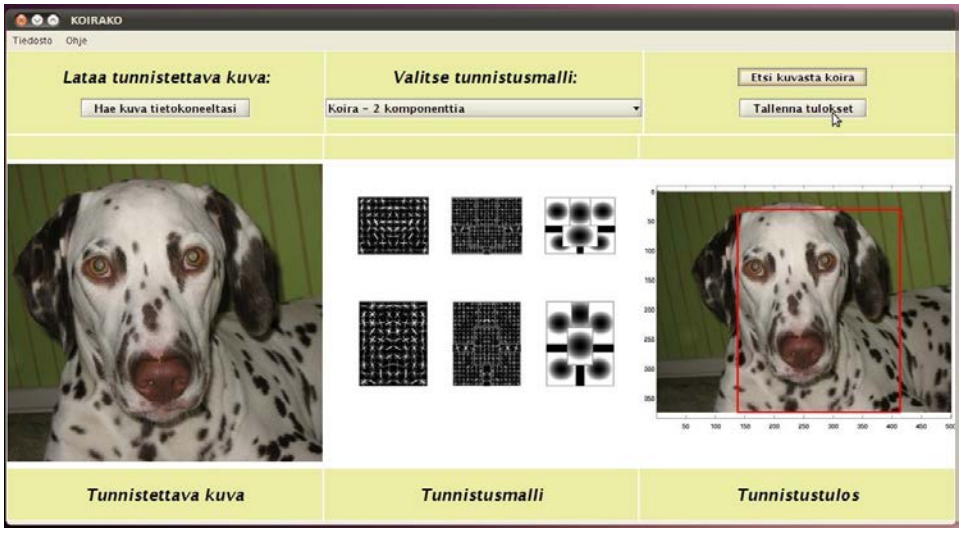
Käyttötapaus Valitse tunnistusmalli (K2)



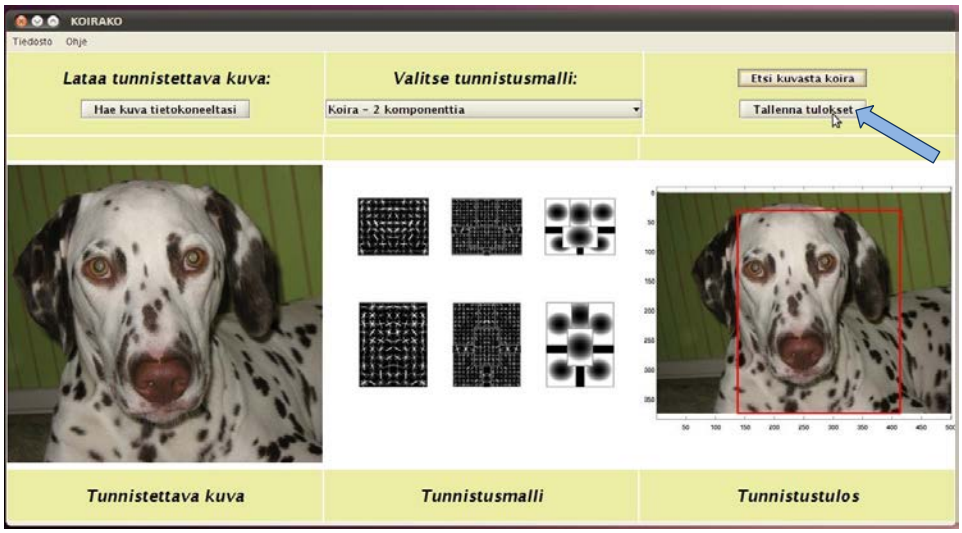


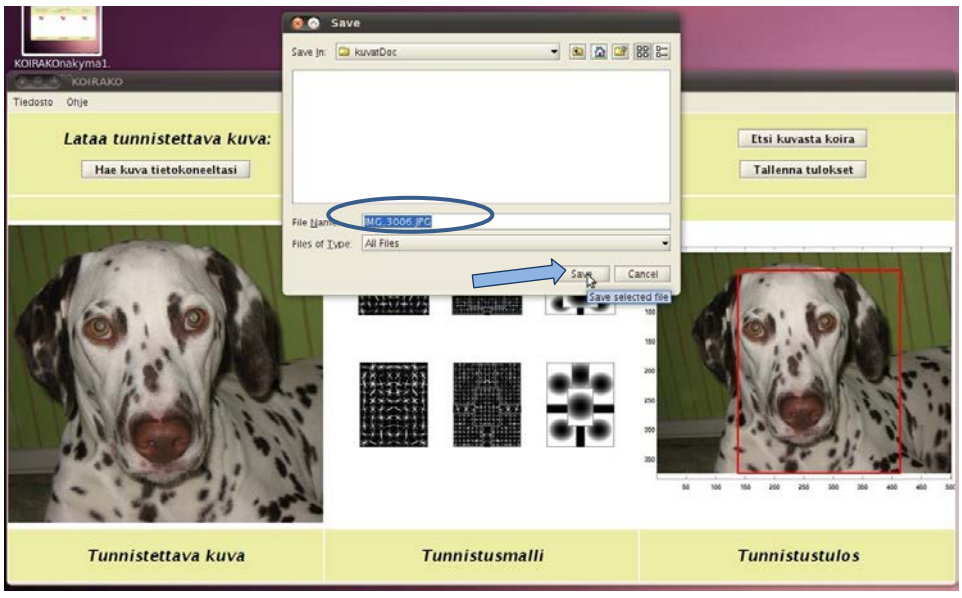
Käyttötapaus Analysoi kuva (K3)



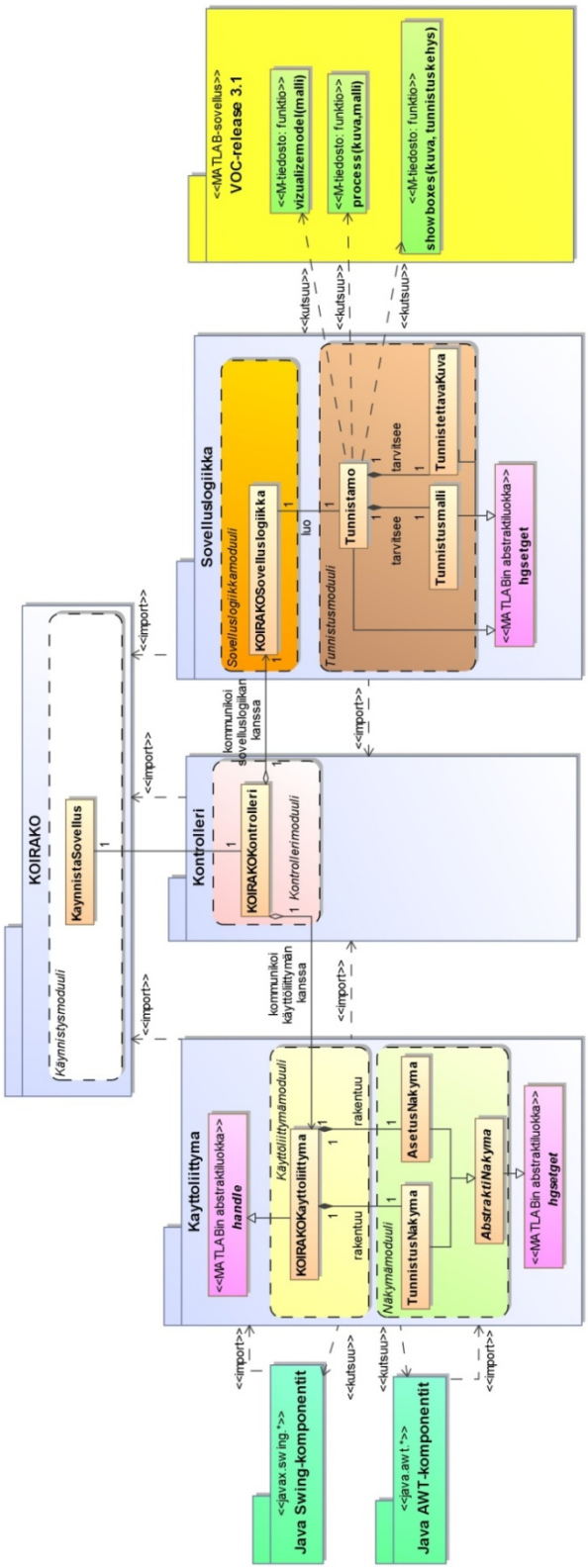


Käyttötapaus Tallenna tunnistustulos (K4)





KOIRAKO: Luokkakaavio



KOIRAKO: Luokka KOIRAKOKayttoliittyma

Kontrollerimoduuli kommunikoi käyttöliittymämoduuliin kuuluvan KOIRAKOKayttoliittyma-luokan kanssa. KOIRAKOKayttoliittyma rakentaa käyttöliittymän muun muassa Swing-komponentteja kutsumalla ja hallinnoi näkymämoduuliin kuuluvia luokkia päivittämällä käyttöliittymää kontrollerimoduulin pyynnöstä.

Kuva 62. Luokan KOIRAKOKayttoliittyma muuttujat ja metodit.

