VAASAN AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES

Anton Kondratev

KONENÄKÖJÄRJESTELMÄN KEHITTÄMINEN TEOLLISUUSROBOTILLE

Tekniikka

2018

VAASAN AMMATTIKORKEAKOULU

Kone- ja tuotantotekniikka

TIIVISTELMÄ

Tekijä	Anton Kondratev
Opinnäytetyön nimi	Konenäköjärjestelmän kehitys teollisuusrobotille
Vuosi	2018
Kieli	suomi
Sivumäärä	42 + 7 liitettä
Ohjaaja	Mika Billing

Työn taustana on robotiikan ja koneennäön kehitys ja optimointi hinnan osalta. Nykyiset, tuotantorobotiikassa käytössä olevat konenäön sovellukset, ovat erittäin kalliita eivätkä standardoituja.

Työssä yritetään kehittää riittävän yksinkertainen ohjelma, joka ottaa valokuvan tietystä alueesta sekä etsii valokuvasta tietynmuotoiset kappaleet. Ohjelma tunnistaa valokuvassa olevat kappaleet, mittaa niiden väliset etäisyydet ja koordinaatit. Työssä käytetään C++-ohjelmointikieltä sekä ohjelmointikirjastoa OpenCV.

Tämän työn asettamat tavoitteet saavutettiin. Ohjelma ottaa valokuvan, purkaa matemaattisesti sen ja tunnistaa kuvassa olevat kappaleet. Tämä ohjelma voidaan kehittää monimutkaisemmaksi, esimerkiksi tunnistamaan vaikeamman muotoiset kappaleet tai kasvattamaan tunnistettavien kappaleiden määrää. Opinnäytetyön ohjelmaa voidaan soveltaa sellaisenaan tuotannossa, mikäli kyse on yksinkertaisemman muotoisista kappaleista.

Avainsanat konenäkö, robotiikka, ohjelmointi

UNIVERSITY OF APPLIED SCIENCES

Kone- ja tuotantotekniikka

ABSTRACT

Author	Anton Kondratev
Title	Development of Computer Vision for Industrial Robotics
Year	2018
Language	Finnish
Pages	42 + 7 Appendices
Name of Supervisor	Mika Billing

The purpose of this thesis was to develop more affordable and more flexible applications of the “Machine learning and vision” technology.

Nowadays all the computer vision applications being used in robotics are very expensive with no flexibility included. They are developed only for certain purpose and for certain conditions.

The application developed as the main objective of the thesis takes a photo of a certain area, where two certain objects are situated. By analyzing the photo, the application determines the type of each object and their position. The C++ programming language was used and the computer vision library OpenCV for C++.

The main purpose of this thesis was attained. The application looks for objects and measures them. This application could be developed further, for example, it could look for more complicated objects.

Keywords Computer vision, robotics and programming

Glossary

DIY(Do It Yourself) – tee se itse. Lyhenne, joka kuvailee asioita, jotka käyttäjä voi suorittaa itse ilman asiantuntijoiden opastusta.

Rpi – raspberry pi

SISÄLLYS

LIITELUETTELO

1 JOHDANTO

Opinnäytetyön tarkoitus on luoda ohjelma, joka sisältää keskeiset koneennäön algoritmit, selittää niiden toimintatavat ja antaa opiskelijoille mahdollisuuden kehittää tätä projektia edelleen. Kuten tiivistelmässä mainitaan, ohjelma osaa ottaa valokuvat ja tunnistaa valokuvissa olevat kappaleet.

1.1 C++

C-ohjelmointikielen loi Bjarne Stroustrup vuonna 1979. Alussa tämä kieli oli tavallinen lineaarinen ohjelmointikieli, mutta pian sen syntymisen jälkeen Stroustrup aloitti sen kehittämistä objektiseksi kieleksi. Nykyään kielen objektisen version nimi on C++ ja tämä versio on suosituin. Tämä on ”yleis”-ohjelmointikieli, joka on tarkoitettu erilaisten alojen ongelmien ratkaisuun ja ympäristöön, jossa on rajoitetusti resursseja.

