SeAMK
SEINÄJOEN AMMATTIKORKEAKOULU
SEINÄJOKI UNIVERSITY OF APPLIED SCIENCES

Matti Oksa

Pinnanlaadun valvonta konenäön avulla

Opinnäytetyö
Kevät 2025
Insinööri (AMK), Automaatiotekniikka

SEINÄJOEN AMMATTIKORKEAKOULU

Opinnäytetyön tiivistelmä

Tutkinto-ohjelma: Insinööri (AMK), Automaatiotekniikka

Suuntautumisvaihtoehto: Koneautomaatio

Tekijä: Matti Oksa

Työn nimi: Pinnanlaadun valvonta konenäön avulla

Ohjaaja: Toni Luomanmäki

Vuosi: 2024 Sivumäärä: 40 Liitteiden lukumäärä: 0

Työn toimeksiantaja Mäkelä Alu Oy, ja työn tavoitteena oli kartoittaa automaattisen laadunvalvonnan mahdollisuuksia konenäön avulla. Alumiiniprofiilit tarkastetaan tällä hetkellä silmämääräisesti, jonka jälkeen profiilit pakataan. Nyt osa tästä pakkausprosessista on tarkoitus automatisoida, jonka vuoksi myös laadunvalvonta on tarpeellista.

Aluksi selvitettiin, mitkä ovat yleisimmät pintavirheet alumiiniprofiileissa, ja onko konenäön avulla mahdollista saada luotettavia tunnistuksia viallisista alumiiniprofiileista. Kun löydettiin sopiva ohjelma virheiden tunnistamiseen, tehtiin konenäköohjelma Pythonilla, jonka avulla pystyi kokeilemaan eri kuvista, miten hyvin tunnistus toimii. Lopuksi suunniteltiin logiikkakaavio, miten kommunikaatio pakkaussolun ja konenäköohjelman välillä voisi toimia.

Työn lopputuloksena saatiin todettua, että suuri osa alumiiniprofiilien pintavirheistä tuli näkyviin konenäköohjelmassa, mutta osa pienistä virheistä jäi löytymättä ja jonkin verran virheellisiä tunnistuksiakin tuli. Lisäksi konenäköohjelman ja pakkaussolun välinen logiikkakaavio saatiin suunniteltua.

[1] Asiasanat: Konenäkö, Alumiini, Python, Automaatio, Laadunvalvonta

SEINÄJOKI UNIVERSITY OF APPLIED SCIENCES

Thesis abstract

Degree programme: Automation Engineering

Specialisation: Machine Automation

Author: Matti Oksa

Title of thesis: Surface quality control using machine vision

Supervisor: Toni Luomanmäki

Year: 2024 Number of pages: 40 Number of appendices: 0

The client for this thesis was Mäkelä Alu Oy, and the aim of the work was to explore the possibilities of automated quality control using *machine vision*. Currently, aluminum profiles are inspected visually, and then the profiles are packed. Now part of the packing process will be automated, which creates a need for automatic quality control.

First, the most common surface defects in aluminum profiles were identified, and it was assessed whether reliable detection of defective aluminum profiles is possible using machine vision. After a suitable program for defect detection was found, a machine vision program was created in Python, which allowed testing with various images to evaluate how well the detection works. Finally, a logic diagram was designed to illustrate how communication between the packing cell and the machine vision program could work.

As a result of the work, it was found that a large portion of the surface defects in aluminum profiles became visible in the machine vision program, but some small defects were missed, and there were also some incorrect detections. Additionally, the logic diagram for communication between the machine vision program and the packing cell was successfully designed.

[1] Keywords: Machine vision, Aluminum, Python, Automation, Quality control

SISÄLTÖ

Kuva-, kuvio- ja taulukkoluettelo

Käytetyt termit ja lyhenteet

For-loop	Toistorakenne ohjelmointikielissä, sen avulla voidaan iteroida esimerkiksi tietorakenteita.
ID	ID on yksilöivä tunniste.
PC	Tietokone.
PLC	Ohjelmoitava logiikka.
RGB	On lyhenne sanoista Red, Green ja Blue, lyhenne kertoo, mitä värejä käytetään.

1 JOHDANTO

1.1 Yritys

Mäkelä Alu Oy on Alajärven Luoma-aholla toimiva alumiiniprofiileja valmistava yritys (Mäkelä Alu, i.a.). Yritys on perustettu vuonna 1937 ja se työllistää noin 300 työntekijää. Noin puolet tuotteista menee ulkomaan vientiin.

1.2 Työn tausta

Työn tarkoituksena on tunnistaa alumiiniprofiileista pintavirheet sekä varmistaa alumiiniprofiilin laatu konenäköjärjestelmän avulla. Tällä hetkellä yrityksessä pakataan varastolta tulevat tuotteet käsin häkkeihin. Tämän pakkauksen yhteydessä tuotteet tarkistetaan silmämääräisesti virheiden varalta. Tarkoituksena olisi osittain automatisoida pakkaaminen, tämän myötä syntyy tarve automatisoidulle laadunvalvonnalle.

1.3 Työn tavoitteet ja rajaus

Opinnäytetyön tavoitteena on asettaa vaatimukset konenäköjärjestelmälle sekä selvittää, mitä haasteita monen erilaisen alumiiniprofiilin kuvaus aiheuttaa, ja selvittää onko mahdollista tunnistaa virheet jollain konenäkösovelluksella.

Työ rajattiin teoreettiseksi tutkimukseksi, jossa selvitetään mahdolliset haasteet sekä minkälaiset laitteet ja laitemäärät olisivat sopivia.

1.4 Työn rakenne

Opinnäytetyössä kerrotaan aluksi konenäön perusteet. Tämän jälkeen kerrotaan tarkemmin erilaisista kameroista, valaistustekniikoista sekä miten optiikka tulisi valita. Lisäksi käydään läpi, mitä konenäköohjelma tekee. Teoriaosuuden jälkeen kerrotaan nykytilanne, jossa käydään läpi profiilien valmistusprosessi sekä syitä, miksi konenäköä tarvittaisiin. Tässä käydään läpi, mitä rajoituksia ja haasteita profiilien pinnanvalvonnassa on konenäköön liittyen sekä millainen laitteisto on käytettävissä. Tämän jälkeen tarkastellaan erilaisia

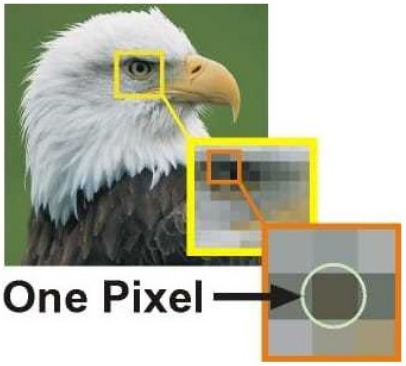
konenäköohjelmia ja tunnistusmenetelmiä sekä tehdään ohjelmat. Seuraavaksi suunnitellaan konenäkösovelluksen logiikkakaavio ja mallijärjestelmä. Lopuksi käydään läpi tulokset ja johtopäätökset.

2 KONENÄKÖ

2.1 Konenäön perusteet

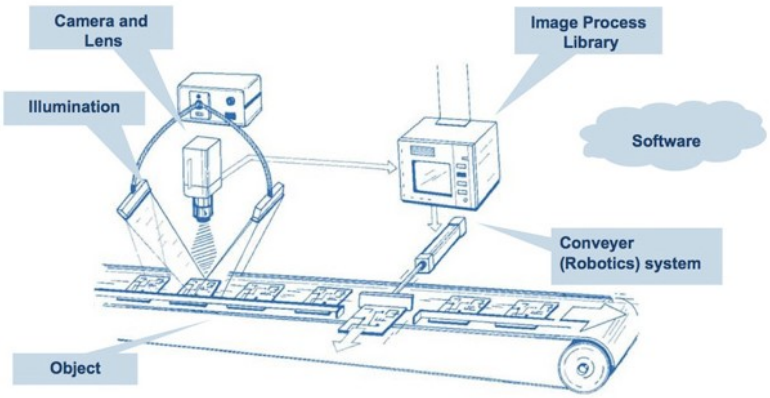
Konenäkö antaa nimensä mukaan teollisuuden laitteille digitaalisen näköaistin (Intel, i.a.). Konenäkö perustuu kameran ottamiin digitaalisiin kuviin. Näitä kuvia käsitellään automaattisesti, jonka avulla pyritään saavuttamaan haluttu tunnistus. Konenäköä käytetään monenlaisiin erilaisiin tunnistustehtäviin, mm. laadun tarkkailuun, kappaleen paikoitukseen, mittaukseen ja moniin muihin tehtäviin.

Konenäkö tarvitsee toimiakseen kameran, optiikan, konenäköohjelman ja lähes aina valaistuksen (Robotics Tomorrow, 2019). Konenäön toiminta perustuu siihen, että tarkasteltava kohde valaistaan, ja kohde heijastaa valoa kameran optiikalle, joka skaalaa valon kameran valoherkälle kennolle. Kenno koostuu useista varausyksiöistä, joita kutsutaan pikseleiksi (kuva 1). Nämä pikselit varautuvat sähköisesti riippuen valon kirkkaudesta ja valotusajasta. Pikseleiden varaus muutetaan digitaaliseksi yleensä 8-, 10- tai 12-bittiseksi (Vision Doctor, i.a.-c). 8-bittisellä syvyydellä voidaan esittää 256 värisävyä, jossa musta on 0 ja valkoinen 255, näillä pikseleillä muodostetaan kuva.



Kuva 1. Pikseli (Ripe Media, 2013)

Kuvanmuodostuksen jälkeen kuva välitetään konenäköohjelmistolle, jossa kuvaa käsitellään halutulla tavalla, ja sen jälkeen matemaattisesti etsitään haluttu ominaisuus kuvasta (VICO Imaging, 2024). Kun ohjelma on suorittanut etsinnät, se lähettää halutun signaalin eteenpäin, jos siihen on tarvetta. Kuva 2 havainnollistaa yhdenlaista konenäköjärjestelmää.