KOIRAKOKayttoliittyma luokan muuttujat ja metodit on listattuna kuvaan 62, jonka jälkeen esitellään ohjelmakoodi.

```
classdef KOIRAKOKayttoliittyma < handle
   properties (Constant)
        % käyttöliittymän tekstit, nimet ja koot
        sovellusNimiStr             = 'KOIRAKO';
        ikkunanLeveysPx             = 1200;
        ikkunanKorkeusPx            = 650;
        tietojaIkkunanLeveysPx    = 250;
        tietojaIkkunanKorkeusPx  = 230;

        tiedostoJMenuStr     = 'Tiedosto';
        ohjeJMenuStr         = 'Ohje';
```

```
        haeKuvaItemStr    = 'Hae kuva...';
        tallennaItemStr     = 'Tallenna';

        suljeItemStr        = 'Sulje sovellus';
        tietojaItemStr      = 'Tietoja KOIRAKOsta';

        tekijaStr           = 'Marika Könönen 0601512';
        pvmStr              = '21.4.2011';
        aiheStr             = 'Hahmontunnistussovellus KOIRAKO';
        versioStr           = 'versio 1.1';
        imgUrl              = '/home/tpj/prog/MATLAB/R2010b/work/KOIRAKO/kuvat/logoPieni.jpg';
        suljeTietojaStr     = 'Sulje';
    end
    properties (SetAccess = private, GetAccess = public)
        % luokkamuuttujat
        asetusnakyma;
        tunnistusnakyma;
        kontrolleri;
        % käyttöliittymän komponentit
        kaliIkkuna          = javax.swing.JFrame;
        sisaltopaneeli      = javax.swing.JPanel;
        borderLayout        = java.awt.BorderLayout;
        % valikon komponentit
        otsikkopalkki       = javax.swing.JMenuBar;
        haeKuvaItem         = javax.swing.JMenuItem(KOIRAKOKayttoliittyma.haeKuvaItemStr);
        tallennaItem        = javax.swing.JMenuItem(KOIRAKOKayttoliittyma.tallennaItemStr);
        suljeItem           = javax.swing.JMenuItem(KOIRAKOKayttoliittyma.suljeItemStr);
        tietojaItem         = javax.swing.JMenuItem(KOIRAKOKayttoliittyma.tietojaItemStr);
        % tietoja-ikkunan komponentit
        tietojaRaami        = javax.swing.JFrame;
        tietojapaneeli      = javax.swing.JPanel;
        tekijaLbl           = javax.swing.JLabel;
        pvmLbl              = javax.swing.JLabel;
        aiheLbl             = javax.swing.JLabel;
        versioLbl           = javax.swing.JLabel;
        imgLbl              = javax.swing.JLabel;
        suljeTietojaBtn     = javax.swing.JButton;
    end
    methods
        function olioKS = KOIRAKOKayttoliittyma()
            try
                % setDefaultCloseOperation(EXIT_ON_CLOSE);
                rakennaKayttoliittyma(olioKS);
%                 disp('KOIRAKOKayttoliittyma konstruktori, ennen testinkutsua');
```

```
%                olioKS.tunnistusnakyma.testi();
%                disp('KOIRAKOKayttoliittyma konstruktori, jälkeen testinkutsua');
            catch exception
                raportti = getReport(exception, 'extended');
                disp(raportti);
            end
        end
        % getterit**********************************************************
        function getAsetusnakyma = get.asetusnakyma(self)
            getAsetusnakyma = self.asetusnakyma;
        end
        function getTunnistusnakyma = get.tunnistusnakyma(self)
            getTunnistusnakyma = self.tunnistusnakyma;
        end
        function getKaliIkkuna = getKaliIkkuna(self)
            getKaliIkkuna = self.kaliIkkuna;
        end
        function getTietojaRaami = getTietojaRaami(self)
            getTietojaRaami = self.tietojaRaami;
        end
        function getHaeKuvaItem = get.haeKuvaItem(self)
            getHaeKuvaItem = self.haeKuvaItem;
        end
        function getTallennaItem = get.tallennaItem(self)
            getTallennaItem = self.tallennaItem;
        end
        function getSuljeItem = get.suljeItem(self)
            getSuljeItem = self.suljeItem;
        end
        function getTietojaItem = get.tietojaItem(self)
            getTietojaItem = self.tietojaItem;
        end
        function getAsetusnakymaHaeKuvaBtn = getAsetusnakymaHaeKuvaBtn(self)
           getAsetusnakymaHaeKuvaBtn = self.asetusnakyma.get.haeKuvaBtn();
        end
        function getAsetusnakymaEtsiKoiraBtn = getAsetusnakymaEtsiKoiraBtn(self)
           getAsetusnakymaEtsiKoiraBtn = self.asetusnakyma.get.etsiKoiraBtn();
        end
        function getAsetusnakymaTallennaTuloksetBtn = getAsetusnakymaTallennaTuloksetBtn(self)
           getAsetusnakymaTallennaTuloksetBtn = self.asetusnakyma.get.tallennaTuloksetBtn();
        end
        function getAsetusnakymaMalliListaCbx = getAsetusnakymaMalliListaCbx(self)
           getAsetusnakymaMalliListaCbx = self.asetusnakyma.get.malliListaCbx();
        end
```

```
function getAsetusnakymaValittuKuvaPolku = getAsetusnakymaValittuKuvaPolku(self)
   getAsetusnakymaValittuKuvaPolku = self.asetusnakyma.get.valittuKuvaPolku();
end
function getTunnistusnakymaValittuKuvaPolku = getTunnistusnakymaValittuKuvaPolku(self)
   getTunnistusnakymaValittuKuvaPolku = self.tunnistusnakyma.get.tunnistettavaKuvaPolku();
end
function getAsetusnakymaAsetuspaneeli = getAsetusnakymaAsetuspaneeli(self)
   getAsetusnakymaAsetuspaneeli = self.asetusnakyma.get.asetuspaneeli();
end
function getTunnistusnakymaTunnistuspaneeli = getTunnistusnakymaTunnistuspaneeli(self)
   getTunnistusnakymaTunnistuspaneeli = self.tunnistusnakyma.get.tunnistuspaneeli();
end
function getTunnistusnakymaTulosKuva = getTunnistusnakymaTulosKuva(self)
   getTunnistusnakymaTulosKuva = self.tunnistusnakyma.get.tulosKuva();
end
function getTunnistusnakymaTulosKuvaPolku = getTunnistusnakymaTulosKuvaPolku(self)
   getTunnistusnakymaTulosKuvaPolku = self.tunnistusnakyma.get.tulosKuvaPolku();
end
% setterit*************************************************************
function setTunnistettavaKuvaPolku(self,tunnistettavaKuva)
   disp(tunnistettavaKuva);
   self.tunnistusnakyma.setTunnistettavaKuvaPolku(tunnistettavaKuva);
end
function setTulosKuvaPolku(self,tulosKuva)
   self.tunnistusnakyma.setTulosKuvaPolku(tulosKuva);
end
% käyttöliittymän kokoaminen*******************************************
function rakennaKayttoliittyma(olioKS)
    import javax.swing.*
    import java.awt.*;

    % luodaan luokkailmentymät
    olioKS.asetusnakyma     = AsetusNakyma();
    olioKS.tunnistusnakyma = TunnistusNakyma();
    % valmistellaan JFrame
    olioKS.kaliIkkuna      = JFrame(KOIRAKOKayttoliittyma.sovellusNimiStr);
    olioKS.sisaltopaneeli = olioKS.kaliIkkuna.getContentPane;
     % asetetaan layout
    olioKS.sisaltopaneeli.setLayout(olioKS.borderLayout);

    % lisätään elementit paneeliin
    olioKS.sisaltopaneeli.add(olioKS.getAsetusnakymaAsetuspaneeli(), BorderLayout.PAGE_START);
    olioKS.sisaltopaneeli.add(olioKS.getTunnistusnakymaTunnistuspaneeli(), BorderLayout.CENTER);
```

```
        % luodaan otsikkopalkki JMenuBar
        % Tiedosto-valikko
        valikko = JMenu(olioKS.tiedostoJMenuStr);
        valikko.setMnemonic(KeyEvent.VK_A);
        valikko.add(olioKS.haeKuvaItem);
        valikko.add(olioKS.tallennaItem);
        valikko.add(olioKS.suljeItem);
        olioKS.otsikkopalkki.add(valikko);
        % Ohje-valikko
        valikko = JMenu(olioKS.ohjeJMenuStr);
        valikko.setMnemonic(KeyEvent.VK_A);
        valikko.add(olioKS.tietojaItem);
        olioKS.otsikkopalkki.add(valikko);
        olioKS.kaliIkkuna.setJMenuBar(olioKS.otsikkopalkki);

        % ulkoasu
        olioKS.setUlkoasu();
        % rakenna tietojaikkuna
        olioKS.naytaTietojaKOIRAKOSTA();

        % asetetaan käyttöliittymäikkunalle koko
        olioKS.kaliIkkuna.setSize(olioKS.ikkunanLeveysPx, olioKS.ikkunanKorkeusPx);
        olioKS.kaliIkkuna.setVisible(true);
    end
% callback-toimintojen asettaminen kontrolleriin**********************
function asetaKontrolleri(self,kontrolleri)
    % testausta
    onkoLuokanOlio = isa(kontrolleri, 'KOIRAKOKontrolleri');
    disp ('asetaKontrolleri:');
    disp (onkoLuokanOlio);

    self.kontrolleri = kontrolleri;
    disp('Kalin metodi: asetaKontrolleri');
    haeKuvaBtnHndl           = handle (self.getAsetusnakymaHaeKuvaBtn,         'CallbackProperties');
    valitseTunnistusmalliHndl = handle (self.getAsetusnakymaMalliListaCbx,        'CallbackProperties');
    etsiKoiraBtnHndl          = handle (self.getAsetusnakymaEtsiKoiraBtn,         'CallbackProperties');
    tallennaTuloksetBtnHndl = handle (self.getAsetusnakymaTallennaTuloksetBtn,'CallbackProperties');
    haeKuvaItemHndl         = handle (self.haeKuvaItem,                        'CallbackProperties');
    tallennaItemHndl         = handle (self.tallennaItem,                       'CallbackProperties');
    suljeItemHndl            = handle (self.suljeItem,                          'CallbackProperties');
    tietojaItemHndl          = handle (self.tietojaItem,                        'CallbackProperties');
    suljeTietojaBtnHndl      = handle (self.suljeTietojaBtn,                     'CallbackProperties');
```

```
    set(haeKuvaBtnHndl,           'ActionPerformedCallback', @kontrolleri.haeKuvaBtnClick);
    set(valitseTunnistusmalliHndl, 'ActionPerformedCallback', @kontrolleri.valitseTunnistusmalli);
    set(etsiKoiraBtnHndl,          'ActionPerformedCallback', @kontrolleri.etsiKoiraBtnClick);
    set(tallennaTuloksetBtnHndl,  'ActionPerformedCallback', @kontrolleri.tallennaTuloksetBtnClick);
    set(haeKuvaItemHndl,          'ActionPerformedCallback', @kontrolleri.haeKuvaBtnClick);
    set(tallennaItemHndl,         'ActionPerformedCallback', @kontrolleri.tallennaTuloksetBtnClick);
    set(suljeItemHndl,            'ActionPerformedCallback', @kontrolleri.suljeItmClick);
    set(tietojaItemHndl,          'ActionPerformedCallback', @kontrolleri.tietojaItmClick);
    set(suljeTietojaBtnHndl,      'ActionPerformedCallback', @kontrolleri.suljeTietojaBtnClick);
end
% haeKuvaBtn-toiminnon metodit****************************************
function avaaTiedostonValitsija(self)
    self.asetusnakyma.avaaTiedostonValitsija();
end
function tallennaValittuKuvaPolkuTunnistusNakymaan(self)
    % hae asetusnakymasta valitun kuvan polku
    valittuKuvaPolku = self.getAsetusnakymaValittuKuvaPolku();
    disp(valittuKuvaPolku);
    % tallenna se tunnistunnakymaan
    self.setTunnistettavaKuvaPolku(valittuKuvaPolku);
end
function vaihdaTunnistettavaKuva(self, uusiKuva)
    self.tunnistusnakyma.vaihdaKuva(uusiKuva);
    disp(uusiKuva);
end
% valitseTunnistusmalli-toiminnon metodit*****************************
function malliIndx = haeValittuMalliIndx(self)
    malliIndx = self.asetusnakyma.valittuMalliIndx();
end
function vaihdaTunnistusmalliKuva(self, uusiMalli)
    self.tunnistusnakyma.vaihdaMalli(uusiMalli);
end
% etsiKoiraBtn-toiminnon metodit**************************************
function vaihdaTuloskuva(self, uusiTulos)
    self.tunnistusnakyma.vaihdaTulos(uusiTulos);
end
% tallennaTuloksetBtn-toiminnon metodit*******************************
function tallennaTulosTiedostoon(self,tulosKuva)
    self.asetusnakyma.tallennaTiedostoon(tulosKuva);
end
% muutetaan näkymien ulkoasua*****************************************
function setUlkoasu(self)
    self.asetusnakyma.setUlkoasu();
    self.tunnistusnakyma.setUlkoasu();
```

```
    end
    % tietoja-ikkuna*******************************************************
    function naytaTietojaKOIRAKOSTA(self)
        import java.awt.*;
        import javax.swing.*;

        self.tietojaRaami    = javax.swing.JFrame(self.tietojaItemStr);
        self.tietojapaneeli = self.tietojaRaami.getContentPane();
        self.tietojapaneeli.setLayout(javax.swing.BoxLayout(self.tietojapaneeli, javax.swing.BoxLayout.Y_AXIS));

        self.tekijaLbl             = javax.swing.JLabel(self.tekijaStr);
        self.pvmLbl                = javax.swing.JLabel(self.pvmStr);
        self.aiheLbl               = javax.swing.JLabel(self.aiheStr);
        self.versioLbl             = javax.swing.JLabel(self.versioStr);
        self.imgLbl                = javax.swing.JLabel(javax.swing.ImageIcon(self.imgUrl));
        self.suljeTietojaBtn       = javax.swing.JButton(self.suljeTietojaStr);
        % lisätään komponentit
        self.tietojapaneeli.add(Box.createRigidArea(Dimension(200, 10)));
        self.tietojapaneeli.add(self.tekijaLbl);
        self.tietojapaneeli.add(Box.createRigidArea(Dimension(200, 7)));
        self.tietojapaneeli.add(self.pvmLbl);
        self.tietojapaneeli.add(Box.createRigidArea(Dimension(200,7)));
        self.tietojapaneeli.add(self.aiheLbl);
        self.tietojapaneeli.add(Box.createRigidArea(Dimension(200, 5)));
        self.tietojapaneeli.add(self.versioLbl);
        self.tietojapaneeli.add(Box.createRigidArea(Dimension(200, 10)));
        self.tietojapaneeli.add(self.imgLbl);
        self.tietojapaneeli.add(Box.createRigidArea(Dimension(200, 10)));
        self.tietojapaneeli.add(self.suljeTietojaBtn);
        % muokataan asettelua
        self.tekijaLbl.setAlignmentX(Component.CENTER_ALIGNMENT);
        self.pvmLbl.setAlignmentX(Component.CENTER_ALIGNMENT);
        self.aiheLbl.setAlignmentX(Component.CENTER_ALIGNMENT);
        self.versioLbl.setAlignmentX(Component.CENTER_ALIGNMENT);
        self.imgLbl.setAlignmentX(Component.CENTER_ALIGNMENT);
        self.suljeTietojaBtn.setAlignmentX(Component.CENTER_ALIGNMENT);

        self.tietojaRaami.setSize(self.tietojaIkkunanLeveysPx, self.tietojaIkkunanKorkeusPx);
        self.tietojaRaami.setVisible(false);
    end
  end
end
```