1.2 Raspberry Pi

Raspberry Pi on itsenäinen tietokone, mutta sen koko ja tarkoitus eroavat tavallisesta kotitietokoneesta. Raspberry Pi sisältää samat komponentit kuin tavallinen kannettava tietokone tai kotikone. Tässä työssä käytetään Raspberry Pi 3 model B, joka on tällä hetkellä viimeisin versio. Viimeisintä versiota käytetään siksi, että tarvitaan kohtalaisen paljon nopeutta. Yhteys tähän tietokoneeseen saadaan sisäänrakennetun Wifi- tai Ethernet-teknologian avulla. Mitään varsinaista näyttöä ei tarvita, paitsi asennettaessa käyttöjärjestelmää. Käyttöjärjestelmänä tietokoneessa on Raspbian Jessie.

Kuva 1. Raspberry Pi

1.3 OpenCV

OpenCV (Open Source Computer Vision Library) on avoin konenäköön ja koneen oppimiseen tarkoitettu ohjelmointikirjasto. OpenCV on kehitetty tarjoamaan yhteisen rakenteen standardisoinnin konenäön sovelluksia varten. Tämä kirjasto jaetaan BSD-lisenssillä. OpenCV-kirjastossa on yli 2 500 konenäköön ja koneen oppimiseen tarkoitettua algoritmia. Näitä algoritmeja voidaan käyttää kasvojen tunnistamiseen ja määrittämiseen, ihmisen toimintojen luokittelemiseen videonauhoituksissa, liikkuvien objektien seuraamiseen, valokuviin pohjautuvien 3D-mallien rakentamiseen jne. Tämä ohjelmointikirjasto on runsaassa käytössä yrityksissä, tutkimusryhmissä ja valtioissa. Se sisältää C++, -C, -Java, -Python ja Matlab-kielten interfeisit ja tukee Windows-, Linux-, Android- ja Mac OS-käyttöjärjestelmiä /1/.

1.4 Konenäkö

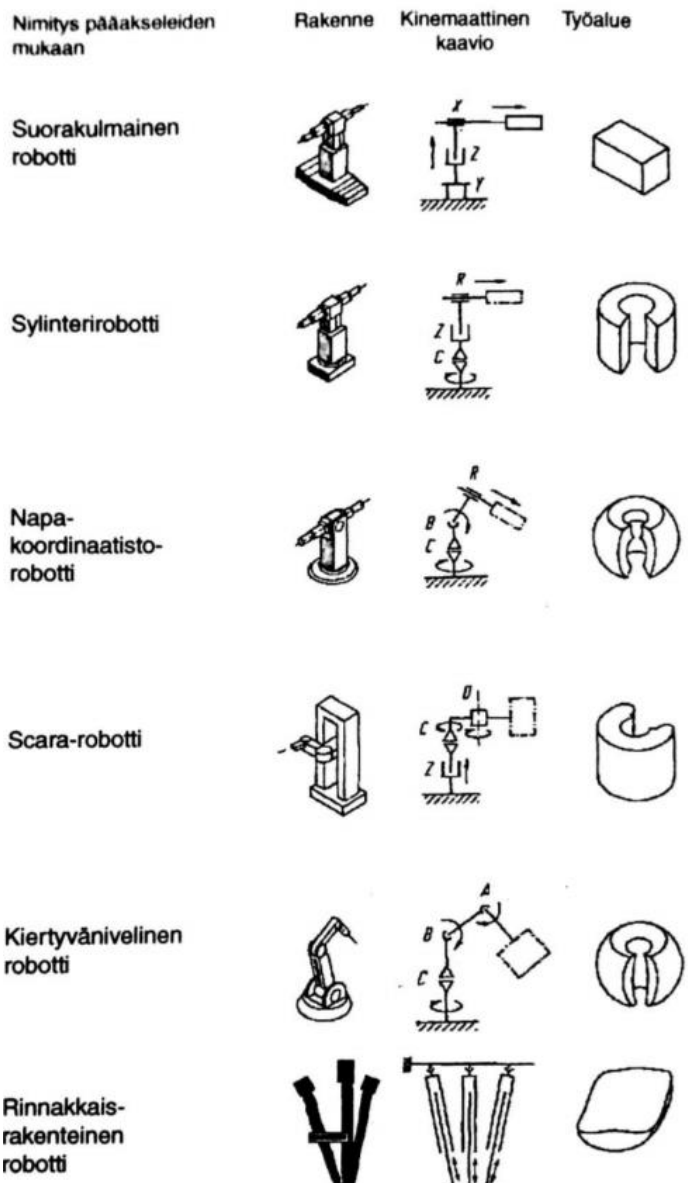
Tässä työssä keskitytään pääasiassa konenäköön ja koneiden oppimisalgoritmit jätetään sivuun, koska ne eivät kuulu aiheeseen. Työssä olevat algoritmit eivät käytä oppimisalgoritmeja.