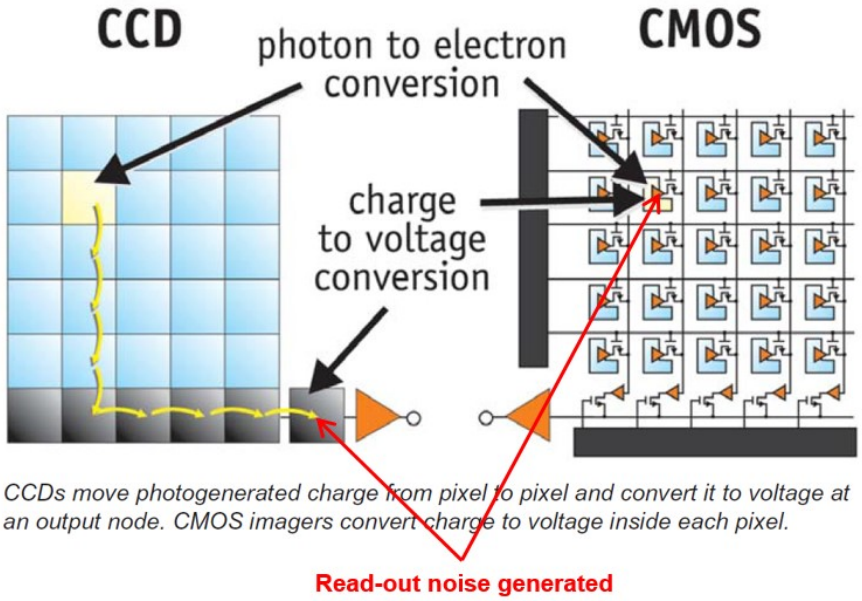
Kuva 2. Konenäköjärjestelmä (Robotics Tomorrow, 2019)

2.2 Kamera

Kamera on keskeinen komponentti konenäössä (Vision Doctor, i.a.-c). Se muodostaa kuvan kuvattavan alueen heijastamasta valosta. Konenäkökameroita on saatavina monenlaisina eri tarkoituksiin sopivina. Kameraa valittaessa on huomioitava, mitä vaatimuksia kuvattava kohde asettaa.

2.2.1 Kennot

Konenäkökamera tarvitsee kennon toimiakseen (Universe Optics, i.a.). Kennon tehtävä on muuttaa kennolle saapuva valo sähköisiksi varauksiksi, joiden perusteella luodaan kuva. Kenno on yleensä CCD- tai CMOS-tyyppinen. Kennojen rakenteessa on erona se, että CMOS-tyypissä kennon varaus muunnetaan heti jännitteeksi (kuva 3). CCD-kennon etuna on parempi kuvanlaatu ja vähemmän virheellisiä pikseleitä kuvaa kohden. Kennon heikkouksia ovat valmistushinta ja virrankulutus. CMOS-kenno on yleensä pienempi ja se kuluttaa vähemmän virtaa. Koska se kuluttaa vähemmän virtaa, se ei myöskään lämpene niin paljon kuin CCD-kenno.

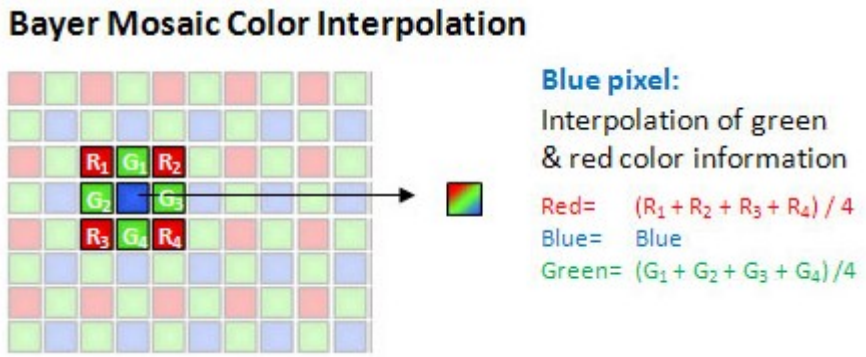


Kuva 3. CCD- ja CMOS-kennon rakenne (PetaPixel, 2021)

CMOS-kennon vähäisen virrankulutuksen ja pienen koon myötä se vakiintui matkapuhelinteollisuuteen paremmaksi vaihtoehdoksi (Sagacious Research, i.a.). Tämän valinnan vuoksi CMOS-kennon kehitys on mennyt paljon eteenpäin. Kennon koko vaikuttaa myös kuvanlaatuun: isompi kenno on tarkempi, mutta myös kalliimpi.

Kennot voivat olla joko harmaasävy- tai värikennoja (SAMK Automaatio, i.a.). Harmaasävykennossa pikselit keräävät valoa kaikilta värin spektreiltä, jolloin kuvasta tulee mustavalkoinen. Harmaasävykennoa ei suodateta, joten se kerää fotoneita nopeammin kuin värikennot, tästä syystä se on myös herkempi valolle. Harmaasävykamera on minimissään 8-bittinen ja maksimissaan 12-bittinen, kun taas värikamera on yleensä 24-bittinen. Tämän takia harmaasävykamera on myös nopeampi, koska dataa liikkuu vähemmän.

Yksi tapa muodostaa värikuva on käyttää kennoa, jossa pikseleissä on suodatin, ja joka ei päästä läpi kuin tiettyä valon spektriä, yleensä punaista, vihreää tai sinistä (Vision Doctor, i.a.-a). Näitä päävärejä yhdistämällä saadaan muodostettua muut värit. Pikselien suodattimet on asetettu mosaic-järjestykseen (kuva 4). Tämän järjestyksen avulla voidaan laskea joka pikselille RGB-arvot. Kuvassa 4 lasketaan lineaarisen interpolaation avulla siniselle pikselille punainen ja vihreä arvo, hyödyntäen viereisiä punaisia ja vihreitä pikseleitä. Tämän jälkeen sinisellä pikselillä on RGB-arvo, jolla voidaan muodostaa oikea väri (kuva 5).



Kuva 4. Bayer Mosaic Interpolation (Vision Doctor, i.a.-a)

255 255 0	163 74 58	222 127 99	236 132 93
255 255 0	104 41 26	193 121 96	252 172 137
255 255 0	255 255 0	255 255 0	255 255 0

Kuva 5. RGB-pikselit (Stack Overflow, 2016)

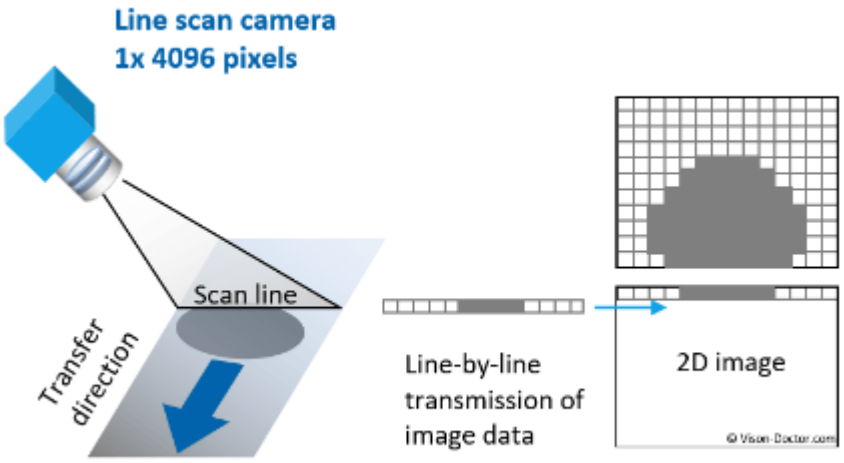
Värikuvan saa muodostettua myös kameralla, jossa on kolme kennoa (JAI, i.a.). Kennoihin valo johdetaan prisman läpi. Kennoissa on suodattimet pääväreille. Kolmekennoisen kameran etuna on parempi värien tarkkuus, mutta se vaatii erikoisoptiikan, jolla on korkeampi hinta ja rajoitettu käyttölämpötila

2.2.2 2D-Kamera

2D-kamera eli matriisikamera on yleisin konenäössä käytettävä kameratyyppi (Zebra Technologies, i.a.). Kameran kennossa on pikseleitä leveyssuunnassa ja korkeussuunnassa. Kertomalla pikseleiden leveys- ja korkeusmäärä keskenään saadaan resoluutio.

2.2.3 Viivakamera

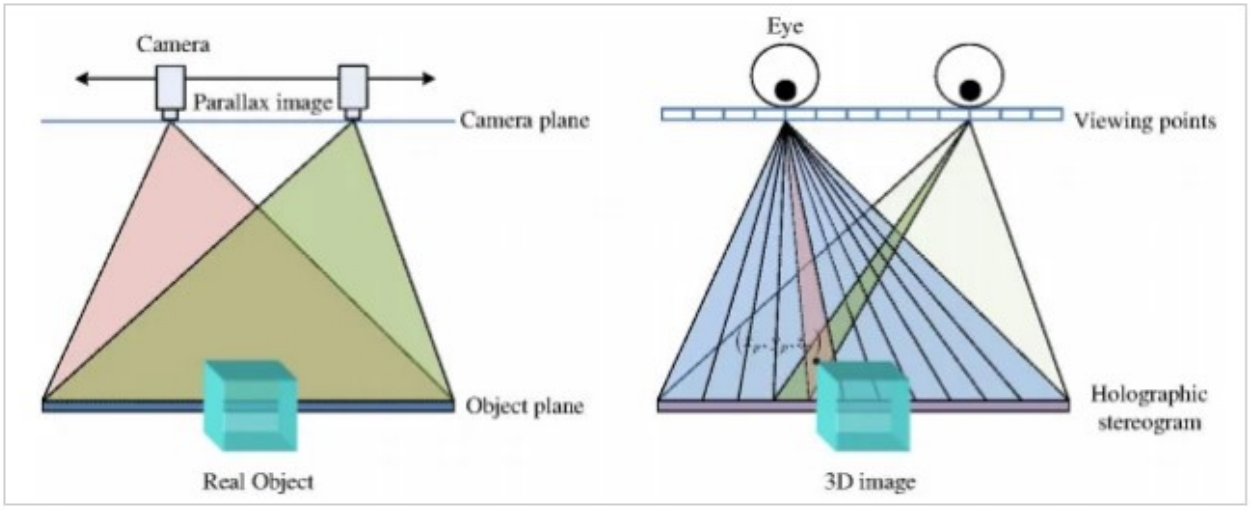
Viivakamera kuvaa nimensä mukaan viivan pikseleitä kerralla (kuva 6) (Vision Doctor, i.a.-b). Tyypilliset resoluutiot viivakameralle ovat 1024, 2048, 4096, 8192 tai 12288 x 1. Viivakamera vaatii pulssianturin, jonka avulla voidaan laskea kappaleen tai kameran siirtymä, jolloin pystytään ottamaan seuraava kuva. Kun riittävän monta kuvaa on otettu, voidaan näistä muodostaa matriisikuva.