KOIRAKO: Luokka AbstraktiNakyma

AbstraktiNakyma on näkymämoduulin abstraktiluokka, jossa määritellään KOIRAKOn ulkoasuun liittyviä asioita, kuten esimerkiksi paneelien värejä ja painikkeiden fontteja.

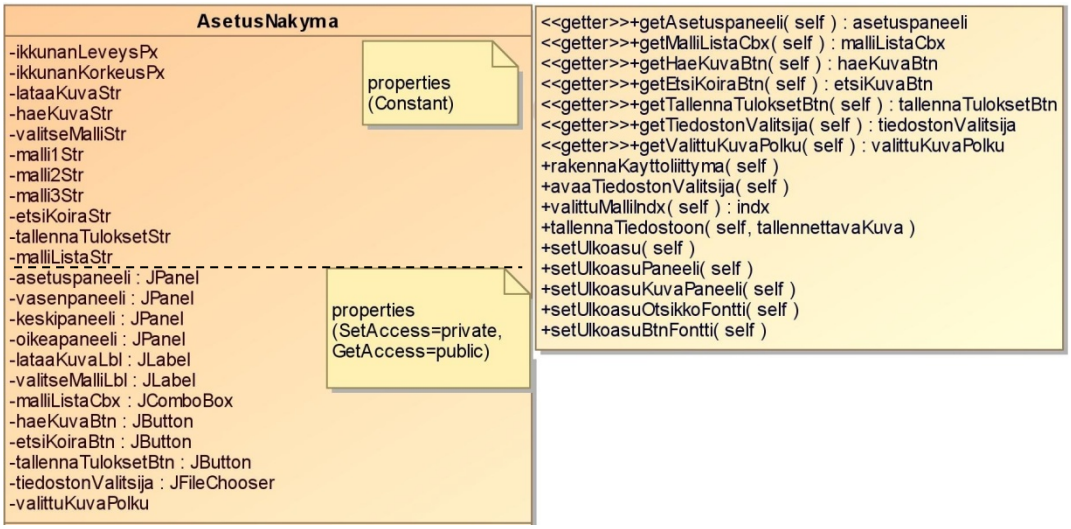
Kuva 63. Luokan AbstraktiNakyma metodit. Abstraktit metodit on kuvassa alleviivattuina.

