Osaamista
ja oivallusta
tulevaisuuden
tekemiseen

Paavo Urpo

Kohteentunnistus ja etäisyydenmittaus toteutettuna konenäöllä

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Sähkö- ja Automaatiotekniikka

Insinöörityö

8.11.2021

Tekijä Otsikko Sivumäärä Aika	Paavo Urpo Kohteentunnistus ja etäisyydenmittaus toteutettuna konenäöllä 26 sivua 8.11.2021
Tutkinto	insinööri (AMK)
Tutkinto-ohjelma	Sähkö- ja automaatiotekniikan tutkinto-ohjelma
Ammatillinen pääaine	Automaatiotekniikka
Ohjaajat	Yliopettaja Erkki Räsänen

Insinöörityön aiheena oli konenäkö. Työn tavoitteena oli toteuttaa sovellus, jolla voitaisiin tunnistaa määriteltyjä kappaleita kuvista ja laskea syvyys määriteltyyn pisteeseen.

Työ toteutettiin Windows-PC:llä käyttäen Python-ohjelmointikieltä sekä OpenCV -konenäkökirjastoa. Kameroina toimi pari keskenään samanlaisia web-kameroita.

Työ alkoi teoreettisen materiaalin läpikäynnillä, josta selvisi erilaisia menetelmiä kappaleen tunnistukseen ja kolmiulotteiseen havainnointiin sekä laskukaavoja. Seuraavaksi perehdyttiin erilaisiin menetelmiin toteuttaa syvyyslaskentaa Python-ohjelmointikielellä sekä OpenCV -konenäkökirjastolla.

Työn aikana rakennettiin omia haar-määrittelijöitä, jotka toimivat melko tarkasti, ympäristöriippuvaisesti. Stereokuvasta laskemalla saatu syvyys kohteeseen toimi myös.

Tuloksista voitiin johtaa laskennan toimivuus. Myös haar-määrittelijän rakentaminen osoittautui toteutettavaksi käytettävissä olevan materiaalin ja laitteiden avulla.

Tulokset ovat hyödynnettävissä etäisyyden mittaamiseen stereokameroilla, haar-luokittelijan rakentamisessa sekä tallennettaessa etäisyyksiä havainnoitavasta ympäristöstä.

Avainsanat	Konenäkö, Ohjelmointi, Python, OpenCV

Author Title Number of Pages Date	Paavo Urpo Object Detection and Measuring Distance using Computer Vision 26 pages 8 November 2021
Degree	Bachelor of Engineering
Degree Programme	Degree Programme in Electrical and Automation Engineering
Professional Major	Automation Engineering
Instructors	Erkki Räsänen, Principal Lecturer

The subject of the thesis is computer vision. The aim of the study was to create a solution which could detect objects and calculate distance to a specified point in the image.

The solution was created on a Windows-PC using Python programming language and OpenCV computer vision library. Two similar webcams were used as cameras.

Working with the thesis project started with going through theoretical material, which included different ways of object detection and depth calculations. After that, different ways of using the theory with Python programming language and OpenCV computer vision library were explored.

During the thesis work, self-made haarcascade classifiers were built up. They were accurate enough, depending on the environment. Also the depth from stereo calculations were accurate.

Based on results, it can be concluded that calculating depth from stereo is accurate, and a cascade classifier can be built with the material and the equipment available.

The results can be used if it is needed to calculate depth from stereo, build a haarcascade classifier or save distances to a memory from the inspected environment.

Keywords	Computer Vision, Programming, Python, OpenCV

Sisällys

1 Johdanto

Opinnäytteenä toteutetaan konenäköohjelma. Vaatimuksena ohjelmalla on pystyä erottelemaan tietokantaansa määritellyt esineet tai asiat sekä pystyä tiettyyn ympäristöön pohjautuvaan kappaleiden paikanmääritykseen. Opinnäytetyön tavoite on tuo määriteltyjen asioiden tunnistaminen, koska kun ohjelma valmistuu, on se käyttökelpoinen hyvin monessa tarpeessa. Sitä voidaan hyödyntää esimerkiksi liikenteessä, lajittelussa tai robotissa.

Käyttökohteita on paljon. Opinnäytteen tekijä voi myös kehittää ohjelman avulla omia laitteitaan ja jakaa ideansa ja osaamisensa muille.

Opinnäytteen aihe on valittu niin, että se voidaan toteuttaa kotona. Toisaalta, jos oppilaitoksen laboratoriossa on jotakin todella tehokkaasti hyödynnettävää laitteistoa, sitä tullaan käyttämään mahdollisuuksien mukaan. Työ tehdään oppilaitokselle. Työ on myös selvitystyö, koska osaaminen ja taidot ovat aloittelijan tasolla.

Teoriaosuudessa esitellään erilaisia tapoja analysoida ja käsitellä kuvia halutun toiminnan toteuttamiseksi. Tällä tarkoitetaan esimerkiksi muotoja ja värivaihteluita, joiden avulla määritellään kuvista tunnistettavia asioita. Lisäksi sisältyy kuvaus ja selostus ohjelmasta ja sen toiminnasta. Ongelmat ja löydökset tullaan myös kuvailemaan.

Aiheeksi tämä valikoitui, koska kotonakin voi ohjelmoida vaativiakin sovelluksia omaaloitteisesti, konenäkö on mielenkiintoista sen monien käyttömahdollisuuksien takia, joita ovat esimerkiksi liikenne, valvonta tai erilaiset lajittelutehtävät. Ohjelmointi on myös oleellinen taito automaatioalalla, ja sen kehittäminen on hyödyllistä, teki sillä sitten mitä tahansa. Tavoitteisiin päästään ajanmukaisen lähdekirjallisuuden sekä dokumentaation noudattamisella.

2 Toiminta konenäön taustalla

Tässä luvussa kerrotaan konenäön toiminnan taustalle tarvittavasta kuvan käsittelystä, sekä erilaisista menetelmistä ja teoriasta, jonka avulla konenäkö etsii kuvasta syvyysulottuvuuden, eli kolmiuloitteisen havainnon tekemisen ja miten tapahtuu kohteentunnistus.

2.1 Kuvaprosessointi

Jotta konenäkö toimii halutulla tavalla, tarvitsee kuvaa käsitellä ja nostaa siitä esiin piirteitä tai merkata siitä haluttuja kiintopisteitä, joiden varassa halutun kaltainen konenäkö toimii. Esimerkiksi värit, sävyt, rajat sekä toistuvat pintakuviot auttavat kohteen tai etäisyyden arvioinnissa.

Esimerkiksi kuvan digitalisoimisessa binääriseksi muutetaan värikuvasta harmaasävykuva, jonka jälkeen harmaasävykuvan pikseleiden arvo on väliltä 0-255. Tämän jälkeen kuvalle voidaan tehdä muunnos, johon asetetaan raja-arvo tiettyyn harmaasävyyn väliltä 0-255, jolla kuvasta saadaan binäärinen, eli kuvassa on vain mustia ja valkoisia pikseleitä. Tuo asetettu arvo siis määrittää, minkä harmaasävyn alle tai yli on pikseli joko musta tai valkoinen.

Kuva 1. Värikuva, josta on löydettävissä laatoitus, autoja, puita, liikennemerkki sekä talo. Näillä asioilla on piirteitä, joiden avulla voidaan määritellä joko asiat itse tai hahmottaa syvyyttä.