Kuva 6. Viivakamera (Vision Doctor, i.a.-b)

2.2.4 3D-kamera

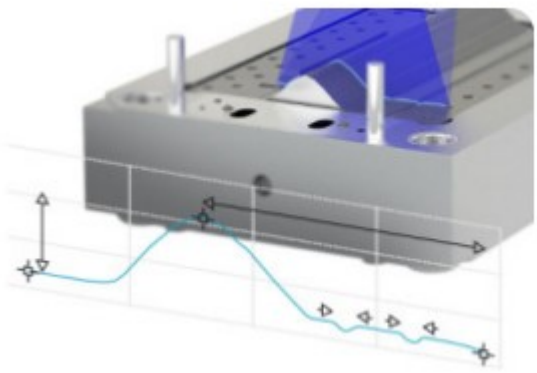
3D-kameran toiminta perustuu siihen, että kuva otetaan kahdesta eri paikasta, joko kahdella kameralla tai liikuttamalla kameraa, jolloin saadaan stereoskooppinen kuva (kuva 7) (All About Circuits, 2020). Tämän kuvan avulla voidaan muodostaa 3D-kuva, jolla saadaan kappaleiden muodot esiin.



Kuva 7. 3D-Kamera (All About Circuits, 2020)

2.2.5 Profiiliskanneri

Profiiliskannerilla kuvataan kappaleen pinnan muotoja (Micro-Epsilon, i.a.-b). Toiminta perustuu siihen, että kuvattavan kohteen pinnalle valaistaan laseriviiva, jota kamera kuvaa pienestä kulmasta. Tämän kulman ansiosta laseriviiva siirtyy kameran kuvassa, kun kappaleen muoto muuttuu. Tämä siirtymä mahdollistaa kappaleen muotojen laskemisen (kuva 8).



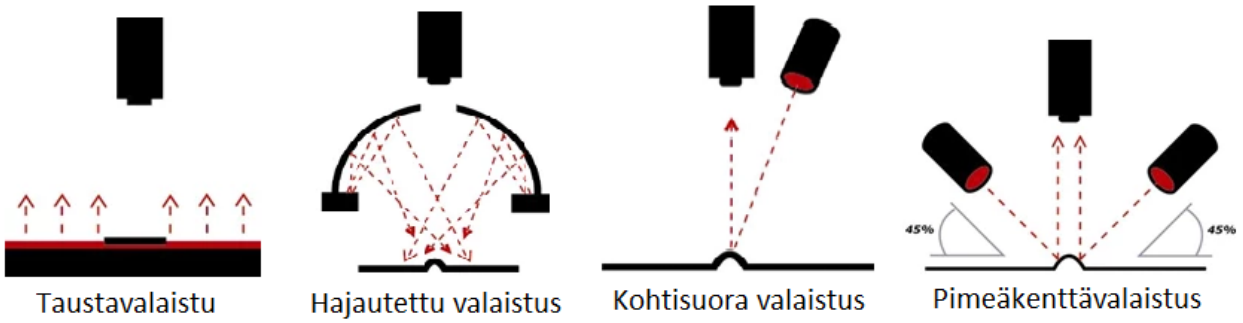
Kuva 8. Profiiliskanneri (Micro-Epsilon, i.a.-b),

2.3 Valaistus

Valaistus on tärkeä osa konenäköä, koska ilman valoa kameran kennon pikselit eivät varaudu, tällöin kuvasta tulee musta (Advanced Illumination, 2013). Valaistustekniikalla voidaan vaikuttaa huomattavasti siihen, miltä kuvattava kohde näyttää. Valaistukseen ei ole oikeaa vaihtoehtoa, vaan valaistus pitää räätälöidä käyttökohteen mukaan riippuen siitä, mitä kappaleesta halutaan korostaa. Valaistusjärjestelmän suunnittelussa on huomioitava ulkoiset tekijät. Ei-halutun valon pääsy kuvattavaan kohteeseen on syytä estää, jotta kuvattava kohde olisi aina samalla tavalla valaistu. Ulkoisia tekijöitä ovat esim. auringon valo, huoneen valaistus, peilaavasta pinnasta heijastuva valo tai varjot. Mitä pidempi valaistusaika kuvalle annetaan, sitä herkemmin siihen vaikuttavat ulkoiset tekijät.

Valaistustekniikoilla voidaan vaikuttaa paljon siihen, mitkä ominaisuudet kohteesta korostuvat. Valaistuksen värillä voidaan korostaa tietyn värisiä piirteitä kuvattavasta kohteesta. Punainen valo lisää punaisen värisen piirteen intensiteettiä, jolloin harmaasävykamerassa nämä piirteet näkyvät kirkkaampina (Vision Doctor, i.a.-d).

Erilaisia valaistustyylejä on monia, näistä muutamia ovat taustavalaistus, hajautettu valaistus, kohtisuora valaistus ja pimeäkenttävalaistus (kuva 9) (Advanced illumination, 2013).



Kuva 9. Valaistustekniikat (Advanced illumination, 2013)

2.4 Optiikka

Optiikka määrittelee, miten kuva muodostuu valoherkälle kennolle (Chouinard, 2015). Optiikkan voi valita, kun tiedetään kuvausalue, kuvausetäisyys, kameran resoluutio ja kennon koko.

Polttoväli saadaan laskettua kaavalla 1 (Chouinard, 2015):

$$FL = (Ss * WD)/Fov \quad (1)$$

missä

FL	on polttoväli
Ss	on kennon koko
WD	on kameran etäisyys
Fov	on kameran näkemä alue

2.5 Konenäköohjelmat

Konenäköohjelmalla käsitellään kuvaa halutulla tavalla, jotta halutut piirteet korostuvat tai ei-halutut piirteet katoavat (Labellerr, 2022). Kuvankäsittely pohjautuu matemaattisiin menetelmiin, esim. jos 8-bittisestä harmaasävykuvasta halutaan tehdä mustavalkoinen, tietyn kynnyksen mukaan, valitaan esimerkiksi kynnykseksi pikselin kirkkausarvo 128 (GeeksforGeeks, 2024). Kaikki kuvassa tämän arvon yli menevät pikselit muuttuvat numeroksi 255, jolloin ne ovat valkoisia, ja taas toisin 128 tai alle jäävät arvot muuttuvat nollaksi, jolloin ne ovat mustia.

Näitä erilaisia kuvankäsittelytapoja on paljon ja usein oikeat menetelmät löytyvät kokeilemalla.

2.6 Kameran valintavaatimukset

Jotta konenäköjärjestelmästä saadaan toimiva, on huolellisesti suunniteltava, mitä kriteerejä konenäköjärjestelmän on täytettävä.

Kun konenäköjärjestelmään valitaan kameraa, pitää tietää pienimmän tunnistettavan kohteen koko, jolloin voidaan laskea erotustarkkuus (Keyence, i.a.). Nyrkkisääntönä pidetään, että vähimmäismäärä pikseleitä per tunnistettava kohde on 2–4 pikseliä.

Erotustarkkuus voidaan laskea kaavalla 2 (Keyence, i.a.):

$$R_s = \frac{S_f}{N_f} \tag{2}$$

Missä

R_s on kameran erotustarkkuus [mm/pikseli]

S_f Tunnistettavan kappaleen koko [mm]

N_f Pikseleiden määrä tunnistettavaa kokoa kohden [kpl]

Vaadittava resoluutio voidaan laskea kaavalla 3 (1st Vision, 2024):

$$R_C = \frac{FOV}{Rs} \tag{3}$$

Missä

R_C on kameran resoluutio [pikseli]

FOV on vaadittavan kuvausalueen koko, x- tai y-akseli [mm]

R_s on kameran erotustarkkuus [mm/pikseli]

3 KONENÄÖN KÄYTETTÄVYYDEN SELVITYS

3.1 Nykytilan kuvaus

Alumiiniprofiilit valmistetaan puristamalla kuuma alumiinitukki muotin läpi, jolloin siitä saadaan halutun muotoinen profiili. Puristuksen jälkeen profiilit leikataan haluttuun mittaan, josta ne menevät lämpökäsittelyuuniin. Uunista profiilit viedään varastoon ja sieltä eteenpäin. Tämän puristuksen ja kuljetuksen yhteydessä profiileihin voi tulla erilaisia vaurioita. Tällä hetkellä alumiiniprofiilit tarkistetaan silmämääräisesti ennen niiden pakkaamista. Erilaisia alumiiniprofiileja valmistetaan vuodessa satoja, jonka takia profiilien opettaminen konenäköjärjestelmälle ei ole vaihtoehto. Opettamista hankaloittaa myös se, että profiilit olisivat vierekkäin kuvattavalla alueella, jolloin ei reunatunnistus onnistu.

3.2 Käytettävissä oleva laitteisto ja ohjelmisto

Käytettävissä SeAMKin puolesta olivat kamera, optiikka, valaistus ja MVTec Merlic -konenäköohjelmisto. Mäkelä Alu Oy:n puolelta saatiin ohjeet laatuluokista ja karkea kuvaus tulevasta pakkaussolusta.

Laitteistot ja ohjelmistot olivat seuraavat:

- kamera
- TPL Vision ebar-250-whi-10
- Optiikka
- Mvtec Merlic
- Octave
- Cognex in-Sight Explorer
- Python.

3.3 Työn valmistelu

Työ aloitettiin selvittämällä alumiiniprofiilin valmistusvaiheet. Tällöin tarkkailtiin pakkaamossa, minkälaisia pintavirheitä valmistuksessa profiileihin muodostuu. Yleisimmät virheet

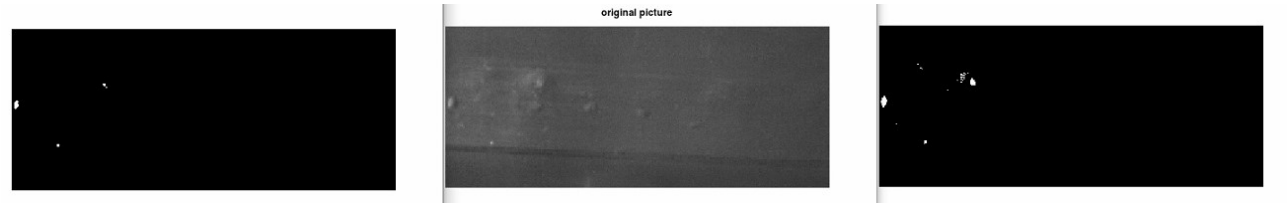
olivat ilmakuplat ja jatkoskohdat. Näistä virheellisistä profiileista leikattiin lyhyitä pätkiä testattavaksi. Virheelliset profiilit kuljetettiin koululle kuvaamista varten. Kuvaamiseen käytettiin koulun kameraa ja näitä kuvia käytettiin kaikilla ohjelmistoilla. Tässä vaiheessa kokeiltiin erilaisia valaistustyylejä, joista pimeäkenttävalaistus osoittautui parhaaksi, koska tällä saatiin helposti kirkastettua pintavirheet profiilista.