Kuvassa 63 on AbstraktiNakyma luokan metodit, joiden jälkeen liitteessä esitetään ohjelmakoodi kokonaisuudessaan.

```
classdef AbstraktiNakyma < hgsetget
    methods
        % ulkoasun muokkaus metodit*****************************************
        function setUlkoasuPaneeli(paneeli)
            paneeli.setBackground(java.awt.Color(int16(234),int16(237),int16(163)));
        end
        function setUlkoasuKuvaPaneeli(paneeli)
            paneeli.setBackground(java.awt.Color(int16(255),int16(255),int16(255)));
        end
        function setUlkoasuOtsikkoFontti(label)
            label.setFont(java.awt.Font('Helvetica',java.awt.Font.BOLD + java.awt.Font.ITALIC, 20));
        end
        function setUlkoasuBtnFontti(btn)
            btn.setFont(java.awt.Font('Helvetica', java.awt.Font.BOLD, 14));
        end
    end
    methods(Abstract, Static)
        rakennaKayttoliittyma(self);
        setUlkoasu(self);
    end
end
```

KOIRAKO: Luokka AsetusNakyma

Näkymämoduuliin kuuluva luokka AsetusNakyma rakentaa käyttöliittymään kuuluvan asetusnäkymän komponentit. AsetusNakyma-luokka perii AbstraktinNakyma-luokan ja saa sieltä ulkoasuasetuksia.



Kuva 64. AsetusNakyma luokan muuttujat ja metodit.

AsetusNakyma luokan muuttujia ja metodeja havainnollistetaan kuvassa 64, minkä jälkeen esitellään luokan ohjelmakoodi kokonaisuudessaan.

```
classdef AsetusNakyma < AbstraktiNakyma
  properties (Constant)
      % käyttöliittymän tekstit, nimet ja koot% tekstit ja muut nimet
      sovellusNimiStr     = 'KOIRAKO';
      ikkunanLeveysPx    = 1200;
      ikkunanKorkeusPx  = 150;

      lataaKuvaStr         = 'Lataa tunnistettava kuva:';
      haeKuvaStr           = 'Hae kuva tietokoneeltasi';
      valitseMalliStr       = 'Valitse tunnistusmalli:';
      malli1Str             = 'Koira - 2 komponenttia';
      malli2Str             = 'Koira - 4 komponenttia';
      malli3Str             = 'Koira - 6 komponenttia';
      etsiKoiraStr          = 'Etsi kuvasta koira';
      tallennaTuloksetStr = 'Tallenna tulokset';
      malliListaStr = {AsetusNakyma.malli1Str, AsetusNakyma.malli2Str, AsetusNakyma.malli3Str};
  end
```

```
properties (SetAccess = private, GetAccess = public)
    % paneelit
     asetuspaneeli        = javax.swing.JPanel;
     vasenpaneeli         = javax.swing.JPanel;
     keskipaneeli         = javax.swing.JPanel;
     oikeapaneeli         = javax.swing.JPanel;

     lataaKuvaLbl         = javax.swing.JLabel(AsetusNakyma.lataaKuvaStr);
     valitseMalliLbl      = javax.swing.JLabel(AsetusNakyma.valitseMalliStr);

     malliListaCbx        = javax.swing.JComboBox(AsetusNakyma.malliListaStr);
     haeKuvaBtn           = javax.swing.JButton(AsetusNakyma.haeKuvaStr);
     etsiKoiraBtn         = javax.swing.JButton(AsetusNakyma.etsiKoiraStr);
     tallennaTuloksetBtn = javax.swing.JButton(AsetusNakyma.tallennaTuloksetStr);

     tiedostonValitsija    = javax.swing.JFileChooser;
     valittuKuvaPolku;
end
 methods
   % konstruktori*************************************************
   function olioAN = AsetusNakyma()
       rakennaKayttoliittyma(olioAN);
   end
   % getterit*****************************************************
   function getAsetuspaneeli = get.asetuspaneeli(self)
       getAsetuspaneeli = self.asetuspaneeli;
   end
   function getMalliListaCbx = get.malliListaCbx(self)
       getMalliListaCbx = self.malliListaCbx;
   end
   function getHaeKuvaBtn = get.haeKuvaBtn(self)
       getHaeKuvaBtn = self.haeKuvaBtn;
   end
   function getEtsiKoiraBtn = get.etsiKoiraBtn(self)
       getEtsiKoiraBtn = self.etsiKoiraBtn;
   end
   function getTallennaTuloksetBtn = get.tallennaTuloksetBtn(self)
       getTallennaTuloksetBtn = self.tallennaTuloksetBtn;
   end
   function getTiedostonValitsija = get.tiedostonValitsija(self)
       getTiedostonValitsija = self.tiedostonValitsija;
   end
   function getValittuKuvaPolku = get.valittuKuvaPolku(self)
       getValittuKuvaPolku = self.valittuKuvaPolku;
```

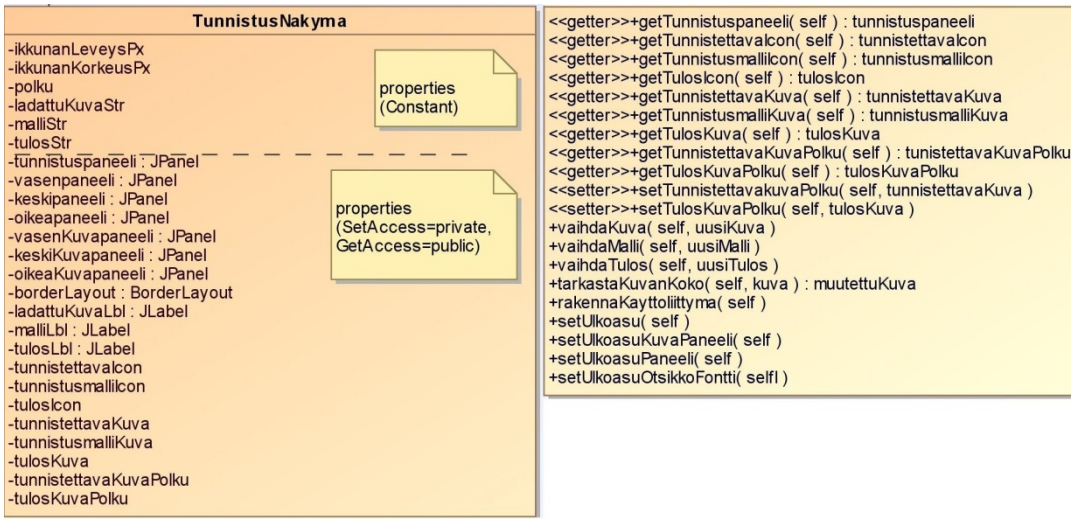
```
end
% käyttöliittymän kokoaminen****************************************
function rakennaKayttoliittyma(olioAN)
    import javax.swing.*
    import java.awt.*;

    olioAN.asetuspaneeli.setLayout(BoxLayout(olioAN.asetuspaneeli, BoxLayout.X_AXIS));
    olioAN.vasenpaneeli.setLayout(BoxLayout(olioAN.vasenpaneeli, BoxLayout.Y_AXIS));
    olioAN.keskipaneeli.setLayout(BoxLayout(olioAN.keskipaneeli, BoxLayout.Y_AXIS));
    olioAN.oikeapaneeli.setLayout(BoxLayout(olioAN.oikeapaneeli, BoxLayout.Y_AXIS));

    % lisätään elementit paneeleihin
    olioAN.asetuspaneeli.add(olioAN.vasenpaneeli, BorderLayout.LINE_START);
    olioAN.asetuspaneeli.add(olioAN.keskipaneeli, BorderLayout.CENTER);
    olioAN.asetuspaneeli.add(olioAN.oikeapaneeli, BorderLayout.LINE_END);

    olioAN.vasenpaneeli.add(Box.createRigidArea(Dimension(400, 20)));
    olioAN.vasenpaneeli.add(olioAN.lataaKuvaLbl);
    olioAN.vasenpaneeli.add(Box.createRigidArea(Dimension(400, 15)));
    olioAN.vasenpaneeli.add(olioAN.haeKuvaBtn);
    olioAN.vasenpaneeli.add(Box.createRigidArea(Dimension(400, 20)));

    olioAN.keskipaneeli.add(Box.createRigidArea(Dimension(400, 20)));
    olioAN.keskipaneeli.add(olioAN.valitseMalliLbl);
    olioAN.keskipaneeli.add(Box.createRigidArea(Dimension(400, 15)));
    olioAN.keskipaneeli.add(olioAN.malliListaCbx);
    olioAN.keskipaneeli.add(Box.createRigidArea(Dimension(400, 20)));

    olioAN.oikeapaneeli.add(Box.createRigidArea(Dimension(400, 20)));
    olioAN.oikeapaneeli.add(olioAN.etsiKoiraBtn);
    olioAN.oikeapaneeli.add(Box.createRigidArea(Dimension(400, 15)));
    olioAN.oikeapaneeli.add(olioAN.tallennaTuloksetBtn);
    olioAN.oikeapaneeli.add(Box.createRigidArea(Dimension(400, 20)));

    % korjataan kuvien ja labeleiden asettelua
    olioAN.lataaKuvaLbl.setAlignmentX(Component.CENTER_ALIGNMENT);
    olioAN.haeKuvaBtn.setAlignmentX(Component.CENTER_ALIGNMENT);
    olioAN.valitseMalliLbl.setAlignmentX(Component.CENTER_ALIGNMENT);
    olioAN.malliListaCbx.setAlignmentX(Component.CENTER_ALIGNMENT);
    olioAN.etsiKoiraBtn.setAlignmentX(Component.CENTER_ALIGNMENT);
    olioAN.tallennaTuloksetBtn.setAlignmentX(Component.CENTER_ALIGNMENT);

    olioAN.vasenpaneeli.setBorder(BorderFactory.createLineBorder (Color.white, 1));
    olioAN.keskipaneeli.setBorder(BorderFactory.createLineBorder (Color.white, 1));
```

```
        olioAN.oikeapaneeli.setBorder(BorderFactory.createLineBorder (Color.white, 1));

        % käyttöliittymäikkunan asetuksia
        olioAN.asetuspaneeli.setSize(olioAN.ikkunanLeveysPx, olioAN.ikkunanKorkeusPx);
end
% komponenttien päivittämistä*****************************************
function avaaTiedostonValitsija(self)
    % testataan metodin toimivuutta
    onkoLuokanOlio = isa(self, 'AsetusNakyma');
    disp ('avaaTiedostonValitsija: ');
    disp (onkoLuokanOlio);

    palautusArvo     = self.tiedostonValitsija.showOpenDialog(self.asetuspaneeli);
    if (palautusArvo == javax.swing.JFileChooser.APPROVE_OPTION)
        disp('avaaTiedostonValitsija-if');
        self.valittuKuvaPolku = self.tiedostonValitsija.getSelectedFile();
        % muutetaan muuttujan tyyppi Javan Filesta
        self.valittuKuvaPolku = self.valittuKuvaPolku.getPath;
        disp(self.valittuKuvaPolku);
    end
    disp('avaaTiedostonValitsija-if jälkeen');
end
function indx = valittuMalliIndx(self)
    indx = self.malliListaCbx.getSelectedIndex();
end
function tallennaTiedostoon(self, tallennettavaKuva)
    palautusArvo = self.tiedostonValitsija.showSaveDialog(self.asetuspaneeli);
     if (palautusArvo == javax.swing.JFileChooser.APPROVE_OPTION)
        valittuTallennuspolku = self.tiedostonValitsija.getSelectedFile();
        % muutetaan muuttujan tyyppi Javan Filesta
        valittuTallennuspolku = valittuTallennuspolku.getPath;
        disp('tallennaTiedostoon');disp(valittuTallennuspolku);
        kuva = imread(tallennettavaKuva);
        imwrite(kuva,char(valittuTallennuspolku));
    end
end
% ulkoasu AbstraktiNakyma-luokasta************************************
function setUlkoasu(self)
    self.setUlkoasuPaneeli();
    self.setUlkoasuOtsikkoFontti();
    self.setUlkoasuBtnFontti();
end
function setUlkoasuPaneeli(self)
    setUlkoasuPaneeli@AbstraktiNakyma(self.vasenpaneeli);
```

```
            setUlkoasuPaneeli@AbstraktiNakyma(self.keskipaneeli);
            setUlkoasuPaneeli@AbstraktiNakyma(self.oikeapaneeli);
        end
        function setUlkoasuOtsikkoFontti(self)
            setUlkoasuOtsikkoFontti@AbstraktiNakyma(self.lataaKuvaLbl);
            setUlkoasuOtsikkoFontti@AbstraktiNakyma(self.valitseMalliLbl);
            setUlkoasuOtsikkoFontti@AbstraktiNakyma(self.oikeapaneeli);
        end
        function setUlkoasuBtnFontti(self)
            setUlkoasuBtnFontti@AbstraktiNakyma(self.haeKuvaBtn);
            setUlkoasuBtnFontti@AbstraktiNakyma(self.etsiKoiraBtn);
            setUlkoasuBtnFontti@AbstraktiNakyma(self.tallennaTuloksetBtn);
            setUlkoasuBtnFontti@AbstraktiNakyma(self.malliListaCbx);
        end
    end
end
```