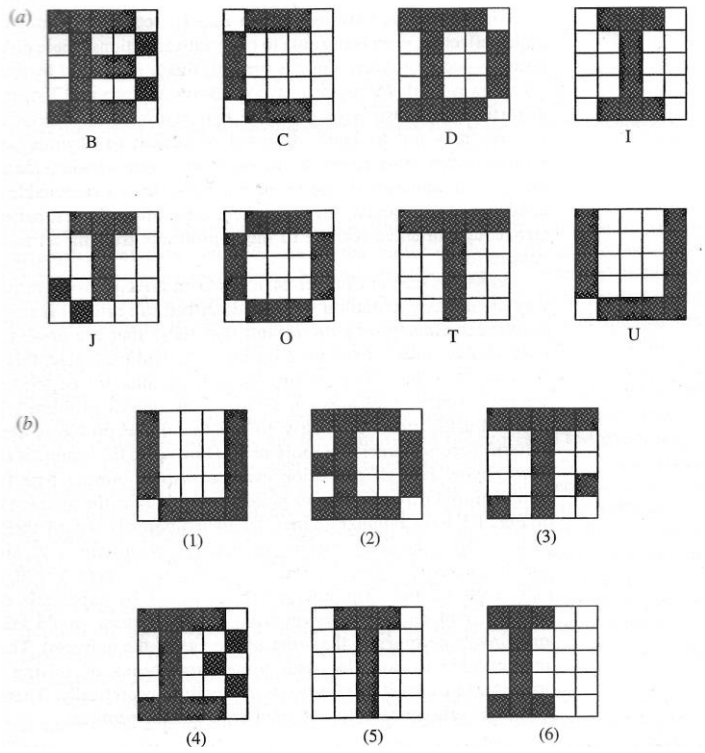
Kuva 2. Harmaasävykuva.

Kuva 3. Binäärikuva. Kuvassa on enää raja-arvon perusteella määritelty mustat ja valkoiset pisteet. Huomataan esimerkiksi laatoituksen pintakuvio. Yksittäisien laattojen reunat erottuvat mustana, jolloin kuvioinnin perusteella voidaan määritellä tarkastelusuuntaa

2.2 Halutun kohteen tunnistaminen bittikartasta

Kuvista tunnistettavien ominaisuuksien etsiminen perustuu algoritmeihin, jotka vertaavat sisään tulevaa tai valmiiksi tallennettua kuvadataa tietokannasta löytyviin sinne tallennettuihin piirteisiin, joita voivat olla erilaiset muodot tai linjat, tai jostakin samasta asiasta, esimerkiksi autosta kuvatut kuvat, joiden perusteella määritellään ja tunnistetaan haluttu asia kuvasta.

Tietokoneen muistiin voidaan tallentaa esimerkiksi 25-bittisiä mustavalkokuvia, joista jokainen on erilainen ja jotka on jaettu luokkiin. Tietokone siis opetetaan kuvilla, ja konenäön tunnistus vertaa vertailtavaa dataa tietokantaansa, josta löytyvät nämä kuvaesimerkit omiksi luokikseen määriteltynä.



Kuva 4. Kuvassa nähtävissä tietokannasta löytyvät opetetut esimerkit (*a*), sekä näihin verrattavat kuvat (*b*), joissa on myös epätäydellisiä kuvia verrattuna opetusesimerkkeihin (1, s. 3).

Kuvan esimerkeissä tunnistettavista (*b*) esimerkiksi numero 1 olisi selvästi U. Tämä tekniikka toimii myös, jos kuvissa esiintyy häiriötä. Häiriön lisääntyessä kuvia ei voida enää tunnistaa, ja vaikka kuvassa ei olisi häiriötä, tunnistus voi mennä pieleen, mikäli esimerkkikuva on väärin sijoitettu tai väärässä asennossa (1, s. 3-4). Edellä mainittu häiriötilanne väärää asentoa koskien kuvaa tätä teoreettista esimerkkiä, tietokoneohjelmassa on joko digitaalisesti käsiteltynä tai muuten syötettynä esimerkkikuvan kaltaiset kuvasta haettavien piirteiden eri käännöt ja asennot määriteltynä, jotta tarkastelupisteen tai kohteen asento ei vaikuttaisi tunnistukseen, esimerkiksi auton piirteet monesta suunnasta.

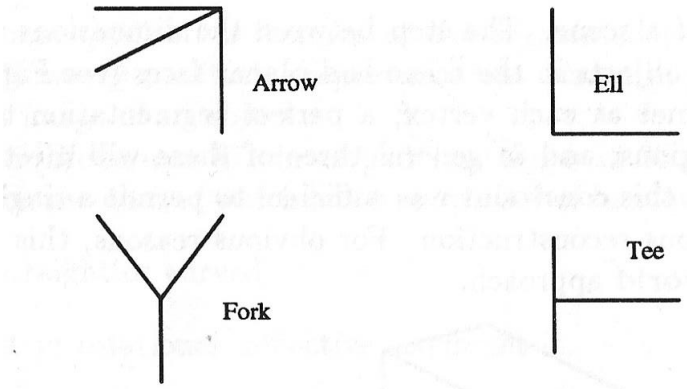
2.3 Kolmiulotteinen havaitseminen

Kolmiulotteiseen hahmottamiseen konenäöllä on olemassa aktiivisia ja passiivisia tekniikoita. Aktiivitekniikassa alueelta, joka halutaan määritellä, heijastetaan takaisin sähkömagneettista säteilyä, esimerkiksi valoa, ja etäisyydet saadaan johtamalla heijastuksen

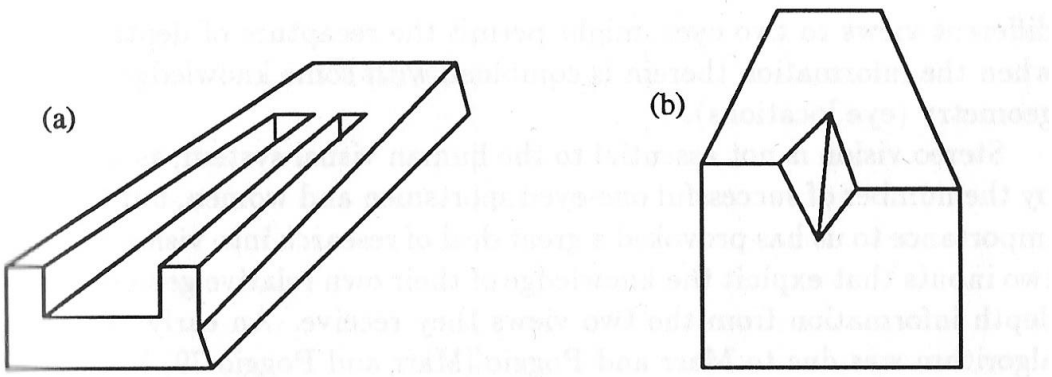
palaamiseen kuluvasta ajasta tai vaihe-erosta. Myös valokeilan muodolla voidaan määritellä pinnan asentoa tai muotoa johdettuna heijastuksen muodosta.

Passiivitekniikoissa hyödynnetään kuvasta löytyviä piirteitä sekä tarkastelupisteen eli kameran liikettä tai kahta tarkastelupistettä, eli stereokuvaa. Kuvasta voidaan etsiä esimerkiksi pystyssä olevia linjoja tai muita selkeästi määriteltäviä pisteitä, joiden siirtymää verrataan tarkastelupisteiden etäisyyteen toisistaan.