3.4 Cognex In-Sight Explorer

Kuvia käsiteltiin Cognex In-Sight Explorerilla, mutta se osoittautui välittömästi liian kankeaksi erilaisten alumiiniprofiilien suuren määrän vuoksi. In-Sightissa on mahdollista tehdä myös laskentaa Excel-pohjalla, mutta In-Sight Explorerin tutkiminen lopetettiin, koska Octave ja Python ovat laskemiseen huomattavasti sopivampia ja joustavampia.

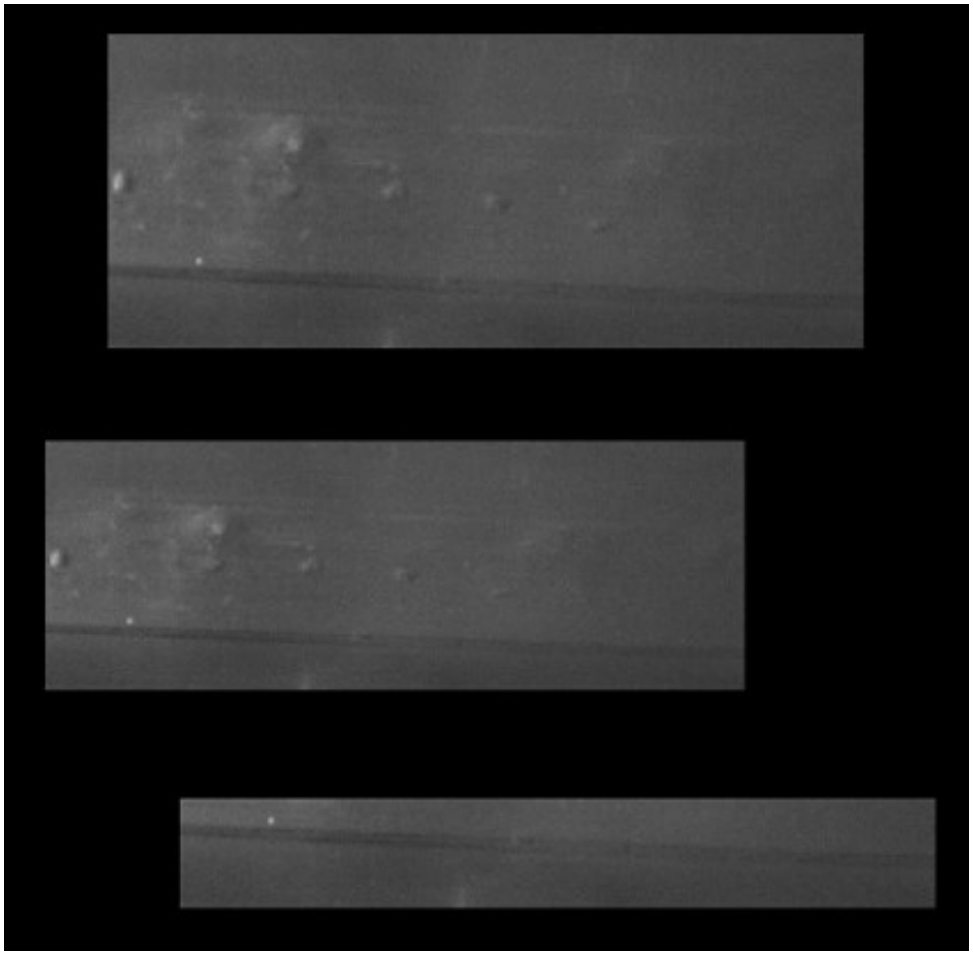
3.5 Octave

Octavella yritettiin tutkia kappaleita ja suodattaa näitä eri menetelmillä. Kuvassa 10 on otettu Cognexin älykameralla kuva huonossa valaistuksessa, mutta tarkoituksena oli testata, miten virheet saadaan näkyviin suodatuksen avulla. Ensimmäisenä kuva luettiin imread-komennolla, jonka jälkeen käytettiin funktiota im2bw, joka muuttaa pikselit joko mustiksi tai valkoisiksi. Kuvassa 10 vasemmalla on käytössä Octaven funktio im2bw parametreillä (kuva,0.5), keskellä on alkuperäinen, kuva mistä suodatus on otettu. Oikeanpuoleisessa kuvassa on käytössä funktio im2bw parametreillä (kuva,0.45). Kuten kuvasta nähdään, jo pelkällä suodattamisella on mahdollista tunnistaa pinnanlaatuvirheet (kuva 10).



Kuva 10. Suodattaminen Octavella (Oksa, 2023).

Seuraavaksi yritettiin erotella profiilit, jos kuvassa on selkeät reunukset, otettiin näistä kuvista histogrammit ja suodatukset (kuva 11).



Kuva 11. Aloituskuva Octaveen (Oksa, 2023).

Kuva luetaan Octaveen, jonka jälkeen tarkastetaan, onko kuva värikuva. Jos kuva on värikuva, tehdään siitä harmaasävykuva. Tämän jälkeen kuvasta otetaan talteen binäärikuva, jossa kaikki alle 20 värin pikselit muuttuvat mustiksi ja loput valkoisiksi. Tästä binäärikuvasta luodaan merkityt alueet matriisiin, eli binäärikuva muokataan siten, että nämä kolme aluetta ovat matriisissa joko 1, 2 tai 3 ja musta osuus on 0. Samassa otetaan myös talteen objektien määrä eli 3 (kuva 12).

```
%Luetaan kuva musta2
I = imread('musta2.png');

%alkuperäinen kuva figure 100
figure 100
imshow(I);

%chekataan onko kuva värikuva jos on niin konvertataan se grayscale kuvaksi
if (ndims(I) == 3 && size(I,3) == 3)
    I = rgb2gray(I);
endif

% Create a binary image by thresholding the grayscale image
binaryImage = I > 20;

% Label the connected components in the binary image
[labeledImage, numberOfObjects] = bwlabel(binaryImage);

% Measure properties of the labeled regions
measurements = regionprops(labeledImage);
```

Kuva 12. Kuvan käsittely Octavessa (Oksa, 2023).

Kun objektien määrä on saatu talteen, voidaan indeksien avulla toistaa (for-loop) halutut jatkokäsittelyt, jottei samaa koodia tarvitse kirjoittaa uudestaan moneen kertaan. Aluksi otetaan talteen jokaisen objektin reunalaatikko, joka on neliön muotoinen ja kooltaan mahdollisimman pieni, mutta joka kuitenkin sisältää kaikki objektin osat. Näistä reunalaatikoista otetaan talteen aloituspisteen koordinaatit x ja y, sekä reunalaatikon pituus ja korkeus, jonka jälkeen näistä tehdään matriisit (kuva 13).

```

% Loop through each object to get its bounding box and area
for k = 1:numberOfObjects
BoundingBoxes{k,1} = measurements(k).BoundingBox;
Area{k} = measurements(k).Area
end

% Convert cell arrays to matrices
BoundingBoxes = cell2mat(BoundingBoxes)
Area = cell2mat(Area)

% Extract bounding box properties for each object
for k = 1:numberOfObjects
    xmitat{k}=BoundingBoxes(k,1)
    ymitat{k}=BoundingBoxes(k,2)
    pituus{k}=BoundingBoxes(k,3)
    korkeus{k}=BoundingBoxes(k,4)
end
% Convert cell arrays to matrices
xmitat = cell2mat(xmitat)
ymitat = cell2mat(ymitat)
pituus = cell2mat(pituus)
korkeus = cell2mat(korkeus)
```

Kuva 13. Profiilien reunalaatikoiden talteen ottaminen (Oksa, 2023).

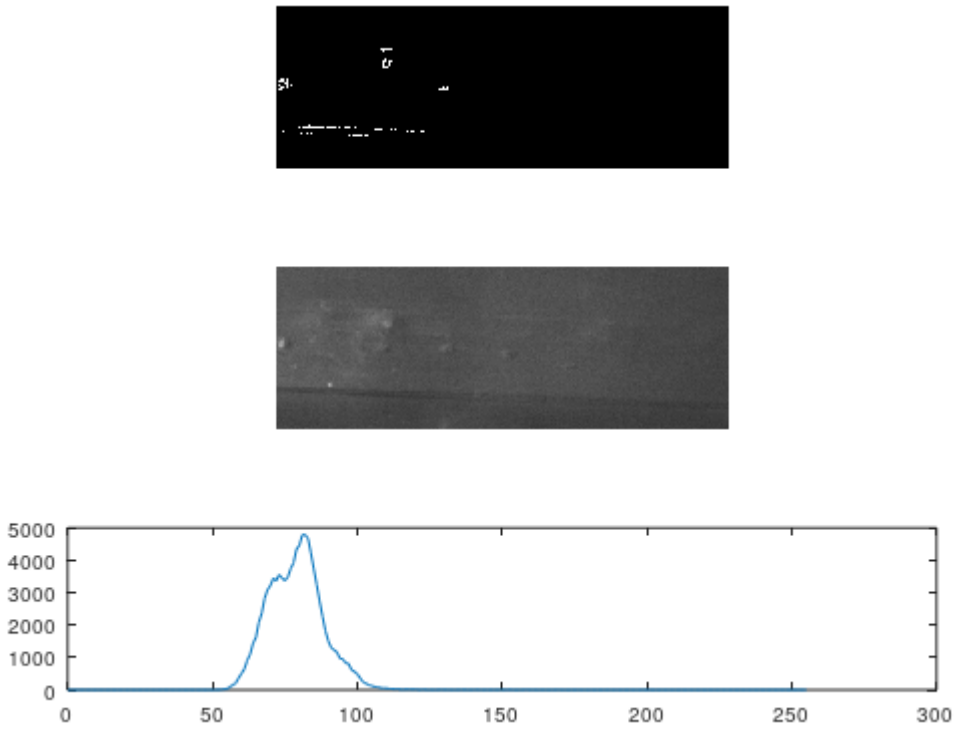
Kuvasta leikataan reunalaatikot, joista tehdään rajatut kuvat. Näistä kuvista otetaan histogrammit ja reunat Prewitt-metodilla. Tämä osoittautui parhaaksi tavaksi reunatunnistukseen testien perusteella. Seuraavaksi binäärikuvista poistetaan liian pienet alueet Bwareopen-komennolla, jossa suodatus on 10 pikseliä. Tämän jälkeen kuvat tulostetaan niin, että näytetään alkuperäinen leikattu kuva, histogrammi ja binäärikuva (kuva 14).

```
% Process each object
for k = 1:numberOfObjects
% Define the region of interest (ROI) for cropping
row_start = xmitat(k);
row_end = pituus(k);
col_start = ymitat(k);
col_end = korkeus(k);

% Extract the pixel values of the specific area
cropped_image = imcrop(I, [row_start, col_start, row_end-1, col_end-1]);
cropatut{k}=cropped_image;

% Compute the histogram of the cropped image
[counts, binLocations] = imhist(cropped_image);

% Detect edges using the Prewitt method
edgeImg = edge(cropped_image, "Prewitt");
BW = bwareaopen(edgeImg,25);

figure (k)
% Show cropped_image
subplot(3,1,2);
imshow(cropped_image);
% Plot the histogram
subplot(3,1,3);
plot(binLocations, counts);
subplot(3,1,1);
imshow(BW);


end
```

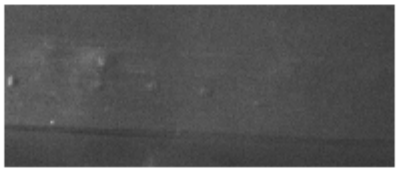
Kuva 14. Reunalaatikoiden leikkaus ja tuloksien tulostus (Oksa, 2023).