KOIRAKO: Luokka TunnistusNakyma

Näkymämoduuliin kuuluva luokka TunnistusNakyma rakentaa käyttöliittymään kuuluvan tunnistusnäkymän komponentit. TunnistusNakyma-luokka perii AbstraktinNakyma-luokan ja saa sieltä ulkoasuasetuksia.



Kuva 65. TunnistusNakyma luokan muuttujat ja metodit.

TunnistusNakyma-luokan muuttujat ja metodit on nähtävillä kuvassa 65, jonka jälkeen esitellään luokan ohjelmakoodi kokonaisuudessaan.

```
classdef TunnistusNakyma < AbstraktiNakyma
   properties (Constant)
      % käyttöliittymän tekstit, nimet ja koot
      sovellusNimiStr         = 'KOIRAKO';
      ikkunanLeveysPx         = 1200;
      ikkunanKorkeusPx        = 500;
      polku                   = '/home/tpj/prog/MATLAB/R2010b/work/KOIRAKO/kuvat/logo.jpg';
      kuvanLeveysPx           = 100;
      kuvanKorkeusPx          = 100;
      paneelinLeveysPx        = 400;
      paneelinKorkeusPx       = 400;

      ladattuKuvaStr          = 'Tunnistettava kuva';
      malliStr                = 'Tunnistusmalli';
      tulosStr                = 'Tunnistustulos';
%       tulosPosStr           = 'Kuvasta löytyi koira';
%       tulosNegStr           = 'Kuvasta ei löytynyt koiraa';
%       tulosPuuttuuStr       = 'Ei tunnistettavaa kuvaa';
```

```
    end
    properties (SetAccess = private, GetAccess = public)
        % Paneelit, labelit, layoutit ja muut muuttujat
        tunnistuspaneeli    = javax.swing.JPanel;
        vasenpaneeli        = javax.swing.JPanel;
        keskipaneeli        = javax.swing.JPanel;
        oikeapaneeli        = javax.swing.JPanel;
        vasenKuvapaneeli = javax.swing.JPanel;
        keskiKuvapaneeli  = javax.swing.JPanel;
        oikeaKuvapaneeli  = javax.swing.JPanel;

        borderLayout       = java.awt.BorderLayout;

        ladattuKuvaLbl     = javax.swing.JLabel(TunnistusNakyma.ladattuKuvaStr);
        malliLbl           = javax.swing.JLabel(TunnistusNakyma.malliStr);
        tulosLbl           = javax.swing.JLabel(TunnistusNakyma.tulosStr);

%       tulosPosLbl       = javax.swing.JLabel(TunnistusNakyma.tulosPosStr);
%       tulosNegLbl      = javax.swing.JLabel(TunnistusNakyma.tulosNegStr);
%       tulosPuuttuuLbl = javax.swing.JLabel(TunnistusNakyma.tulosPuuttuuStr);

        tunnistettavaIcon;
        tunnistusmalliIcon;
        tulosIcon;

        tunnistettavaKuva;
        tunnistusmalliKuva;
        tulosKuva;

        tunnistettavaKuvaPolku;
        tulosKuvaPolku;
    end

    methods
        % konstruktori*************************************************
        function olioTN = TunnistusNakyma()
            rakennaKayttoliittyma(olioTN);
        end
        % getterit*****************************************************
        function getTunnistuspaneeli = get.tunnistuspaneeli(self)
            getTunnistuspaneeli = self.tunnistuspaneeli;
        end
        function getTunnistettavaIcon = get.tunnistettavaIcon(self)
            getTunnistettavaIcon = self.tunnistettavaIcon;
```

```
end
function getTunnistusmalliIcon = get.tunnistusmalliIcon(self)
    getTunnistusmalliIcon = self.tunnistusmalliIcon;
end
function getTulosIcon = get.tulosIcon(self)
    getTulosIcon = self.tulosIcon;
end
function getTunnistettavaKuva = get.tunnistettavaKuva(self)
    getTunnistettavaKuva = self.tunnistettavaKuva;
end
function getTunnistusmalliKuva = get.tunnistusmalliKuva(self)
    getTunnistusmalliKuva = self.tunnistusmalliKuva;
end
function getTulosKuva = get.tulosKuva(self)
    getTulosKuva = self.tulosKuva;
end
function getTunnistettavaKuvaPolku = get.tunnistettavaKuvaPolku(self)
    getTunnistettavaKuvaPolku = self.tunnistettavaKuvaPolku;
end
function getTulosKuvaPolku = get.tulosKuvaPolku(self)
    getTulosKuvaPolku = self.tulosKuvaPolku;
end
% setterit************************************************************
function setTunnistettavaKuvaPolku(self, tunnistettavaKuva)
    self.tunnistettavaKuvaPolku = tunnistettavaKuva;
end
function setTulosKuvaPolku(self, tulosKuva)
    self.tulosKuvaPolku = tulosKuva;
end
% käyttöliittymän kokoaminen*************************************************
function rakennaKayttoliittyma(olioTN)
    import javax.swing.*
    import java.awt.*;

  % lisätään layoutit
  olioTN.tunnistuspaneeli.setLayout(olioTN.borderLayout);
  olioTN.vasenpaneeli.setLayout(BoxLayout(olioTN.vasenpaneeli, BoxLayout.Y_AXIS));
  olioTN.keskipaneeli.setLayout(BoxLayout(olioTN.keskipaneeli, BoxLayout.Y_AXIS));
  olioTN.oikeapaneeli.setLayout(BoxLayout(olioTN.oikeapaneeli, BoxLayout.Y_AXIS));

  olioTN.tunnistettavaIcon   = javax.swing.ImageIcon(olioTN.polku);
  olioTN.tunnistusmalliIcon  = javax.swing.ImageIcon(olioTN.polku);
  olioTN.tulosIcon           = javax.swing.ImageIcon(olioTN.polku);
```