Passiivitunnistuslaite voi myös etsiä ennalta määritetyn muotoisia ja kokoisia kappaleita, tai hyödyntää toistuvia pintakuviointeja sekä valaistusta, jonka avulla hahmotusta johdetaan heijastuksesta tai varjostuksesta. (1, s. 446-448; 2, s. 413-415.)



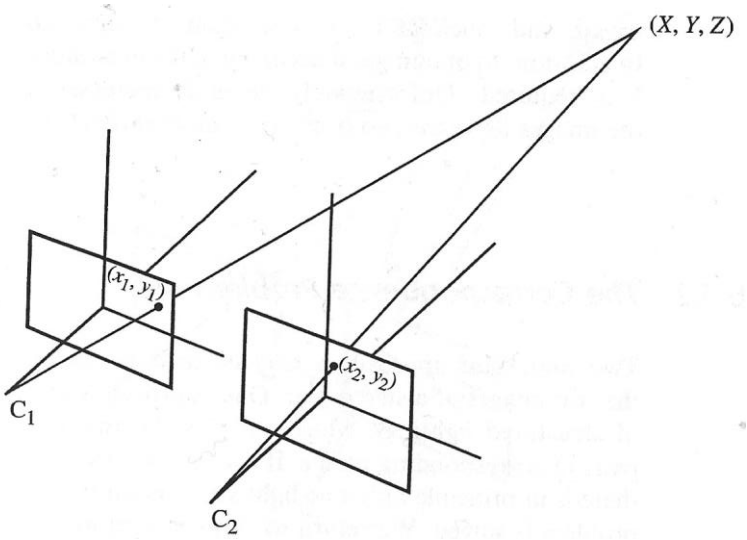
Kuva 5. 4 erilaista vaihtoehtoa tunnistaa kulma, kun oletetaan jokaisella kappaleella olevan kolme sivua, joiden reunat yhdistyvät yhdessä pisteessä (2, s. 384).



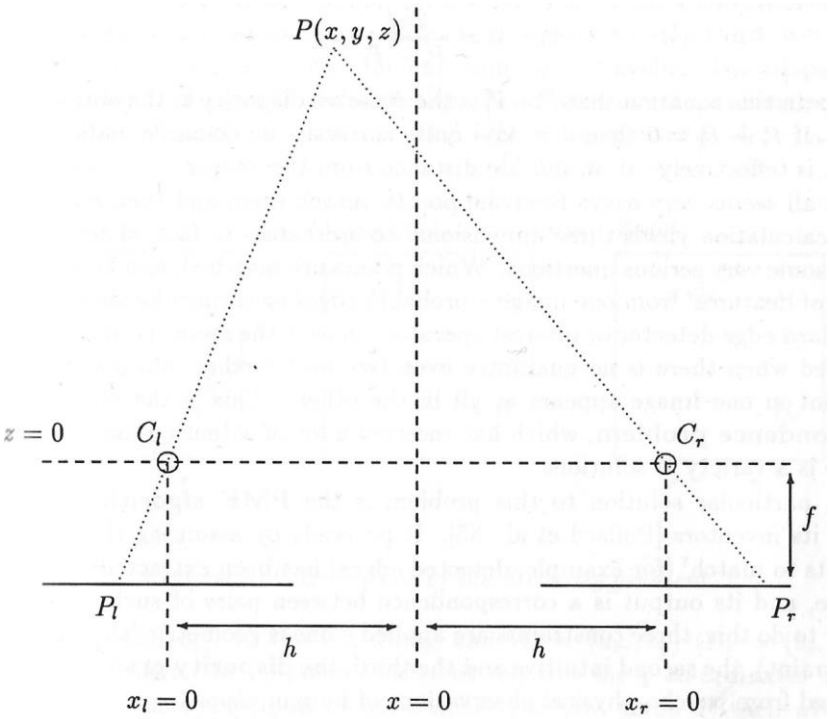
Kuva 6. Kaksi esimerkkiä mahdottomasta reunojen, sivujen ja kulmien avulla määriteltävästä kappaleesta (2, s. 385).

2.3.1 Syvyyskartta stereokuvasta

Syvyyskartan luomiseen stereokuvasta tarvitaan kuva kahdelta kameralta, jotka ovat yhdensuuntaiset, kameroiden etäisyys toisistaan sekä polttoväli (1, s. 451). Ohjelma etsii samat pisteet kahdesta kuvasta, joiden siirtymän perusteella se laskee etäisyyden.



Kuva 7. Kuvassa kolmiulotteinen esimerkki kahden tarkastelupisteen asetelmasta ja niiden tuottamista kuvista. C_1 on vasen tarkastelupistettä ja C_2 on oikea tarkastelupiste. x_1 ja x_2 puolestaan ovat vasemman ja oikean tarkastelupisteen tuottamat koordinaatit pisteelle, jonka syvyys halutaan laskea (1, s. 451).



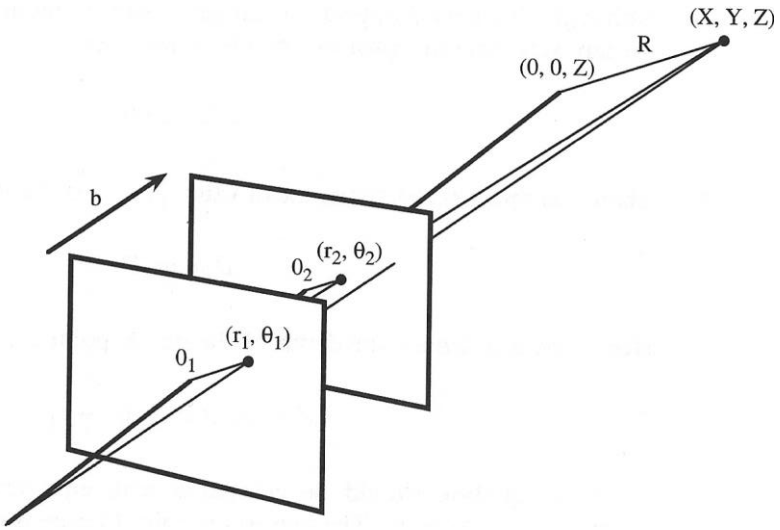
Kuva 8. Kuva esittää kahta tarkastelupistettä ja näiden suhdetta pisteeseen, jonka syvyys halutaan laskea (2, s. 387).

$$z = \frac{2hf}{(x_{l-}x_{r)}} \quad (1)$$

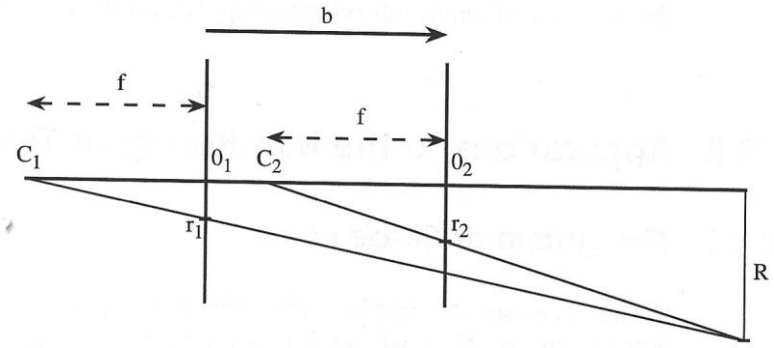
jossa z on syvyys, h on kameran etäisyys kantalinjan keskeltä, f on polttoväli, xl on vasemman kameran kuvasta löytyvän pisteen koordinaatti x-akselilla ja xr on oikean kameran kuvasta löytyvän pisteen koordinaatti x-akselilla.