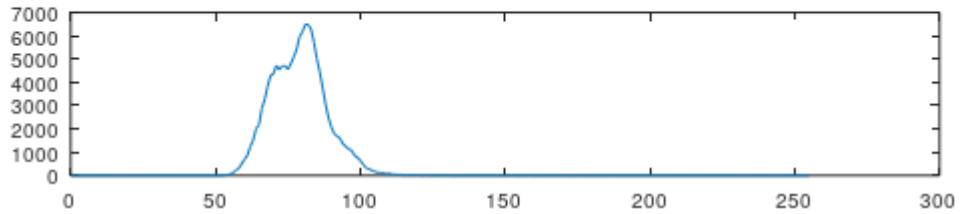
Kuvista 15–17 näkee selkeästi, että binäärikuvalla saadaan joitain pinnan epäpuhtauksia esiin, vaikka lähdekuva on huonolaatuinen ja valaistus on huono. Histogrammi ei ole tässä kuvassa kovin hyödyllinen huonon valaistuksen takia, mutta jos valaistus olisi hyvä pimeäkenttävalaistus, luultavasti histogrammi näyttäisi piikkiä 240–255 alueella ilmakuplien takia. Kuvista selviää myös, että jos profiilit ovat irrallaan toisista, olisi luultavasti mahdollista saada joka profiilista oma pinnanlaatutarkkailu reunalaatikoiden avulla. Tämä vaatisi kuitenkin riittävän kontrastin profiilinreunan ja taustan kanssa.

Octaven ja Cognexin testien jälkeen tulos ei ollut vielä tyydyttävä, joten tutkittiin vaihtoehtoisia avoimen lähdekoodin ohjelmia. Tällöin löydettiin OpenCV-kirjasto, jota voidaan käyttää C++- ja Python-ohjelmointikielillä. Tämän työn tekijällä ei ollut aikaisempaa kokemusta kummastakaan ohjelmointikielestä, mutta Python vaikutti yksinkertaisemmalta ja soveltuu tämän vuoksi huomattavasti paremmin testaamiseen.

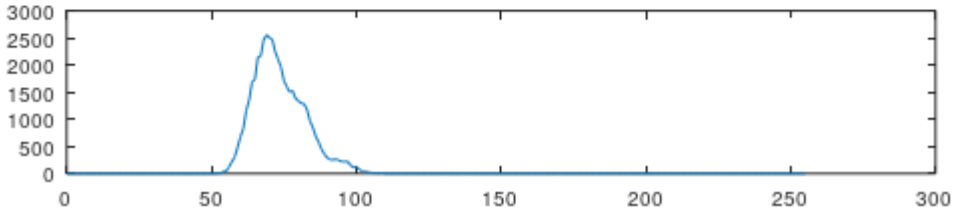


Kuva 15. Suodatus ja histogrammi kuvan 11 ylimmästä profiilista (Oksa, 2023).





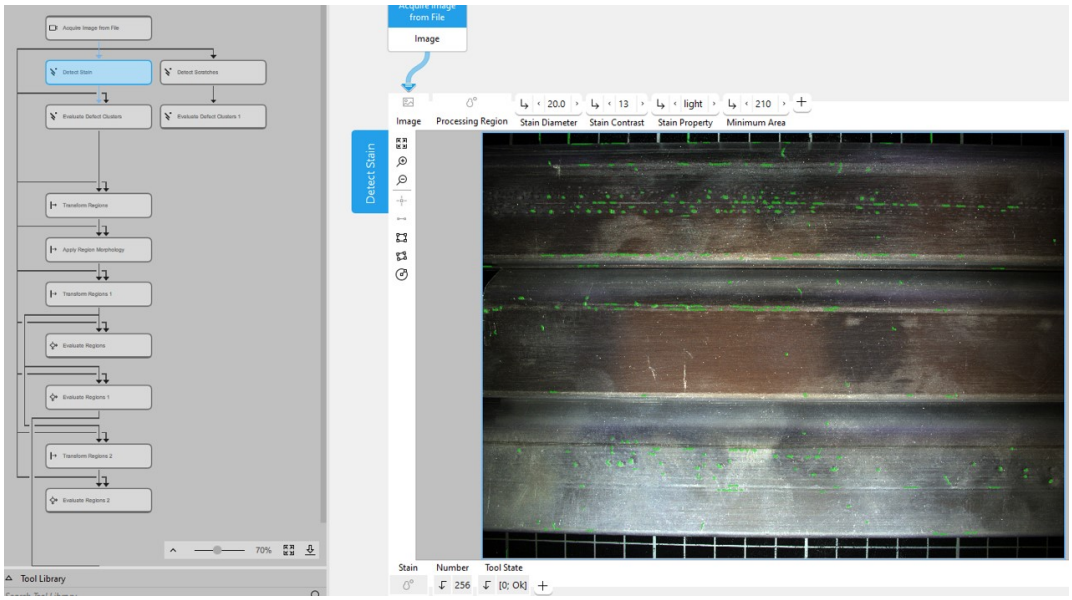
Kuva 16. Suodatus ja histogrammi kuvan 11 keskimmäisestä profiilista (Oksa, 2023).



Kuva 17. Suodatus ja histogrammi kuvan 11 alimmasta profiilista (Oksa, 2023).

3.6 MVTec Merlic

SeAMKlla oli käytössä MVTec Merlic -ohjelmisto. Ohjelmisto oli helppokäyttöinen ja siihen oli valmiiksi kameravalmius, jonka takia profiilien kuvaukseen käytettiin tätä ohjelmistoa. Merlicillä voi käsitellä kuvia ja etsiä niistä erilaisia asioita riippuen millaisia eri suodattimia käytetään. Kuvasta 18 voidaan nähdä, että pinnanlaatua on mahdollista valvoa, mutta asetuksien säätäminen sopiviksi oli haastavaa.



Kuva 18. MVTec Merlic (Oksa, 2023).

3.7 Python ja OpenCV

OpenCV-kirjasto ja Python-ohjelmointikieli osoittautuivat ehdottomasti parhaaksi yhdistelmäksi, koska siinä oli täysi kontrolli, miten kuvaa käsitellään ja mitä kuvasta otetaan talteen. Kuvat profiileista otettiin Merlicillä ja sen jälkeen niitä käsiteltiin Pythonilla.

3.7.1 Mallinsovitus

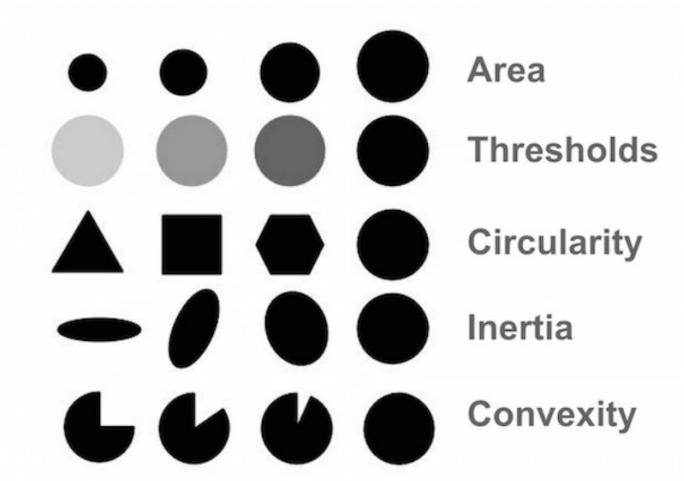
Ensiksi kokeiltiin mallinsovituskomentoa. Komento toimii niin, että otetaan pieniä kuvia tunnistettavasta asiasta, ja sitten näitä kuvia sovitetaan haluttuun kuvaan. Jos mallin kaltainen kohta löytyy kuvasta, saadaan siitä osuma. Esimerkiksi kuvassa 19 on mallina Super Mario -pelin kolikot ja näitä etsitään sitten kokonaisesta kuvasta. Osumat voidaan rajata ja laskea.

Kuva 19. Template matching (OpenCV).

Mallin sovitus ei kuitenkaan ollut toimiva vaihtoehto, koska ilmakuplat ovat niin erilaisia toisiinsa nähden ja malleja tarvittaisiin paljon. Tämä mallien paljous hidasti ohjelmaa huomattavasti, vaikka malleja oli vain kymmenen. Testien mukaan näitä malleja olisi tarvittu paljon enemmän kuin kymmenen, joten päätettiin etsiä parempia vaihtoehtoja.

3.7.2 Blob detector

OpenCV-kirjaston Blob detector -toiminto vaikutti lupaavalta ilmakuplien tunnistukseen. Tämä toiminto tunnistaa erilaisia yhtenäisiä alueita ja sillä voidaan säätää parametrejä, jotta saadaan tunnistus halutunlaiseksi. Parametreistä löytyy säädöt alueen koolle, kontrastierolle, alueen pyöreydelle, alueen pitkulaisuudelle ja kuperuudelle (kuva 20).



Kuva 20. Blob detector -parametrit (LearnOpenCV).