Konenäön sovellukset ovat mielenkiintoisia ja laajaan käyttöalueeseen sopivia, mm. tieliikenteen sovelluksiin (nopeuskamerat), turvajärjestelmiin, tilastoihin (väen laskeminen esim. rautatieasemalla) ja lääketieteeseen (röntgenkuvien analysointi). Tämä on lisäksi ajankohtainen ja riittävän luotettava teknologia. Teknologia vaatii kuitenkin aika paljon resursseja ja sen seurauksena laadukasta ja kallista laitteistoa luotettavaan toimintaan.

1.5 Robotiikka

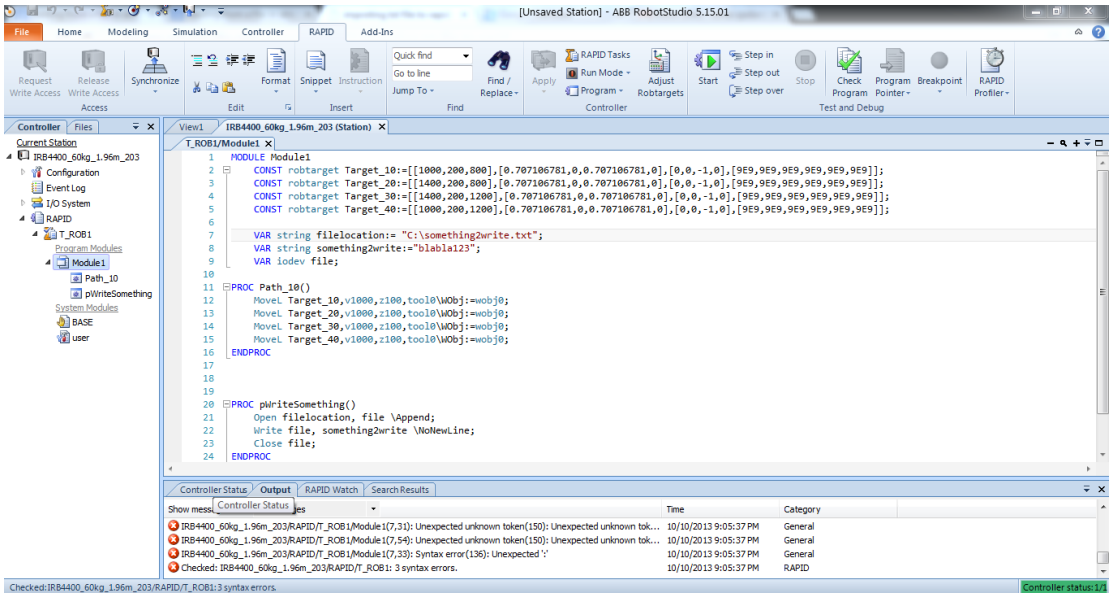
Robotiikka on erittäin hyödyllinen teknologia, joka mahdollistaa tuotannon kustannusten vähentämisen. Robotit ovat tarkkoja, nopeita eivätkä vaadi palkkoja, lomia sekä palkkojen korotuksia. Robottijärjestelmät käyttävät C++-kaltasta ohjelmointikieltä RAPID (ABB Robotit), joka on hyvin tuttu ohjelmointia osaavalle henkilölle. Järjestelmän etuna on myös se, että robotit toiminnassaan

voivat suorittaa vaikeita matemaattisia laskutoimituksia. Hyvänä ominaisuutena on myös robottien välinen kommunikointi. Robotiikkajärjestelmän voi ohjelmoida niin, että tuotannossa olevat robotit kommunikoivat keskenään tai lähettävät esimerkiksi tilastot tiettyyn tietokoneeseen. Robotti määritellään kansainvälisen yhdestyksen mukaan (**Kuva 2.**) laitteeksi, jossa on vähintään kolme liikkuvaa niveltä ja se voidaan ohjelmoida uudelleen /2/.