2.3.2 Syvyys kameran liikkeestä

Syvyyden määritteleminen kameran liikkeestä perustuu samaan menetelmään kuin stereokuvasta. Tarvitaan yksi kamera, jonka ottamien kuvien muutoksien ja liikutun matkan ja suunnan perusteella voidaan laskea etäisyyksiä haluttuihin pisteisiin.



Kuva 9. Kolmiulotteinen esimerkki tilanteesta, joka on saatu aikaan tarkastelupisteen siirtämisellä (1, s. 517).



Kuva 10. Tarkastelupisteen liikkeellä saavutettavissa olevat mitat, joista on johdettavissa syvyys halutulle pisteelle (1, s. 517).

Laskukaavaa varten, jolla johdetaan syvyys tarkastelupisteen liikkeestä, tarvitaan polttoväli, r, r1 ja r2 sekä tarkastelupisteiden etäisyys b.

$$z = br/(r_2 - r_1) \quad (2)$$

jossa z on syvyys, b on kameran tarkastelupisteiden etäisyys toisistaan, r on $(\frac{1}{2}r_1 + r_2)$, r1 on tarkastelupiste 1 ja r2 on tarkastelupiste 2. (1, s. 516-518.)

3 Sovellus

Luvussa esitellään laitteisto ja ohjelmisto, perustelu laitteiston valinnalle sekä ohjelman toiminta.

3.1 Perustelu laitteiston valinnalle

Ohjelmat, eli Python ja OpenCV valikoituivat käytettäväksi saatavuutensa ja laajan internetistä löytyvän tiedon ja esimerkkien takia. Tarvitaan lisäksi vaatimukset täyttäviä rinnakkaisilla kameroilla otettuja stereokuvia sekä kameroiden polttoväli, sensorin koko ja kameroiden etäisyys toisistaan kuvaamishetkellä.

Opinnäytteen tekijä oli myös hieman perehtynyt OpenCV -kirjastolla ja Python-ohjelmointikielellä toteutettuun konenäköön, kun tarvittiin laitetta, joka ottaisi kuvia, kun kuvatulla alueella tapahtuu muutos, eli käytännössä liikettä.

Sovellus toimii millä tahansa tietokoneella, johon voidaan asentaa Python ja OpenCV ja jonka suoritusteho riittää kuvaprosessoinnin ja laskennan tekemiseen.

3.2 Ohjelma ja sen toiminta

Ohjelma hakee kahden kameran ottamista kuvista tunnistustavalla, jota kutsutaan haartunnistimeksi, englanninkielinen termi *haarcascade-classifier*, luokitellun tunnistettavan kohteen, ja laskee löytyneiden koordinaattien avulla aiemmin työssä esitellyn kaavan mukaan syvyyden stereokuvasta tunnistettuun pisteeseen.

Ohjelma tarkistaa, löytyvätkö x-, y- ja z-koordinaatit, sekä tallentaa ne ja paikkaa puuttuvat koordinaatit arvolla 1 ennen listaan tallennusta. Puuttuvat arvot on muutettava arvoksi 1, koska grafiikan piirtämiseen ohjelma tarvitsee kolmen koordinaatin sarjoja.

Ohjelma ottaa lukemaa ajalta, jolloin vain toisesta kuvasta saadaan tunnistus. Aika katkeaa, kun molemmista kuvista saadaan tunnistus. Tämän jälkeen tallennetaan listaan

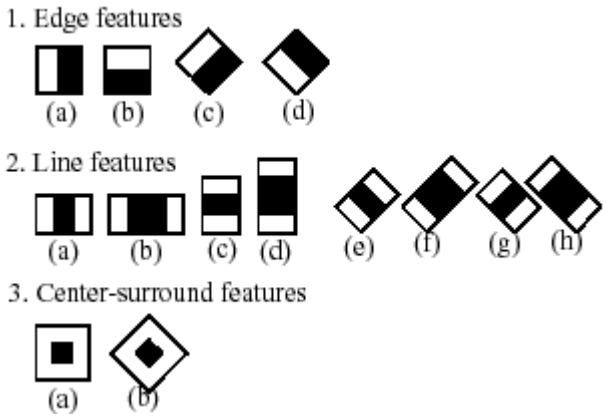
ohjelmakierrosten määrä, jolta havainto puuttuu. Vasemmalle ja oikealle kuvalle on molemmille omat listansa. Aiemmin mainitussa x- y- ja z-koordinaattien listaan tallennuksessa korjaaminen arvoksi 1 johtuu kuvaillusta tunnistuksen puuttumisesta.

Kun ohjelman kierto katkaistaan, ohjelma printtaa virheajan, jolloin tunnistus puuttui, ohjelmakiertojen määrän, jakaa virheajan ohjelmakiertojen määrällä, jotta saadaan osuus kierroksista, joilla tunnistusta ei tapahtunut sekä vasemman ja oikean virhelistat. Ohjelma myös piirtää grafiikan 3D-koordinaatistoon saaduista tunnistuksista sekä 2D-grafiikan etäisyydestä.

3.2.1 Haar-piirteisiin perustuva kappaleen tunnistus

Ohjelma käyttää kappaleentunnistukseen luokittelijaa, jonka englanninkielinen termi on *haarcascade-classifier*, joka etsii haar-piirteitä kuvasta. Etsittäessä kohdetta kuvasta käydään haar-luokittelijalla kuvan jokainen kohta läpi. Luokittelija voidaan myös pienentää tai suurentaa, jotta löydetään etsittävää kohdetta eri kokoisina.

Luokittelijan nimen sana *cascade* kuvaa luokittelijan rakennetta. Luokittelija koostuu muutamasta yksinkertaisemmasta luokittelijasta, vaiheista, (eng. *stages)*, joita käytetään toisensa jälkeen tarkasteltavaan kohtaan, jolloin jossakin vaiheessa luokittelija hylkää, eli päättää, että kohdetta ei löydy, taikka kohde löytyy, joka tarkoittaa, että alue on läpäissyt tunnistuksen kaikki vaiheet. (3.)



Kuva 11. Kuvassa on esitettynä tämänhetkisen algoritmin käyttämät haar-piirteet (3.).

Piirre, jota käytetään tietyssä luokittelijassa, määrittyy sen muodon, sijainnin ja koon perusteella. Nämä ovat suhteessa tarkkailtavaan alueeseen kuvasta, eivätkä suhteessa tarkasteltavaan kuvaan. Esimerkiksi kuvan 11. viivapiirteet (2. *Line features*) esimerkki (c), vaste on koko neliön alueen pikseleiden yhteenlasku, joka lasketaan vielä yhteen vain mustan alueen pikseleiden määrällä, joka kerrotaan kolmella. Kolmella kertomisella kompensoidaan alueiden kokoeroa. Yhteenlaskut tehdään nopeasti käyttämällä integraalikuvia. (3.)