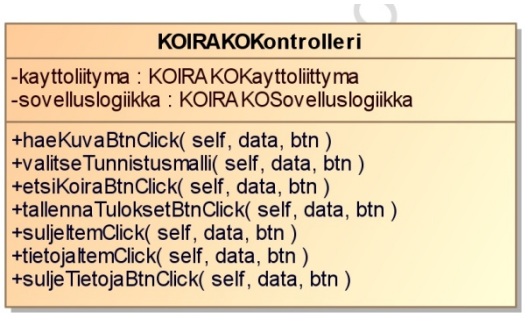
```
olioTN.tunnistettavaKuva    = javax.swing.JLabel(olioTN.tunnistettavaIcon);
olioTN.tunnistusmalliKuva  = javax.swing.JLabel(olioTN.tunnistusmalliIcon);
olioTN.tulosKuva             = javax.swing.JLabel(olioTN.tulosIcon);


% lisätään elementit paneeleihin
olioTN.tunnistuspaneeli.add(olioTN.vasenpaneeli, BorderLayout.LINE_START);
olioTN.tunnistuspaneeli.add(olioTN.keskipaneeli, BorderLayout.CENTER);
olioTN.tunnistuspaneeli.add(olioTN.oikeapaneeli, BorderLayout.LINE_END);

olioTN.vasenpaneeli.add(Box.createRigidArea(Dimension(300, 30)));
olioTN.vasenpaneeli.add(olioTN.vasenKuvapaneeli);
olioTN.vasenKuvapaneeli.add(olioTN.tunnistettavaKuva);
olioTN.vasenpaneeli.add(Box.createRigidArea(Dimension(300, 20)));
olioTN.vasenpaneeli.add(olioTN.ladattuKuvaLbl);
olioTN.vasenpaneeli.add(Box.createRigidArea(Dimension(300, 20)));

olioTN.keskipaneeli.add(Box.createRigidArea(Dimension(300, 30)));
olioTN.keskipaneeli.add(olioTN.keskiKuvapaneeli);
olioTN.keskiKuvapaneeli.add(olioTN.tunnistusmalliKuva);
olioTN.keskipaneeli.add(Box.createRigidArea(Dimension(300, 20)));
olioTN.keskipaneeli.add(olioTN.malliLbl);
olioTN.keskipaneeli.add(Box.createRigidArea(Dimension(300, 20)));

olioTN.oikeapaneeli.add(Box.createRigidArea(Dimension(300, 30)));
olioTN.oikeapaneeli.add(olioTN.oikeaKuvapaneeli);
olioTN.oikeaKuvapaneeli.add(olioTN.tulosKuva);
olioTN.oikeapaneeli.add(Box.createRigidArea(Dimension(300, 20)));
olioTN.oikeapaneeli.add(olioTN.tulosLbl);
olioTN.oikeapaneeli.add(Box.createRigidArea(Dimension(300, 20)));

% korjataan paneelien kokoa, väriä ja muuta
olioTN.vasenpaneeli.setPreferredSize(Dimension(olioTN.paneelinLeveysPx, 0));
olioTN.vasenpaneeli.setBorder(BorderFactory.createLineBorder (Color.white, 1));
olioTN.vasenKuvapaneeli.setPreferredSize(Dimension(150, 0));
olioTN.vasenKuvapaneeli.setBorder(BorderFactory.createLineBorder (Color.white, 1));

olioTN.keskipaneeli.setPreferredSize(Dimension(olioTN.paneelinLeveysPx, 0));
olioTN.keskipaneeli.setBorder(BorderFactory.createLineBorder (Color.white, 1));
olioTN.keskiKuvapaneeli.setPreferredSize(Dimension(150, 0));
olioTN.keskiKuvapaneeli.setBorder(BorderFactory.createLineBorder (Color.white, 1));

olioTN.oikeapaneeli.setPreferredSize(Dimension(olioTN.paneelinLeveysPx, 0));
olioTN.oikeapaneeli.setBorder(BorderFactory.createLineBorder (Color.white, 1));
```

```
  olioTN.oikeapaneeli.setBackground(Color.white);
  olioTN.oikeaKuvapaneeli.setPreferredSize(Dimension(150, 0));
  olioTN.oikeaKuvapaneeli.setBorder(BorderFactory.createLineBorder (Color.white, 1));

  % korjataan kuvien ja labeleiden asettelua
  olioTN.tunnistettavaKuva.setAlignmentX(Component.CENTER_ALIGNMENT);
  olioTN.ladattuKuvaLbl.setAlignmentX(Component.CENTER_ALIGNMENT);
  olioTN.tunnistusmalliKuva.setAlignmentX(Component.CENTER_ALIGNMENT);
  olioTN.malliLbl.setAlignmentX(Component.CENTER_ALIGNMENT);
  olioTN.tulosKuva.setAlignmentX(Component.CENTER_ALIGNMENT);
  olioTN.tulosLbl.setAlignmentX(Component.CENTER_ALIGNMENT);
  % kuvien asetuksia EI TOIMI!!
  olioTN.tunnistettavaKuva.resize(olioTN.kuvanLeveysPx, olioTN.kuvanKorkeusPx);
  olioTN.tunnistusmalliKuva.resize(olioTN.kuvanLeveysPx, olioTN.kuvanKorkeusPx);
  olioTN.tulosKuva.resize(olioTN.kuvanLeveysPx, olioTN.kuvanKorkeusPx);

  % käyttöliittymäikkunan asetuksia
  olioTN.tunnistuspaneeli.setSize(olioTN.ikkunanLeveysPx, olioTN.ikkunanKorkeusPx)
end
% komponenttien päivittämistä****************************************
function vaihdaKuva(self, uusiKuva)
    % tarkastetaan kuvan koko
    kuva = self.tarkastaKuvanKoko(uusiKuva);
    % asetetaan kuva ImageIconiin
    self.tunnistettavaKuva.setIcon(javax.swing.ImageIcon(kuva));
end
function vaihdaMalli(self, uusiMalli)
    self.tunnistusmalliKuva.setIcon(javax.swing.ImageIcon(uusiMalli));
end
function vaihdaTulos(self, uusiTulos)
    self.tulosKuva.setIcon(javax.swing.ImageIcon(uusiTulos));
    haettuKuva = self.tulosKuva.getIcon();
    disp('vaihdaTulos,tunnistusnakyma');disp(haettuKuva);
end
function muutettuKuva = tarkastaKuvanKoko(self,kuva)
    % muutetaan Javan String MATLABiin kelpaavaksi char-muuttujaksi
    muutettuKuva = imread(char(kuva));
    % luetaan kuvan tiedot
    info = imfinfo(char(kuva));
    % pienennetään kuvaa, mikäli se on liian korkea tai leveä eli yli
    % 500 pikseliä
    if info.Height > 500
        muutettuKuva  = imresize(muutettuKuva, [500 NaN]);
    end;
```

```
            if info.Width > 500
                muutettuKuva  = imresize(muutettuKuva, [NaN 500]);
            end;
            % muutetaan MATLABin kuva Javan kuvaksi
            muutettuKuva = im2java(muutettuKuva);
        end
        % ulkoasu AbstraktiNakyma-luokasta**********************************
        function setUlkoasu(self)
            self.setUlkoasuPaneeli();
            self.setUlkoasuKuvaPaneeli();
            self.setUlkoasuOtsikkoFontti();
        end
        function setUlkoasuPaneeli(self)
            setUlkoasuPaneeli@AbstraktiNakyma(self.vasenpaneeli);
            setUlkoasuPaneeli@AbstraktiNakyma(self.keskipaneeli);
            setUlkoasuPaneeli@AbstraktiNakyma(self.oikeapaneeli);
        end
        function setUlkoasuKuvaPaneeli(self)
            setUlkoasuKuvaPaneeli@AbstraktiNakyma(self.vasenKuvapaneeli);
            setUlkoasuKuvaPaneeli@AbstraktiNakyma(self.keskiKuvapaneeli);
            setUlkoasuKuvaPaneeli@AbstraktiNakyma(self.oikeaKuvapaneeli);
        end
        function setUlkoasuOtsikkoFontti(self)
            setUlkoasuOtsikkoFontti@AbstraktiNakyma(self.ladattuKuvaLbl);
            setUlkoasuOtsikkoFontti@AbstraktiNakyma(self.malliLbl);
            setUlkoasuOtsikkoFontti@AbstraktiNakyma(self.tulosLbl);
        end
    end
end
```

KOIRAKO: Luokka KOIRAKOKontrolleri

Kontrollerimoduulin ainoa luokka KOIRAKOKontrolleri on yhteydessä käynnistysmoduulin KaynnistaSovellus luokkaan. Lisäksi KOIRAKOKontrolleri kommunikoi käyttöliittymämoduulin KOIRAKOKayttoliittyma-luokan ja sovelluslogiikkamoduulin KOIRAKOSovelluslogiikka-luokan kanssa.



Kuva 66. Luokan KOIRAKOKontrolleri muuttujat ja metodit.