Blob detectoria testattiin niin, että tuotiin ensin tarvittavat OpenCV-, numpy- ja time-kirjastot Python-ohjelmaan. Koska haluttiin testata erilaisia parametriarvoja, lisättiin takaisinkytkentäkomento halutuille parametreille. Tälle takaisinkytkentäkomennolle tehtiin ikkuna,

jossa oli liukusäätimet parametreille. Kuvassa 21 tuodaan tarvittavat kirjastot ja luodaan liukusäätimet ohjelmaan (kuva 21).

```python
import cv2
import numpy as np
import time
#from matplotlib import pyplot as plt

# Trackbar callback function to update HSV values and other parameters
def callback(x):
    global minArea,maxArea,minRepeatability,minInertia,V_low,V_high
    #assign trackbar position value to H,S,V High and low variable
    minArea = cv2.getTrackbarPos('minArea','controls')
    maxArea = cv2.getTrackbarPos('maxArea','controls')
    minRepeatability = cv2.getTrackbarPos('minRepeatability','controls')
    minInertia = cv2.getTrackbarPos('minInertia','controls')
    #V_low = cv2.getTrackbarPos('low V','controls')
    #V_high = cv2.getTrackbarPos('high V','controls')


#create a seperate window named 'controls' for trackbar
cv2.namedWindow('controls',2)
cv2.resizeWindow("controls", 550,10);


# Global variables initialization
minArea = 0
maxArea = 179
minRepeatability= 2
minInertia = 0
V_low= 0
V_high = 0

# Create trackbars to adjust the parameters
cv2.createTrackbar('minArea','controls',35,350,callback)
cv2.createTrackbar('maxArea','controls',550,1500,callback)

cv2.createTrackbar('minRepeatability','controls',2,8,callback)
cv2.createTrackbar('minInertia','controls',14,99,callback)

#cv2.createTrackbar('low V','controls',0,100,callback)
#cv2.createTrackbar('high V','controls',100,100,callback)

```

Kuva 21. Kirjastojen lisäys ja liukusäätimien tekeminen (Oksa, 2023).

Kun liukusäädin on valmis, laitetaan ohjelma loopiin (kuva 22) ja aloitetaan ottamalla aloitusaika talteen, jotta voidaan myöhemmin tarkastella ohjelmakierron nopeutta. Jotta kuvaa voidaan käsitellä, luetaan se img-muuttujaan komennolla cv2.imread kohdepolusta. Koska kuvat olivat isoja, skaalattiin näitä kuvia 35 % pienemmiksi. Skaalauksesta aiheutuu hieman pikselikatoa. Kuvan skaalauksen jälkeen kuvasta tehdään harmaasävykuva ja pikseliarvoista tehdään käänteiset, jonka jälkeen kuvalle tehdään Gaussian sumennus 5x5 kernelillä, jotta kuvasta vähenee kohina (kuva 22).

```
42  while(1):
43      start_time = time.time()
44
45      # Read the input image
46      #img =cv2.imread('kuvat/2023_04_21T10-06-43_971_Me_Image.png')
47      img =cv2.imread('kuvat/2023_04_21T10-14-40_702_Me_Image.png')
48
49      # Resize the image to 35% of its original size
50      scale_percent = 35
51      width = int(img.shape[1] * scale_percent / 100)
52      height = int(img.shape[0] * scale_percent / 100)
53      dim = (width, height)
54      img = cv2.resize(img, dim, interpolation = cv2.INTER_AREA)
55
56      # Convert the image to grayscale
57      gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
58
59      # Set up the detector with default parameters.
60      im=cv2.bitwise_not(gray)
61      # Apply Gaussian blur to the image
62      im = cv2.GaussianBlur(im,(5,5),3)
```

Kuva 22. Kuvan luenta (Oksa, 2023).

Kuvankäsittelyn jälkeen voidaan alustaa Blob detectorin parametrit. Parametrit on löydetty kokeilemalla, mikä tuottaa parhaan tuloksen. Parametrien asetuksien jälkeen voidaan blob detectoria käyttää kuvaan. Osumien määrä tallennetaan muuttujaan, jotta voidaan käsitellä numeerista lukua (kuva 23).

```
    # Set up the parameters for blob detection
    params = cv2.SimpleBlobDetector_Params()
    params.filterByArea = True
    params.minArea = minArea + 1
    params.maxArea = maxArea
    params.filterByCircularity = True
    params.minCircularity = 0.35
    params.maxCircularity = 1
    params.filterByColor = False
    params.filterByInertia = True
    miniInertia=minInertia/100 + 0.01
    params.minInertiaRatio = miniInertia
    print(miniInertia,"mintresh")
    params.maxInertiaRatio = 1
    params.minThreshold = 0
    params.maxThreshold = 255
    #params.minDistBetweenBlobs = 0.00001

    # So minRepeatability is how a blob is
    # stable across different thresholds on the grayscale image.

    params.minRepeatability=minRepeatability + 1
    params.thresholdStep = 25
    params.filterByConvexity = False

    # Create the blob detector with the set parameters
    detector = cv2.SimpleBlobDetector_create(params)

    # Detect blobs in the image
    keypoints = detector.detect(im)
```

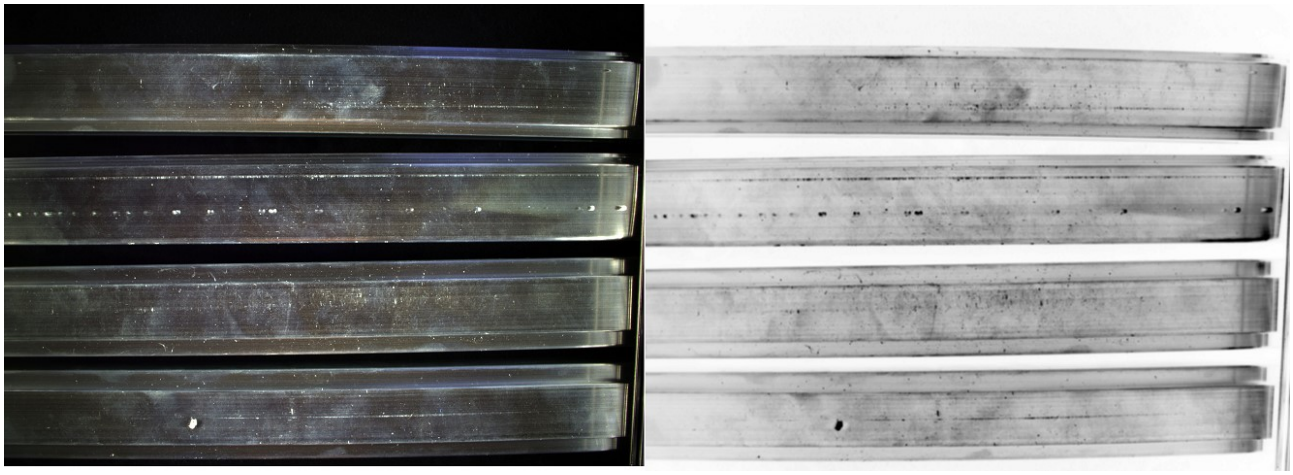
Kuva 23. Blob detector -parametrit (Oksa, 2023).

Tunnistukset piirretään kuvaan, jotta voidaan tarkastella, ovatko tunnistukset toimivia. Kuvaan myös asetetaan tunnistuksien määrä, ja kuinka monta kuvaa sekunnissa ohjelma pystyy käsittelemään. Ennen kuvien näyttämistä tulostetaan ohjelman kiertoaika vähentämällä tämän hetkisestä ajasta kierronaloitusaika, ja tulostetaan kiertoaika tunnistuksien määrän kanssa. Lopuksi voidaan tulostaa kuvat (kuva 24).

```
    # Draw detected blobs as red circles.
    # cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS ensures the size of the circle corresponds to the size of blob
    im_with_keypoints = cv2.drawKeypoints(img, keypoints, np.array([]), (0,0,255), cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)

    # Text settings for displaying the number of keypoints detected
    # font
    font = cv2.FONT_HERSHEY_SIMPLEX
    # org
    org = (50, 50)
    # fontScale
    fontScale = 1
    # Blue color in BGR
    color = (0, 0, 255)
    # Line thickness of 2 px
    thickness = 2

    keypointteja = len(keypoints)
    # Display the number of keypoints detected on the image
    im_with_keypoints = cv2.putText(im_with_keypoints, (f'Keypointteja {keypointteja} kpl'), org, font,
                        fontScale, color, thickness, cv2.LINE_AA)
    print(keypointteja," key pointteja")
    print("--- %s seconds ---" % (time.time() - start_time))

    # Show the image with keypoints
    cv2.imshow('im_with_keypoints_and_lines', im_with_keypoints)
    cv2.imshow('bitwise', im4)
    cv2.imshow('gray', gray)


    # Wait for the user to press the escape key to exit the loop
    k = cv2.waitKey(1) & 0xFF
    if k == 27:
        break
```

Kuva 24. Osumien piirtäminen ja kuvan tulostus (Oksa, 2023).