Itse koodissa osa, joka etsii tunnistuksen, on alta löytyvä.

```
detectMultiScale(kuvatulo, scaleFactor, minNeighbors)
```

Jossa kuvatulo tarkoittaa kuvaa, josta tunnistusta etsitään, scaleFactor luokittelijan skaalaamista, esimerkiksi 1.05, joka tarkoittaa 5 %:n pienentämistä skaalaamisessa ja minNeighbors rinnakkaisten tunnistusten määrää, joka tunnistukseen tarvitaan. (5.)

3.2.2 Haar-luokittelijan rakentaminen

Luokittelija rakennetaan käyttämällä muutamaa sataa kuvaa kappaleesta, joka kuvasta halutaan tunnistaa. Näitä kuvia kutsutaan positiivisiksi näytteiksi. Ne muutetaan samaan kokoon, esimerkiksi 20x20 pikseliä. Luokittelija tarvitsee myös negatiivisia näytteitä, jotka ovat kaikkea muuta kuin tunnistettavaksi haluttua asiaa (3). Käsitykseksi on työtä tehdessä muodostunut, että näitä negatiivisia kuvia tulisi olla kymmenkertainen määrä verrattuna positiivisiin, mutta tietyssä ympäristössä toimiva luokittelija saatiin toimimaan pienemmällä määrällä. Myös positiivisien näytteiden määrä, joka on ollut kymmenesosa mainitusta suosituksesta, on toiminut tässä suppeassa ympäristössä.

Haar-luokittelijan rakentamiseen löytyi internetohje. (4.)

3.2.3 Toteutettu syvyyslaskenta

Syvyyslaskenta laskee etäisyyden tunnistettuun kohteeseen kahdesta kuvasta tehdyn tunnistuksen perusteella. Kun siis kuvat on otettu rinnakkaisilla kameroilla, näkyy tunnistettu kohde eri kohdalla kuvissa, joten tunnistuksen paikan x-koordinaattien erotuksella

saadaan laskettua etäisyys. Koko menetelmä on aiemmin teoriaosuudessa esitetty syvyys stereokuvasta.

Ohjelma muuttaa kaikki yksiköt pikselin leveyden fyysiseen mittaan ja tekee laskun käyttämällä niitä.

Aluksi määritetään pikselin leveys:

$$piksLev = \frac{\text{imgPlane}}{\text{lev\_oik}}$$

jossa imgPlane on sensorin leveys millimetreinä ja lev_oik on kuvan leveys pikseleinä.

seuraavaksi muunnokset pikselin leveyden yksiköksi polttovälin f ja tarkastelupisteiden etäisyyden B osalta:

$$polt\_piks = \frac{\text{f}}{\text{piksLev}}$$

$$bslne\_piks = \frac{\text{B}}{\text{piksLev}}$$

Joissa polt_piks ja bslne_piks saadaan pikselin leveyden yksiköinä.

Seuraavaksi tehdään lasku saaduilla yksiköillä ja kerrotaan pikselin leveydellä, jotta tulos saadaan millimetreinä:

$$Etäisyys\ mm = \left(\frac{\text{baselinepxl} * \text{polttovälipxl}}{x_{oikea} - x_{vasen}}\right) * pikselin\ leveys$$

3.2.4 Python-koodi

Luvussa ohjelman koodi on jaettu toiminnallisiin osiin. Toiminta ja koodin osan yhteydet kuvaillaan jokaisen osan jälkeen.

```
import cv2 as cv
import numpy as np
import matplotlib.pyplot as plt

kamera_oik = cv.VideoCapture(1)
kamera_vas = cv.VideoCapture(0)

kappale = cv.CascadeClassifier(
    './cascades/haarcascade_frontalcatface.xml')

frame_rate = 30

x_lists = []
y_lists = []
D_lists = []

virhe_lista_vas = []
virhe_lista_oik = []

yCoord = float
xCoord = float

B = 62.5
f = 4
imgPlane = 2.6

OhjelmaKierrot = 0
HavKat = 0
VirheAika = 0

VirheAikaväliVas = 0
VirheAikaväliOik = 0

Kuvajarjestys = 0

tunnistusAlku = bool

tunnistusAlku = False
```

Esimerkkikoodi 1. Ohjelman aluksi määritellään tarvittavat kirjastot, kamerat stereokuvaa varten, tunnistettavan kappaleen luokittelija-tiedosto, kuvataajuus, x-,y-, ja syvyyskoordinaattien ja virheiden tallentamiseen tarvittavat listat, x- ja y-koordinaattien muuttujatyypit, tarkastelupisteiden etäisyys toisistaan (B), kameran polttoväli ja sensorin koko, ohjelmakierrot, virheaika sekä vasen ja oikea virheaikaväli. Ohjelmakierrot, virheaika sekä vasemman ja oikean virheaikavälin muuttujat asetetaan alussa nollaksi.

```
def syvyyslaskenta(tunnistus_oik, tunnistus_vas, y_vas, y_oik):

    kork_oik, lev_oik, kanavat_oik = kuva_oik.shape
    piksLev =  ((imgPlane)/lev_oik)

    polt_piks = (f/piksLev)
    bslne_piks = (B/piksLev)

    x_oik = tunnistus_oik
    x_vas = tunnistus_vas

    ero = x_vas-x_oik
```

```
    if ero <= 0:
        ero = 1

    Oikea_Y = y_vas
    syvyys_mm = ((bslne_piks*polt_piks)/ero)*piksLev
    syvyys = syvyys_mm

    syvyys_pikseli = (bslne_piks*polt_piks)/ero
    Oikea_X_lasku = ((imgPlane/2)/f)*syvyys_mm*x_oik*piksLev
    Oikea_X = Oikea_X_lasku

    return syvyys, syvyys_pikseli, Oikea_X, Oikea_Y
```

Esimerkkikoodi 2. Syvyyslaskennan, eli etäisyyden laskemisen funktio. Funktioon syötetään 4 arvoa, jotka ovat x ja y koordinaatit vasemmasta ja oikeasta kuvasta. shape tuo arvot kuvan ulottuvuuksista, näistä lev_oik määrittelee kuvan leveyden pikseleinä. Ohjelma tekee syvyyslaskun, laskutoimitus on kuvattu kappaleessa 3.2.3. Toteutetaan lasku, jolla saadaan syvyys pikseleinä. Lasketaan Oikea_X, joka tarkoittaa X-koordinaatin oikeata etäisyyttä vasemman kameran kuvan vasemmasta reunasta, tämä on millimetreinä. Funktio palauttaa syvyyden, syvyyden pikseleinä, Oikea_X:n ja Oikea_Y:n.

```
def kappaleentunnistus(tunnistus, Kuvajarjestys):

    harmaa = cv.cvtColor(tunnistus, cv.COLOR_BGR2GRAY)
    kap_tun = kappale.detectMultiScale(harmaa, 1.05, 15)


    for (x, y, w, h) in kap_tun:
        kap_tun = cv.rectangle(tunnistus, (x, y), (x+w, y+h), (0, 0, 255), 2)

        tunnistus_alue = tunnistus[y:y+h, x:x+w]

        Kuvajarjestys = Kuvajarjestys + 1

        cv.imwrite("./tunnistustarkistus/k%s.jpg" % Kuvajarjestys, tunnis-
tus_alue)

        x = (x+(w/2))
        y = (y+(y/2))

        y_coord = y.astype('float64')
        x_coord = x.astype('float64')

        return x_coord, y_coord, Kuvajarjestys
```