KOIRAKOKontrolleri-luokan muuttujat ja metodit ovat havainnollistettuna kuvaan 66, jonka jälkeen esitellään luokan ohjelmakoodi kokonaisuudessaan.

```
classdef KOIRAKOKontrolleri
    properties (Access = private)
        kayttoliittyma;
        sovelluslogiikka;
    end
    methods
        % konstruktori*****************************************************
        function olioKntr = KOIRAKOKontrolleri(kayttoliittyma, sovelluslogiikka)
            olioKntr.kayttoliittyma   = kayttoliittyma;
            olioKntr.sovelluslogiikka = sovelluslogiikka;
        end
        % callback-toiminnot***********************************************
        function haeKuvaBtnClick(self,data,btn)
            % Käyttöliittymä: avaa tiedoston eli kuvan valintanäytön
            self.kayttoliittyma.avaaTiedostonValitsija();
            % Käyttöliittymä: tallenna kuvan polku tunnistusnakymään
            self.kayttoliittyma.tallennaValittuKuvaPolkuTunnistusNakymaan();
            % Käyttöliittymä: hae kuva polku muuttujaan
            valittuKuvaPolku = self.kayttoliittyma.getTunnistusnakymaValittuKuvaPolku();
            % Sovelluslogiikka: tallenna kuva TunnistettavaKuva-luokan muuttujaan
            self.sovelluslogiikka.tallennaValittuKuvaPolku(valittuKuvaPolku);
```

```
            % Käyttöliittymä: vaihda kuva käyttöliittymään
            self.kayttoliittyma.vaihdaTunnistettavaKuva(valittuKuvaPolku);
        end
        function valitseTunnistusmalli(self,data,btn)
            % Käyttöliittymä: hae mallin nimi
            valittuMalliIndx = self.kayttoliittyma.haeValittuMalliIndx();
            disp(valittuMalliIndx);
            % Sovelluslogiikka: tallenna ja hae mallin tiedot
            mallinkuvaPolku = self.sovelluslogiikka.haeTunnistusmallinTiedot(valittuMalliIndx);
            % Käyttöliittymä: vaihda mallin kuva käyttöliittymään
            self.kayttoliittyma.vaihdaTunnistusmalliKuva(mallinkuvaPolku);
        end
        function etsiKoiraBtnClick(self,data,btn)
            disp('Kontrollerin metodi: etsiKoiraBtnClick');
            % Sovellulogiikka: tunnista koira kuvasta
            tulosKuvaPolku = self.sovelluslogiikka.tunnistaKuvastaKoira();
            % Käyttöliittymä: vaihda tuloskuva käyttöliittymään
            self.kayttoliittyma.vaihdaTuloskuva(tulosKuvaPolku);
            disp('Tunnistusnakymasta haettu:');
            disp(self.kayttoliittyma.getTunnistusnakymaTulosKuva());
        end
        function tallennaTuloksetBtnClick(self,data,btn)
            % Sovelluslogiikka: hae tuloskuvan polku
            tulosKuva = self.sovelluslogiikka.getTunnistamoTunnistustulosImg;
            % Käyttöliittymä: tallenna tunnistustulos
            self.kayttoliittyma.tallennaTulosTiedostoon(tulosKuva);
        end
        function suljeItmClick(self,data,btn)
            % Käyttöliittymä: sulje JFrame

self.kayttoliittyma.getKaliIkkuna().setDefaultCloseOperation(self.kayttoliittyma.getKaliIkkuna().EXIT_ON_CLOSE);
        end
        function tietojaItmClick(self,data,btn)
            % Kööyttöliittymä: avaa tietoja-ikkuna
            self.kayttoliittyma.getTietojaRaami().setVisible(true);
        end
        function suljeTietojaBtnClick(self,data,btn)
            % Kööyttöliittymä: sulje tietoja-ikkuna
            self.kayttoliittyma.getTietojaRaami().setVisible(false);
        end
    end
end
```

KOIRAKO: Luokka KOIRAKOSovelluslogiikka

KOIRAKOSovelluslogiikka-luokka kuuluu sovelluslogiikkamoduuliin, johon kontrolleri-moduulin KOIRAKOKontrolleri-luokka on yhteydessä, lisäksi KOIRAKOSovelluslogiikka-luokka hallinnoi tunnistusmoduulin luokkien toimintaa Tunnistamo-luokan välityksellä.

Kuva 67. KOIRAKOSovelluslogiikka luokan muuttujat ja metodit.

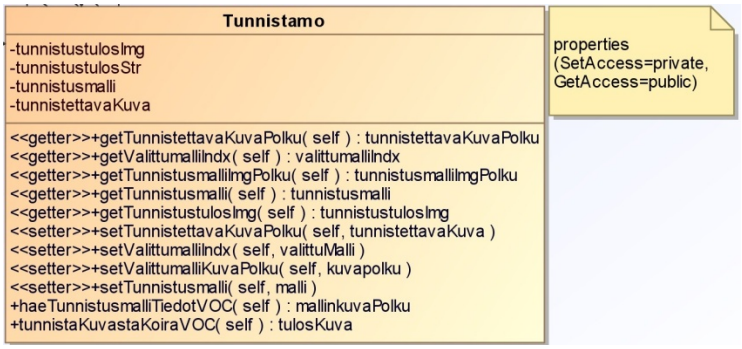
Luokan KOIRAKOSovelluslogiikka muuttujat ja metodit ovat näkyvissä kuvassa 67, jonka jälkeen esitellään luokan ohjelmakoodi.

```
classdef KOIRAKOSovelluslogiikka
    properties (Constant)
        VOCPolku = '/home/tpj/prog/MATLAB/R2010b/work/voc-release3.1';
    end
    properties (SetAccess = private, GetAccess = public)
        tunnistamo;
    end
    methods
      % konstruktori************************************************************
      function olioSov = KOIRAKOSovelluslogiikka()
          olioSov.tunnistamo = Tunnistamo();
          addpath(olioSov.VOCPolku);
      end
      % getterit****************************************************************
      function getTunnistamoTunnistustulosImg = getTunnistamoTunnistustulosImg(self)
          getTunnistamoTunnistustulosImg = self.tunnistamo.get.tunnistustulosImg;
      end
      % muut metodit************************************************************
      function tallennaValittuKuvaPolku(self,valittuKuvaPolku)
          self.tunnistamo.setTunnistettavaKuvaPolku(valittuKuvaPolku);

          testi = self.tunnistamo.getTunnistettavaKuvaPolku();
          disp('KOIRAKOSovelluslogiikka: '); disp(testi);
      end
      function mallinkuvaPolku = haeTunnistusmallinTiedot(self, valittuMalli)
```

```
            % tallennetaan valitun mallin indxk Tunnistusmalli-luokkaan
            self.tunnistamo.setValittumalliIndx(valittuMalli);
            % haetaan VOCista tiedot tunnistusmallista
            mallinkuvaPolku = self.tunnistamo.haeTunnistusmalliTiedotVOC();
        end
        function tulosKuva = tunnistaKuvastaKoira(self)
            tulosKuva = self.tunnistamo.tunnistaKuvastaKoiraVOC();
        end
    end
end
```

KOIRAKO: Luokka Tunnistamo

Tunnistusmoduulin keskeinen luokka on Tunnistamo, jonka välityksellä KOIRAKO-sovellus kutsuu VOC-sovelluksen funktioita. Lisäksi Tunnistamo tallentaa VOC-sovelluksesta kutsuttuja tietoja tunnistusmoduulin Tunnistusmalli ja TunnistettavaKuva luokkiin. KOIRAKOKontrolleri-luokka ohjailee KOIRAKOSovelluslogiikan välityksellä Tunnistamo-luokan toimintoja.



Kuva 68. Tunnistamo luokan muuttujat ja metodit.