Kuva 2. Robottien tyypit /2/

Robotin toimintoja voi ohjelmoida usealla eri tavalla. Helpoin tapa on käyttää RobotStudio - ohjelmaeditoria, joka ulkonäöltään muistuttaa tavallista ohjelmointikielen editoria (**Kuva 3.**).



Kuva 3. RobotStudio RAPID-editor

Toinen tapa ohjelmoida on käyttää käsiohjainta (**Kuva 4.**).

Kuva 4. Käsiohjain

Robotit kommunikoivat keskenään verkossa käyttämällä ”Socket”-periaatetta. Ohjelma luo soketin, asettaa sen ominaisuudet ja aloittaa keskustelun verkossa tai odottaa kunnes jokin muu palvelu yrittää tavoittaa sen.

Seuraava koodin esimerkki suorittaa yllä mainitut toiminnot. Tämä esimerkki on otettu ABB:n foruumista /6, s. 282/.

```
Module Socket

VAR socketdev socket1;
VAR socketstatus state;

PROC main ( )

SocketCreate socket1;
SocketConnect socket1, "192.168.0.1", 1025;
!Communication
SocketSend socket1 Str:=received_string;
state := SocketGetStatus( socket1 );

Endproc

ENDMODULE
```

Tässä projektissa on myös käytetty tätä teknologiaa, jotta robotti voi kommunikoida Rpi:n kanssa. Rpi luo soketin, asettaa sen ominaisuudet ja siirtyy valmiustilaan odottamaan signaalia robottijärjestelmältä. Tätä vaihetta käsitellään myöhemmin. Tärkentä on mainita, että tämän projektin rajoituksena on se, että sekä robotin että Rpi:n täytyy olla samassa verkossa. Jos tämä ei ole mahdollista, paikallinen verkon administraattori voi järjestää systeemin toiminnan. Mahdollisesti Rpi:n ohjelma voidaan muuttaa universaalisemmaksi.

2 KONENÄÖN PERIAATTEET

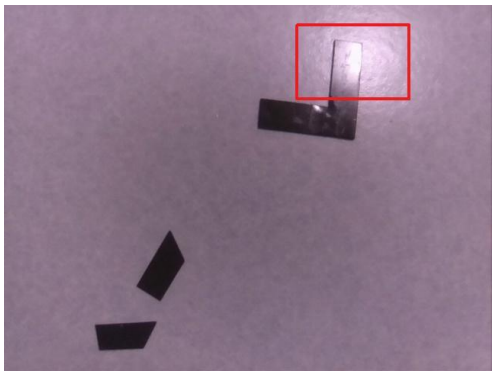
2.1 Konenäön ongelmat

Konenäön algoritmit pyrkivät vastaamaan ihmisen näköjärjestelmää. Konenäön algoritmien päätavoite on valokuvien ja videoiden sisällön ymmärtäminen. Katsoessaan ihmisen silmä huomioi seuraavat seikat: väri, kirkkaus, muodot jne. ja tallentaa ne jossain muodossa. Nähdessä jonkin objektin, esimerkiksi auton, ihminen tunnistaa tämän objektin autoksi, ajattelematta sitä lainkaan. Jos tämä tunnistettava objekti siirrettäisiin kauemmas tai käännettäisiin tai jopa muutettaisiin sen mitat, ihminen kuitenkin tunnistaa objektin meidän esimerkissämme autoksi. Tämä on jo ns. segmentoinnin tehtävä, eli objektin luokitteleminen. Tietokoneilla tätä tunnistuskykyä ei ole. Pitää ottaa huomioon se, että koneet näkevät kuitenkin eri tavalla kuin ihmiset. Jo muutos valaistuksessa voi vaikuttaa kohtalokkaasti tunnistamiseen. Valaistuksesta pitää kertoa lyhyesti erikseen. Tämä ominaisuus on mielestäni yksi tärkeimmistä. Valaistus pitää säätää hyvin tarkasti. Sitä ei saa olla liian paljon eikä liian vähän. Eri ympäristössä kone näkee objektit eri lailla. Kuvassa 5 voidaan nähdä niukan valaistuksen esimerkki. Kokemus osoittaa, että valaistuksen tulee olla hajanainen, eikä pistemmäinen.