Esimerkkikoodi 3. Kappaleentunnistus. Funktioon tuodaan kuva, joka muutetaan harmaakuvaksi ja josta etsitään kappaletta haar-luokittelijalla. Luodaan tunnistusalueelle koordinaateilla tunnistus_alue ja tallennetaan tuo alue kuvana tilastointia varten. Tehdään lasku, jolla saadaan koordinaatit tunnistusalueen keskelle. muutetaan y ja x arvo tyyppiin float64, sekä palautetaan arvot ohjelmalle. Funktio myös nostaa kuvajärjestyksen arvoa yhdellä kuvien tallennuksen takia.

```
while True:

    if tunnistusAlku == True:
```

```
        OhjelmaKierrot = OhjelmaKierrot + 1

    ret_oik, kuva_oik = kamera_oik.read()
    ret_vas, kuva_vas = kamera_vas.read()

    try:

        pikseli_oik, y_oik, Kuvajarjestys = kappaleentunnistus(kuva_oik, Kuva-
jarjestys)

    except TypeError:
        pikseli_oik = None

    try:

        pikseli_vas, y_vas, Kuvajarjestys = kappaleentunnistus(kuva_vas, Kuva-
jarjestys)

    except TypeError:
        pikseli_vas = None

    if pikseli_oik == None or pikseli_vas == None:
        if tunnistusAlku == True:
            HavKat = HavKat + 1

        if np.all(pikseli_oik) == True:
            VirheAikaväliVas = VirheAikaväliVas + 1
            VirheAika = VirheAika + 1
            print('oikea koordinaatti', pikseli_oik)
            tunnistusAlku = True

        if np.all(pikseli_vas) == True:
            VirheAikaväliOik = VirheAikaväliOik + 1
            VirheAika = VirheAika + 1
            print('vasen koordinaatti', pikseli_vas)
            tunnistusAlku = True
```

Esimerkkikoodi 4. Ohjelman ”Main loopin” alku. Lisätään ohjelmakiertolaskuriin 1 kierto, mikäli ensimmäinen tunnistus on saatu. Tämän jälkeen haetaan uudet kuvat kameroilta. Yritetään ”Try” hakea kappaleentunnistusfunktiolta koordinaatti tunnistuksesta. Jos tunnistusta ei löydy, määritetään koordinaattien arvot ”none”. Ohjelma tarkistaa or-ehdolla, onko vasen tai oikea pikseli ”none”. Mikäli toinen tunnistuksista löytyy, lisätään virheaika-arvoon 1, ja oikean tai vasemman virheaikaväliin 1. Vain oikeassa kuvassa löytyvä tunnistus aiheuttaa virheaika-arvon lisäyksen yhdellä, ja vasemman virheaikavälin arvon lisäyksen yhdellä. Virheaikaväli katkeaa, kun molemmista kuvista saadaan tunnistus, jolloin ohjelma siirtyy suoraan seuraavaan ”else” ehtoon. Vasemman ja oikean havainnon ehdot myös muuttavat molemmat tunnistusAlku-muuttujan todeksi, jotta ohjelma alkaa laskea ensimmäisestä tunnistuksesta asti.

```
    else:

        tunnistusAlku = True

        if VirheAikaväliVas > 0:
            virhe_lista_vas.append([VirheAikaväliVas])
        if VirheAikaväliOik > 0:
            virhe_lista_oik.append([VirheAikaväliOik])
```

```
        print('Virhe_Vas_aikaväli', VirheAikaväliVas)
        print('Virhe_Oik_aikaväli', VirheAikaväliOik)

        VirheAikaväliOik = 0
        VirheAikaväliVas = 0

        syvyys, syvyys_pikseli, Oikea_X, Oikea_Y = syvyyslaskenta(pikseli_oik,
pikseli_vas, y_oik, y_vas)

        cv.putText(kuva_oik, str(round(syvyys)), (20, 20), cv.FONT_HER-
SHEY_SIMPLEX, 1, (0, 255, 0), 3, cv.LINE_4)
        cv.putText(kuva_oik, str(round(Oikea_X)), (20, 60), cv.FONT_HER-
SHEY_SIMPLEX, 1, (0, 255, 0), 3, cv.LINE_4)
        cv.putText(kuva_oik, str(round(Oikea_Y)), (20, 100), cv.FONT_HER-
SHEY_SIMPLEX, 1, (0, 255, 0), 3, cv.LINE_4)


        try:
            print(syvyys)
        except ValueError:
            print('syvyys ei löydy')

        try:
            y_lists.append([Oikea_Y])
        except NameError:
            Oikea_Y = 1
            y_lists.append([Oikea_Y])
        try:
            x_lists.append([Oikea_X])
        except NameError:
            Oikea_X = 1
            x_lists.append([Oikea_X])
        try:
            D_lists.append([syvyys])
        except NameError:
            syvyys = 1
            D_lists.append([syvyys])
```

Esimerkkikoodi 5. Alussa muutetaan tunnistusalku todeksi, koska saadaan molemmilta kameroilta tunnistus. Ohjelma kirjaa molemmat virheaikavälit omiin listoihinsa ja printtaa molemmat virheaikavälit, muuttaa molemmat virheaikavälin arvot takaisin nollaan, koska virhe on katkennut molempien tunnistuksien löytyessä. Ohjelma hakee syvyyden, syvyyden pikselileveyden määränä, Oikea_X:n ja Oikea_Y:n syvyyslaskennasta. Ohjelma kirjoittaa oikean kameran kuvan päälle syvyyden millimetreinä, Oikea_X:n ja Oikea_Y:n. Ohjelma yrittää syvyyden printtausta. Ohjelma yrittää jokaisen listoihin kirjattavan koordinaatin syöttöä. Jos koordinaatti ei löydy, muutetaan sen arvoksi 1, jotta grafiikan piirrolla olisi jokainen täysi kolmen sarja XYZ-koordinaatistoon.

```
    cv.imshow("vasen kamera", kuva_vas)
    cv.imshow("oikea kamera", kuva_oik)

    if cv.waitKey(1) & 0xFF == ord('q') or OhjelmaKierrot == 500:

        print('VirheAika', HavKat)
        print('Ohjelmakierrokset', OhjelmaKierrot)
        try:
            print('virheen osuus', VirheAika/OhjelmaKierrot)
        except ZeroDivisionError:
```

```
    pass
print('oikea virhelista', virhe_lista_oik)
print('vasen virhelista', virhe_lista_vas)

fig = plt.figure(1)
ax = fig.add_subplot(projection='3d')

ax.scatter(x_lists, D_lists, y_lists, marker='o')

ax.set_xlabel('X Arvo')
ax.set_ylabel('Syvyys Millimetrejä')
ax.set_zlabel('Y Arvo')

ax.scatter([0,500], [0,1000], [0,500], marker='^')

plt.figure(2)
plt.ylabel('etäisyys')
plt.xlabel('havainnot järjestyksessä')
plt.grid(True)
plt.plot(D_lists)


plt.show()

break
```