Tunnistusmoduuliin kuuluvan luokan Tunnistamon muuttujat ja metodit ovat nähtävissä kuvassa 68, jonka jälkeen esitetään Tunnistamon ohjelmaakoodi kokonaisuudessaan.

```
classdef Tunnistamo < hgsetget
   properties (SetAccess = private, GetAccess = public)
       % tunnistustulokseen liittyvät
       tunnistustulosImg;
       tunnistustulosStr;
       % tunnistusmalliin liittyvät
       tunnistusmalli;
       % tunnistettavaan kuvaan liittyvät
       tunnistettavaKuva;
    end
    methods
      % konstruktori
      function olioSov = Tunnistamo()
          olioSov.tunnistusmalli     = Tunnistusmalli();
          olioSov.tunnistettavaKuva = TunnistettavaKuva();
      end
      % getterit
      function getTunnistettavaKuvaPolku = getTunnistettavaKuvaPolku(self)
```

```
    getTunnistettavaKuvaPolku = self.tunnistettavaKuva.get.tunnistettavaKuvaPolku();
end
function getValittumalliIndx = getValittumalliIndx(self)
    getValittumalliIndx = self.tunnistusmalli.get.valittumalliIndx();
end
function getTunnistusmalliImgPolku = getTunnistusmalliImgPolku(self)
    getTunnistusmalliImgPolku = self.tunnistusmalli.get.tunnistusmalliImgPolku();
end
function getTunnistusmalli = getTunnistusmalli(self)
    getTunnistusmalli = self.tunnistusmalli.get.tunnistusmalli();
end
function getTunnistustulosImg = getTunnistustulosImg(self)
    getTunnistustulosImg = self.tunnistustulosImg;
end
 % setterit
function setTunnistettavaKuvaPolku(self,tunnistettavaKuva)
    self.tunnistettavaKuva.setTunnistettavaKuvaPolku(tunnistettavaKuva);
end
function setValittumalliIndx(self,valittuMalli)
    self.tunnistusmalli.setValittumalliIndx(valittuMalli);
end
function setValittumalliKuvaPolku(self, kuvapolku)
    self.tunnistusmalli.setTunnistusmalliImgPolku(kuvapolku);
end
function setTunnistusmalli(self, malli)
    self.tunnistusmalli.setTunnistusmalli(malli);
end

% VOCin metodit
function mallinkuvaPolku = haeTunnistusmalliTiedotVOC(self)
    valittuMalliIndx = self.getValittumalliIndx();
    malliPolku       = '/home/tpj/prog/MATLAB/R2010b/work/voc-release3.1/mallit/';
    switch valittuMalliIndx
        case 0
            disp('valittu 0');
            malliImg = [malliPolku 'dog_final_2compMalli.jpg'];
            % ladataan malli VOCiin
            load /home/tpj/prog/MATLAB/R2010b/work/voc-release3.1/mallit/dog_final_2comp.mat;
            % tallenetaan mallin kuva tiedostoon,
            % model on VOCin muuttuja
            malli = figure, visualizemodel(model);
            saveas(malli, malliImg);
            close(malli);
            % pienennetään kuvaa ja tallennetaan se
```

```
                imwrite(imresize(imread(malliImg), 0.35),malliImg);
                % tallennetaan malli ja sen kuvan polku
                self.setTunnistusmalli(model);
                self.setValittumalliKuvaPolku('/home/tpj/prog/MATLAB/R2010b/work/voc-
release3.1/mallit/dog_final_2compMalli.jpg');

            case 1
                disp('valittu 1');
                malliImg = [malliPolku 'dog_final_4compMalli.jpg'];
                % ladataan malli VOCiin
                load /home/tpj/prog/MATLAB/R2010b/work/voc-release3.1/mallit/dog_final_4comp.mat;
                % tallenetaan mallin kuva tiedostoon,
                % model on VOCin muuttuja
                malli = figure, visualizemodel(model);
                saveas(malli, malliImg);
                close(malli);
                % pienennetään kuvaa ja tallennetaan se
                imwrite(imresize(imread(malliImg), 0.35),malliImg);
                % tallennetaan malli ja sen kuvan polku
                self.setTunnistusmalli(model);
                self.setValittumalliKuvaPolku('/home/tpj/prog/MATLAB/R2010b/work/voc-
release3.1/mallit/dog_final_4compMalli.jpg');
            case 2
                disp('valittu 2');
                malliImg = [malliPolku 'dog_final_6compMalli.jpg'];
                % ladataan malli VOCiin
                load /home/tpj/prog/MATLAB/R2010b/work/voc-release3.1/mallit/dog_final_6comp.mat;
                % tallenetaan mallin kuva tiedostoon,
                % model on VOCin muuttuja
                malli = figure, visualizemodel(model);
                saveas(malli, malliImg);
                close(malli);
                % pienennetään kuvaa ja tallennetaan se
                imwrite(imresize(imread(malliImg), 0.35),malliImg);
                % tallennetaan malli ja sen kuvan polku
                self.setTunnistusmalli(model);
                self.setValittumalliKuvaPolku('/home/tpj/prog/MATLAB/R2010b/work/voc-
release3.1/mallit/dog_final_6compMalli.jpg');
        end
        mallinkuvaPolku = self.getTunnistusmalliImgPolku();
    end
    function tulosKuva = tunnistaKuvastaKoiraVOC(self)
        tuloskuvanPolku = '/home/tpj/prog/MATLAB/R2010b/work/KOIRAKO/tulokset/';
        kuva = imread(char(self.getTunnistettavaKuvaPolku()));
```

```
        % tarkastetaan kuvan koko
        info = imfinfo(char(self.getTunnistettavaKuvaPolku()));
        if info.Height > 500
            kuva  = imresize(kuva, [500 NaN]);
        end;
        if info.Width > 500
            kuva  = imresize(kuva, [NaN 500]);
        end;

        tunnistuskehys = process(kuva, self.getTunnistusmalli());
        tulos          = figure, showboxes(kuva, tunnistuskehys);
        saveas(tulos, [tuloskuvanPolku 'tunnistustulos'], 'jpg');
%           close(tulos);
        imwrite(imresize(imread([tuloskuvanPolku 'tunnistustulos.jpg']), 0.40),[tuloskuvanPolku 'tunnistustulos.jpg']);
        self.tunnistustulosImg = [tuloskuvanPolku 'tunnistustulos.jpg'];
        tulosKuva = self.tunnistustulosImg;
        disp('tunnistamo tuloskuvapolku:'); disp(tulosKuva);
      end
    end
end
```

KOIRAKO: Luokka TunnistettavaKuva

TunnistettavaKuva on tunnistusmoduuliin kuuluva luokka ja Tunnistamo tallentaa ja hakee luokasta tietoa analysoitavasta kuvasta.

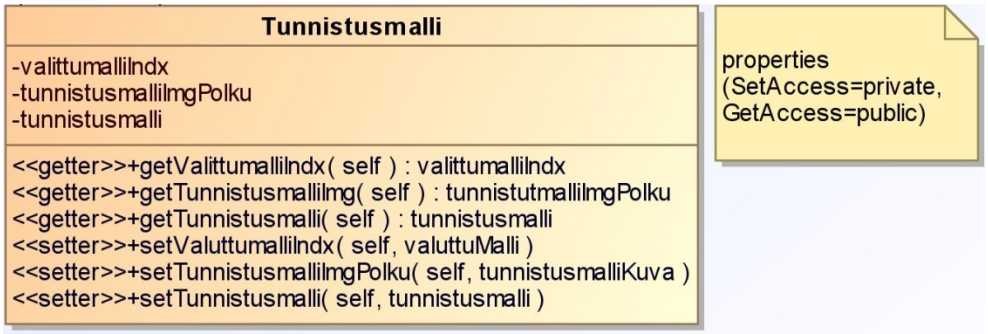
Kuva 69. Luokan TunnistettavaKuva muuttujat ja metodit.

Kuvassa 69 on näkyvissä luokan TunnistettavanKuva muuttujat ja metodit, joiden jälkeen esitellään ohjelmakoodi kokonaisuudessaan.

```
classdef TunnistettavaKuva < hgsetget
    properties (SetAccess = private, GetAccess = public)
        tunnistettavaKuvaPolku;
    end
    methods
        % getterit
        function getTunnistettavaKuvaPolku = get.tunnistettavaKuvaPolku(self)
          getTunnistettavaKuvaPolku = self.tunnistettavaKuvaPolku;
        end
        % setterit
        function self = setTunnistettavaKuvaPolku(self, tunnistettavaKuva)
            self.tunnistettavaKuvaPolku = tunnistettavaKuva;
        end
    end
end
```

KOIRAKO: Luokka Tunnistusmalli

Tunnistusmalli on tunnistusmoduuliin kuuluva luokka ja Tunnistamo tallentaa ja hakee luokasta tietoa tunnistusmallista.



Kuva 70. Luokan Tunnistusmalli muuttujat ja metodit.

Kuvassa 70 on näkyvissä luokan Tunnistusmalli muuttujat ja metodit, joiden jälkeen esitellään ohjelmakoodi kokonaisuudessaan.

```
classdef Tunnistusmalli < hgsetget
    properties (SetAccess = private, GetAccess = public)
        valittumalliIndx;
        tunnistusmalliImgPolku;
        tunnistusmalli;
    end
    methods
        % getterit*****************************************************
        function getValittumalliIndx = get.valittumalliIndx(self)
          getValittumalliIndx = self.valittumalliIndx;
        end
        function getTunnistusmalliImg = get.tunnistusmalliImgPolku(self)
          getTunnistusmalliImg = self.tunnistusmalliImgPolku;
        end
        function getTunnistusmalli = get.tunnistusmalli(self)
          getTunnistusmalli = self.tunnistusmalli;
        end
        % setterit*****************************************************
        function self = setValittumalliIndx(self, valittuMalli)
            self.valittumalliIndx = valittuMalli;
        end
        function self = setTunnistusmalliImgPolku(self, tunnistusmalliKuva)
```

```
            self.tunnistusmalliImgPolku = tunnistusmalliKuva;
        end
        function self = setTunnistusmalli(self, tunnistusmalli)
            self.tunnistusmalli = tunnistusmalli;
        end
    end
end
```

KOIRAKO: Luokka KaynnistaSovellus

```
classdef KaynnistaSovellus
    properties (Access = private)
        kayttoliittyma;
        sovelluslogiikka;
        kontrolleri;
    end
    methods
        % konstruktori*****************************************************
        function olioKaSov = KaynnistaSovellus()
            olioKaSov.kayttoliittyma    = KOIRAKOKayttoliittyma();
            olioKaSov.sovelluslogiikka = KOIRAKOSovelluslogiikka();
            olioKaSov.kontrolleri       = KOIRAKOKontrolleri(olioKaSov.kayttoliittyma, olioKaSov.sovelluslogiikka);
            olioKaSov.kayttoliittyma.asetaKontrolleri(olioKaSov.kontrolleri);
        end
    end
end
```