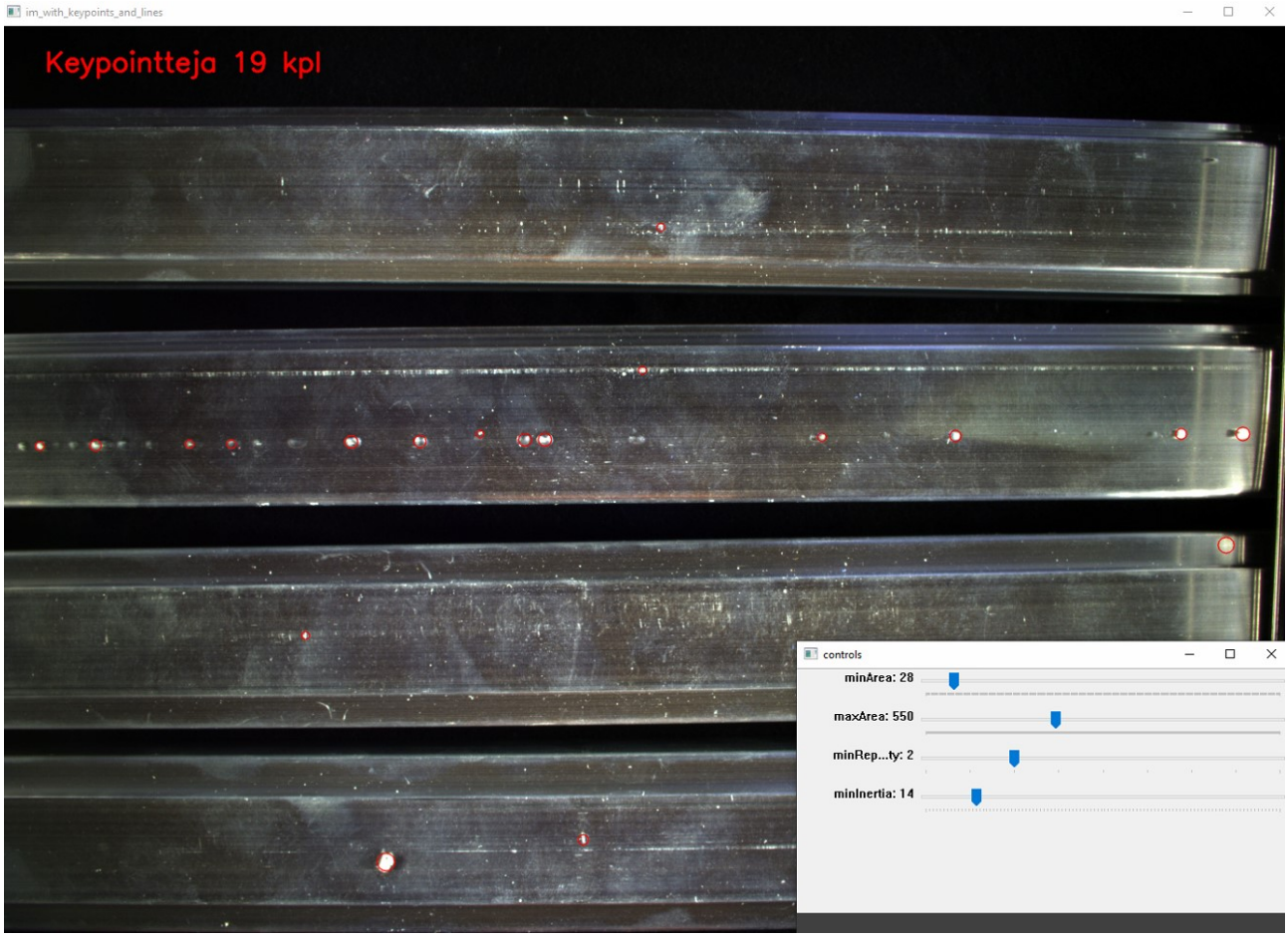
Kuvassa 25 nähdään alkuperäinen kuva ja vieressä käsitelty kuva, josta lähdettiin Blob detectorilla tunnistuksia hakemaan.



Kuva 25. Alkuperäinen kuva ja käsitelty kuva (Oksa, 2023).

Kuvassa 26 näkyy selvästi, että Blob detector on löytänyt ison osan ilmakuplista, mutta myös muutama ylimääräinen osuma on tullut luultavasti profiilin naarmuista. Liukusäätimillä oli helppo etsiä sopivat parametriarvot. Nämä parametrit voidaan tallentaa

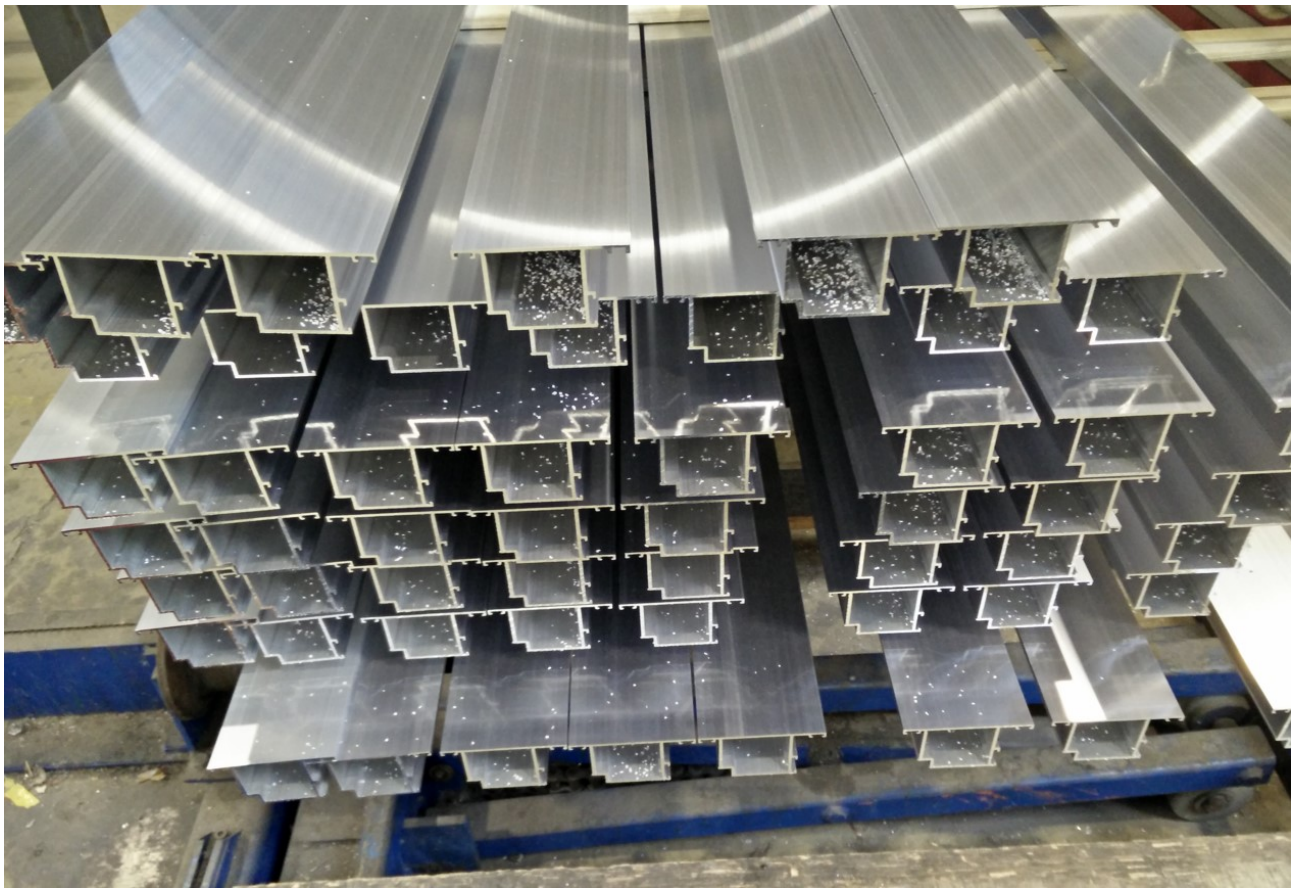
tietokantaan profiilin ID:n mukaan, jolloin ne voitaisiin tuoda takaisin kun samaa profiliia kuvataan uudestaan.



Kuva 26. Löydetyt ilmakuplat (Oksa, 2023).

4 KONENÄKÖJÄRJESTELMÄÄ RAKENNETTAESSA HUOMIOITAVAA

Alumiiniprofiilit tulevat varastolta leikattuna. Tämän leikkauksen myötä tulee alumiinisilppua, jota puhalletaan pois, mutta joskus profiilipinoon jää leikkaussilppua. Tämä silppu aiheuttaa varmasti ylimääräisiä tunnistuksia, joten se pitää saada poistettua. Koska kuvauspaikka tulisi olemaan mahdollisesti pakkaussolun sisällä, rajoittaa tämä kuvausmahdollisuuksia. Ylhäältä päin profiili saadaan kuvattua hyvin, mutta sivuilta ei ollenkaan, koska profiilit ovat toisissaan kiinni. Alapuolelta kuvaamista rajoittavat kapeat hihnakuljettimet, joita pitkin profiilit liikkuvat. Nämä hihnakuljettimet saattavat myös alhaaltapäin kuvattaessa tuottaa ongelmia, koska paras valaistus olisi pimeävalokenttä, mutta kuljettimet tulisivat luultavasti valaistuksen tielle. Kuljettimien heijastuvuus saattaa tuottaa ongelmia, jolloin ei saada tasalaatuisia kuvia. Kuljettimet myös peittävät kuljettimen leveyden verran profiilia. Kaikki kiiltävät osat olisi hyvä olla maalattuina mattamustalla, jotta vältytään turhilta heijastuksilta.



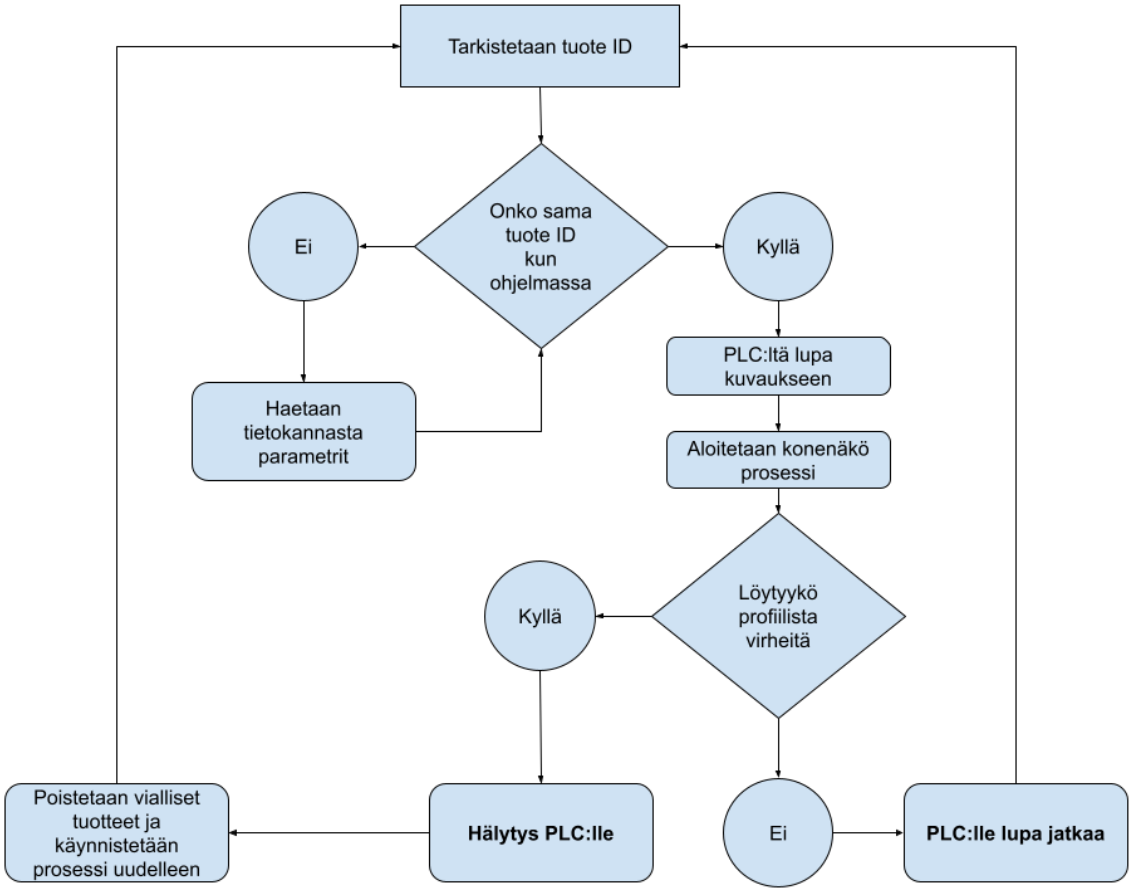
Kuva 27. Alumiinisilppua (Oksa, 2023).