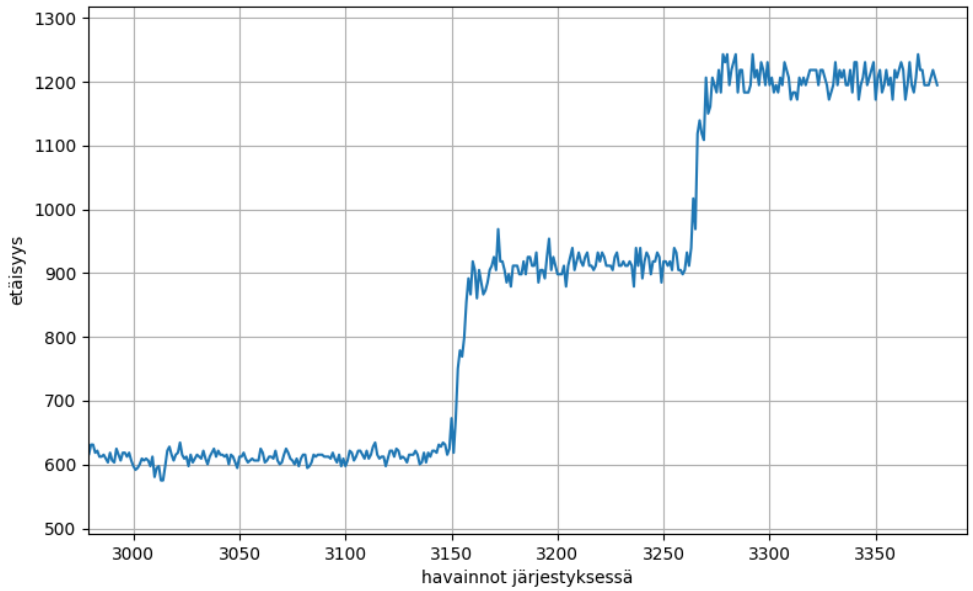
Esimerkkikoodi 6. Ohjelman ”Mainloopin” loppuosa. Näytetään molemmat kameran kuvat. Odotetaan lopetusehtoa, eli q-näppäimen painallusta tai 500 ohjelmakierron tosi-tilaa. Lopetusehdon täyttyessä ohjelma printtaa virheajan, ohjelmakierrosten määrän, yrittää printata virheen osuutta ohjelmakierroista, sekä oikean ja vasemman virhelistat . Ohjelma myös piirtää 3D-koordinaatistoon grafiikan tallennetuista havainnoista sekä havainnoinnin helpottamiseksi toiset pisteet, jotka vastaavat kiinteitä syvyyksiä ja x- sekä y-arvoja. Ohjelma piirtää myös 2D-grafiikan syvyydestä. Lopuksi ”break” katkaisee loopin ja ohjelma loppuu.

4 Tulokset

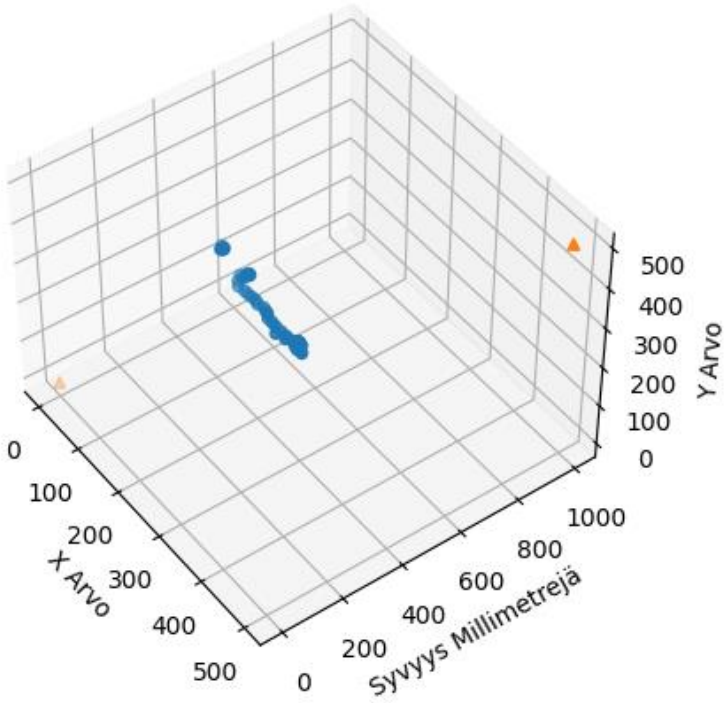
Kappaleessa kuvataan syvyystunnistuksen tarkkuutta sekä kohteentunnistuksen tarkkuus. Otan OpenCV -esimerkeistä löytyvän kasvotunnistuksen ja kissan kasvot vertailukohteeksi, kun verrataan tarkkuutta itse rakennettuihin tunnistimiin.

Etäisyyden laskemisen tarkkuus on tuloksieni mukaan ±2,5% mitatusta etäisyydestä. Mielestäni tämä on riittävä tarkkuus, kun otetaan huomioon, että käytettävissäni oli hyvin alkeellisia menetelmiä asettaa kamerat samaan linjaan ja etäisyyteen toisistaan. Lisäksi kameroina toimiva pari web-kameroita eivät olleet huippulaitteita, jotka olisi rakennettu juuri tähän kyseiseen tarkoitukseen.

Testatessa etäisyyslaskentaa siirrettiin tunnistettavaa kappaletta 300, 600 ja 1200 millimetrin etäisyydelle kameroista.



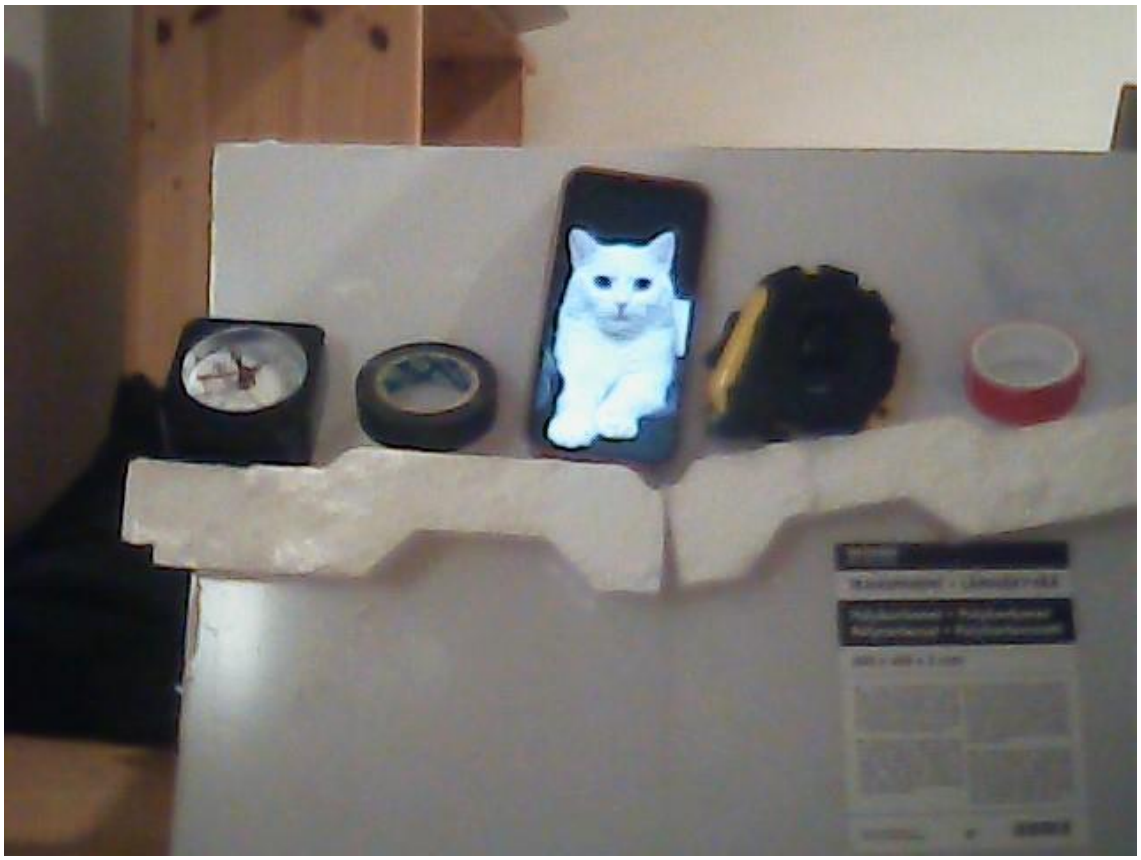
Kuva 12. Kuvassa grafiikkaa etäisyyden tarkkuudesta millimetreinä.



Kuva 13. 3D-grafiikkaa. Havainnot samalta syvyydeltä, X-koordinaatti muuttuu.

Kohteentunnistuksen tarkkuus poikkeaa itse rakennettujen ja OpenCV:n asennuksen mukana tulevien luokittelijoiden välillä. Käytin neutraalin valkoista taustaa testatessani tunnistimia. Mainittakoon, että itse rakennetut luokittelijat löysivät kuvasta kohtia, joissa on mustan ja valkoisen rajoja kun ohjelma tuottaa vääriä havaintoja.

Kohteentunnistuksen tarkkuuden tarkistamisessa luotetaan sekä ohjelman antamiin arvoihin että ohjelman ottamiin kuviin tunnistetusta alueesta. Kuvat käydään läpi käsin, ja erotetaan väärät tunnistukset oikeista tunnistuksista. Lisäksi väärät tunnistukset käydään läpi, ja erotellaan ne, joissa on jokin muu koejärjestelyn tavaroista, tai jokin muu sattumalta tunnistuksen aiheuttanut alue kuvassa.



Kuva 14. Kuvassa käytetty koeympäristö kuvattuna toisen käytetyn kameran läpi. Tunnistettaessa kasvoja oli kasvokuva puhelimen näytöllä. Tavarat on kiinnitetty valkoiseen taustaan.

Seuraavassa taulukossa on kuvattu tarkemmaksi asetetun kappaleentunnistuksen tulokset. Käytin OpenCV:n detectMultiScale-funktion scaleFactor-arvona 1,05 ja minNeighbors arvona 50.

Taulukko 1. Taulukossa vaakarivillä tunnistettavat asiat ja pystyrivillä osumien lukumäärä. Muu tarkoittaa kuvaa muusta kuin mistään tunnistukseen valitusta kappaleesta, katoaa tarkoittaa saman aikaista tunnistuksen puuttumista. Kaikkia kuvattiin 501 ohjelmakierrosta. Ohjelma voi ottaa 2 kuvaa kierrokselta. Tiedot on syötetty manuaalisesti.

	kasvot	kissa	muteippi	rullamitta	punteippi	kello	puoliksi	katoaa	muu
kasvot	1002								
kissa		515					486	1	
muteippi			1002						
rullamitta				744			258		
punteippi									
kello									

Seuraavassa taulukossa on kuvattu vähemmän tarkaksi asetettu tunnistus, scaleFactor arvona 1,05 ja minNeighbors 5.

Taulukko 2. Taulukossa kuvattuna eri tunnistustarkkuudella tapahtuneet tunnistukset. Ohjelmakierroksia myös 501. Tiedot syötetty manuaalisesti.

	kasvot	kissa	muteippi	rullamitta	punteippi	kello	puoliksi	katoaa	muu
kasvot	1002								
kissa		1002							
muteippi			320						682
rullamitta				389					613
punteippi									
kello									

Testatessa ja vertaillessa kohteentunnistusta tuli esiin myös, että ihmiskasvojen tunnistus otti myös jokaisella ohjelmakierrolla tunnistuksen kissan kasvoista, kun kissan kasvokuva asetettiin tunnistusalueelle. Päinvastoin kissan kasvojen tunnistukseen käytettävä luokittelija ei tunnistanut ihmiskasvoja.

Muutettaessa minNeighbors-arvoa pienemmäksi konenäkökirjaston sisältämät luokittelijat näyttävät pitävän tarkkuutensa verrattuna itse rakennettuihin.

5 Yhteenveto

5.1 Yleistä pohdintaa

Opinnäytettä voi pitää onnistuneena. Ohjelma on hyödynnettävissä muihin opinnäytteen tekijän tarpeisiin. Lisäksi selvitystyötä tehdessä syntyneet ratkaisut sekä ohjelma toimivat myös muistiona. Ainoa asia, mikä opinnäytetyöprosessissa epäonnistui, oli ajankäyttö. Opinnäyte myöhästyi alkuperäisestä aikataulusta.

5.2 Syvyyslaskenta

Käytetty syvyyslaskentatapa mahdollistaa minkä tahansa määritellyn alueen koordinaatin hakemisen. Kun vain molemmista kuvista saadaan x-alueella selkeästi määritellyn alueen koordinaatteja, joilla käytetty laskenta tehdään, saadaan etäisyys kohteeseen. Tätä voisi hyödyntää esimerkiksi etäisyyden laskentaan pystyssä tai vaakatasossa olevien linjojen tai muuten eroteltavissa olevien alueiden laskemiseen.

Koordinaattitietoja voisi myös käyttää mallin rakentamiseen fyysisestä ympäristöstä, jonka hyödyntämiseen on mielestäni rajattomat mahdollisuudet.

5.3 Kohteentunnistus

Kohteentunnistuksen luokittelijoita voi rakentaa huolellisemmin. Asioiden tunnistamiseen on myös muita tapoja kuin käytetty tapa. Kohteentunnistuksen tarkkuutta arvioidessa on huomioitava myös sen olevan opinnäytteessä hyvin ympäristöriippuvainen.

Lähteet

1 Davies, E.R. 2005. Machine Vision Theory Algorithms Practicalities. Amsterdam: Boston, Elsevier cop.

2 Boyle, Roger, Hlavac, Vaclav, Sonka, Milan. 1996. Image Processing, Analysis and Machine Vision. Lontoo: Chapman & Hall.

3 Haar Feature-based Cascade Classifier for Object Detection. N.d. OpenCV. Verkkoaineisto. <https://docs.opencv.org/master/d5/d54/group__objdetect.html> Luettu 15.7.2021.

4 Training and applying Haar cascade classifier to detect cats and dogs faces. 2019. Adnan Al-Mnini. Verkkoaineisto. <https://www.youtube.com/watch?v=POSYDLcspIk&list=WL&index=1> Luettu 20.7.2021.

5 cv::CascadeClassifier Class Reference. N.d. OpenCV. Verkkoaineisto. <https://docs.opencv.org/4.x/d1/de5/classcv_1_1CascadeClassifier.html#aaf8181cb63968136476ec4204ffca498> Luettu 18.7.2021.