4.1 Mallijärjestelmä

Järjestelmä voitaisiin rakentaa niin, että konenäkösovelluksen parametreille luodaan tietokanta, jonne kerätään profiilin ID mukaan parametrit. ID:t saadaan Mäkelän Alun omasta varastotietokannasta. Jos ID:lle ei löydy parametrejä, otetaan oletusparametreistä arvot, ja tämän jälkeen niitä muokataan tulosten mukaan sopiviksi. Viivakamera soveltuu tähän hyvin, koska profiilit ovat pitkulaisia ja viivakamera poistaa kuvavääristymää. Kuvausleveys olisi 400 mm, jolloin viivakameran pikseleiden määrän tulisi olla vähintään 2048, jotta 1 mm pintavirhettä kohden saadaan vähintään vaadittu 4 pikseliä. Valaistuksen on syytä olla riittävän voimakas, jolloin ulkoisten valojen häiriöt pysyvät mahdollisimman pieninä ja kameran valaistusaikaa voidaan lyhentää. Tämä mahdollistaa profiilien nopeamman kuvaamisen.

Kameraa pitää liikuttaa profiilien myötäisesti, jotta ne saadaan kuvattua kokonaan. Tällöin tarvitaan pulssianturi, moottori, taajuusmuuttaja sekä lineaarikisko, mihin kamera voidaan kiinnittää. Ohjelmoitavana logiikkana voi toimia esimerkiksi Twincat3 Beckhoffin PC:llä, jolloin konenäköohjelma voi pyöriä ohjelmoitavan logiikan tietokoneella ja kommunikaatio konenäköohjelman ja ohjelmoitavan logiikan välillä voi tapahtua esimerkiksi ADS-kommunikaation kautta. Kameraohjelmaparametrit voidaan hakea tietokannasta, jota päivitetään aina uuden tuotteen tullessa pakkauslinjalle.

Kuviossa 1 on esimerkki siitä, miten kameraohjelman logiikka voisi toimia. Konenäköohjelman parametriasetusten ID:tä verrataan, jonka mukaan päätellään haetaanko uudet parametrit. Kun parametrit ovat oikeat, odotetaan pakkaussolulta lupaa kuvaukseen. Kuvataan profiili ja tarkistetaan, löytyikö virheitä. Jos virheitä löytyy, hälytys laukeaa ja pakkaussolu tai käyttäjä tekee tarvittavat toimenpiteet. Jos virheitä ei löydy, pakkaussolu saa luvan jatkaa.



Kuvio 1. Kamerasovelluksen logiikkakaavio (Oksa, 2024).

5 TULOKSET

Profiileista löydettiin konenäköohjelman avulla iso osa ilmakuplista, mutta muutama ylimääräinenkin tunnistus tuli. Osa ilmakuplista jäi löytymättä, jos ne olivat loivia tai hyvin pieniä. Ilmakuplia voi siis lähteä etsimään Blob detectorin tai histogrammin avulla.

Konenäkösovelluksen logiikka saatiin suunniteltua, mutta laitteistoa ei voitu tarkasti suunnitella, koska pakkaussolusta ei ollut tarkkoja mekaniikkakuvia tai ohjelmarajapintoja.

Vaikka profiileista saatiin ilmakuplia ja pintavirheitä esiin, tarvitsee se kuitenkin paremman testauksen, jotta voidaan varmistua tunnistuksien luotettavuudesta.

6 JOHTOPÄÄTÖKSET JA POHDINTA

Konenäkösovelluksen suunnittelu oli äärimäisen haastavaa, koska työ oli pääasiassa teoreettinen ja vähäisen laitteiston takia testaus jäi pieneksi. Pakkaussolun tarkkojen teknisten tietojen puute hankaloitti myös suunnittelua. Koska profiileja ei voida opettaa konenäköohjelmalle suurien erilaisten tuotteiden määrien takia, perinteiset konenäkösovellukset eivät soveltuneet virheiden etsimiseen. Alumiini itsessään myös tuotti paljon hankaluuksia kuvauksissa, koska se heijastaa valoa hyvin. Profiilit myös naarmuuntuivat helposti, jonka takia pimeäkenttävalaistus myös korosti näitä naarmuja. Profiileissa oli myös pieniä sävyeroja, mutta syytä tähän sävyeroon ei löytynyt.

Kuvaukselle parempi paikka olisi profiilinpuristuksen jälkeen, koska profiili liikkuu pitkittäin, ja profiilien päitä ei tarvitse erikseen huomioida. Tällöin kamera voisi olla paikalla ja siihen pystyisi rakentamaan kuvauslaatikon.

Vaikka työ oli haastavaa, työssä opittiin todella paljon konenäöstä ja sen haasteista. Koska normaalit konenäköohjelmat eivät taipuneet pintavirheiden tunnistukseen, pakotti se myös tutkimaan uudenlaisia tunnistustyylejä. Positiivisena yllätyksenä tuli, kuinka helppokäyttöinen Python oli, vaikka aiempaa kokemusta siitä ei juurikaan ollut.

LÄHTEET

1st Vision. (18.9.2020). *Imaging basics: Calculating resolution and field of view.* 1st Vision Machine Vision Solutions. https://www.1stvision.com/machine-vision-solutions/2015/07/imaging-basics-calculating-resolution.html

Advanced Illumination. (2013). *A practical guide to machine vision lighting* (Version 4). https://www.advancedillumination.com/wp-content/uploads/2018/10/A-Practical-Guide-to-Machine-Vision-Lighting-v.-4-Generic.pdf

All About Circuits. (18.9.2020). *3D imaging: Heart of machine vision forges ahead*. All About Circuits. https://www.allaboutcircuits.com/news/3d-imaging-heart-of-machine-vision-forges-ahead/

Chouinard, J. (16.9.2015). *Imaging basics – Calculating lens focal length*. 1st Vision. https://www.1stvision.com/machine-vision-solutions/2015/09/imaging-basics-calculating-lens-focal.html

GeeksforGeeks. (13.6.2024). *Image thresholding techniques in computer vision*. GeeksforGeeks. https://www.geeksforgeeks.org/image-thresholding-techniques-in-computer-vision/

Intel. (i.a.). *What is machine vision?* https://www.intel.com/content/www/us/en/manufacturing/what-is-machine-vision.html

JAI. (i.a.). *Line scan cameras.* JAI. JAI | Color imaging in machine vision: how to choose the right camera…

Keyence. (i.a.). *Basic practices in machine vision*. Keyence. https://www.keyence.com/ss/products/vision/visionbasics/basic/practice02/

Labellerr. (22.12.2022) *What is image enhancement and how does it affect machine vision?* Labellerr. https://www.labellerr.com/blog/what-is-image-enhancement-and-how-does-it-affect-machine-vision/

LearnOpenCV. (i.a.). *Blob detection using OpenCV (Python/C++)*. LearnOpenCV. https://learnopencv.com/blob-detection-using-opencv-python-c/

Micro-Epsilon. (i.a.-a). *Laser profile scanners*. Micro-Epsilon. https://www.micro-epsilon.com/2d-3d-measurement/laser-profile-scanners/

Micro-Epsilon. (i.a.-b). Laser scanners for 2D/3D profile measurement | Micro-Epsilon Optronic GmbH (micro-optronic.com)

Mäkelä Alu. (i.a.). *Mäkelä Alu*. https://makelaalu.fi/

OpenCV. (i.a.). *Template Matching*. OpenCV Documentation. https://docs.opencv.org/4.x/d4/dc6/tutorial_py_template_matching.html

PetaPixel. (4.8.2021). *What is CCD CMOS sensor?* PetaPixel. https://petapixel.com/what-is-ccd-cmos-sensor/

Ripe Media. (28.3.2013). *Why I love pixels and so can you*. Ripe Media. https://www.ripe-media.com/why-i-love-pixels-and-so-can-you/

Robotics Tomorrow. (17.12.2019). *What is machine vision?* Robotics Tomorrow. https://www.roboticstomorrow.com/article/2019/12/what-is-machine-vision/14548

Sagacious Research. (i.a.). *CMOS image sensors: Evolution, patent trends, leading players & more*. Sagacious Research. https://sagaciousresearch.com/blog/cmos-image-sensors-evolution-patent-trends-leading-players-more/

SAMK Automaatio. (i.a.). *Harmaasävykuvaus*. https://automaatio.samk.fi/testi-sivu/perinteiset-konenakojarjestelmat/harmaasavykuvaus/

Stack Overflow. (i.a.). *Converted RGB image to HSV but still displaying RGB values*. Stack Overflow. https://stackoverflow.com/questions/38028407/converted-rgb-image-to-hsv-but-still-displaying-rgb-values

Universe Optics. (i.a.). *CCD vs CMOS – which is best?* https://www.universeoptics.com/ccd-vs-cmos-which-is-best/

VICO Imaging. (14.6.2024). *A beginner's guide to machine vision systems*. https://vicoimaging.com/a-beginners-guide-to-machine-vision-system/#Computing_platform

Vision Doctor. (i.a.-a). *Bayer colour interpolation*. Vision Doctor. https://www.vision-doctor.com/en/area-scan-cameras/single-chip-colour-cameras/bayer-colour-interpolation.html

Vision Doctor. (i.a.-b). *Line scan cameras*. Vision Doctor. https://www.vision-doctor.com/en/line-scan-cameras.html

Vision Doctor. (i.a.-c). *Camera*. Vision Doctor. https://www.vision-doctor.com/en/camera.html

Vision Doctor. (i.a.-d). *Coloured light and coloured objects in machine vision*. https://www.vision-doctor.com/en/coloured-light-coloured-objects.html

Zebra Technologies. (i.a.). *What is a machine vision camera?* Zebra Technologies. https://www.zebra.com/us/en/resource-library/faq/what-is-machine-vision-camera.html