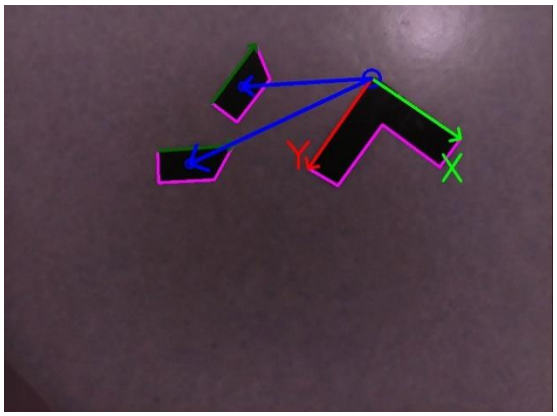
Pistemmäinen valaistus voi myös aiheuttaa tunnistamisongelmia. Tällaisessa valaistuksessa tunnistettavan objektin muoto katoaa. Kuva 6 kuvailee tätä tilannetta. Kuvassa 6 esitetty kohde ei mene oikein binarisointivaiheen läpi[1].

[1] Binarisointi – mustavalkoisen (grayshade) kuvan värien muuttaminen oikeaksi ”mustavalkoiseksi”, jossa mustaa väriä vastaava arvo on 0 ja valkoista väriä vastaava arvo on 255. Tästä kerrotaan tulevissa luvuissa.

Kuva 5. Niukasti valaistus



Kuva 6. Pistemmäinen valaistus



Kuva 7. Hyvä valaistus

Kuva 7 esittää riittävän hyvän valaistuksen esimerkin. Kuvien 5 ja 6 välinen ero on selkeästi huomattavissa. Kuvassa 6 oleva kirkas valkoinen piste katoaa ja objektien tunnistaminen helpottuu.

2.2 Tunnistaminen

Ihmiset kykenevät muistamaan objekteja perustuen objektien muotoihin, mittoihin ja mikä on tärkein, tärkeisiin objektin piirteisiin. Yksinkertaisemmin erilaisilla objekteilla on ainutlaatuiset piirteet, jotka kuvaavat sen tyypin, ja joiden avulla voidaan tunnistaa se yksiselitteisesti. Meidän aivot määrittävät analysoimansa objektin piirteet ja oppivat niiden avulla tunnistamaan objektin tulevaisuudessa. Samoin pyritään kehittämään konenäön algoritmit.

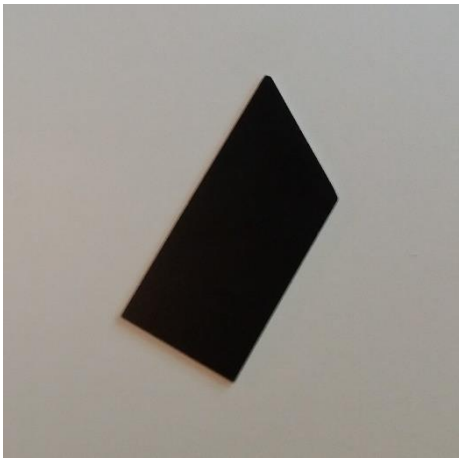
Kone poimii kuvasta mahdolliset piirteet ja tunnistaa objektin tai koneenoppimistapauksessa tallentaa nämä piirteet tulevaa käsittelyä varten. Yksinkertaiset geometriset muodot me ihmiset tunnistamme yleensä muodolla tai sivujen määrällä. Tässä työssä käytetään yksinkertaisia geometrisia objekteja, joten sovelletaan lähestymistapaa, jossa tunnistetaan perustuen yksinkertaisiin geometrisiin piirteisiin.

On olemassa muita tunnistamisalgoritmeja, esimerkiksi suosittu menetelmä ”SURF” (Speed-Up Robust /3, s. 545-551/ Features). SURF soveltaa toista tunnistamisperiaatetta, jossa se etsii kuvasta ”erikoiset” pisteet ja kerää tiedot näistä pisteistä. Kaikki SURF-menetelmän keräämät pisteet sekä niiden deskriptorit tallentuvat tiedostoon ja seuraavaa kuvaa analysoidessa ei tarvitse enää käynnistää algoritmia, ainoastaan saada tarvittavat tiedot tiedostosta. Tätä algoritmia käytetään yleisimmin autojen rekisterikilpien tunnistamiseen, kasvojen löytämiseen ja muiden haastavampien kohteiden löytämiseen.

3 TYÖN TOTEUTTAMINEN

3.1 Tehtävän vaatimukset

Vaatimuksia määriteltäessä sovittiin opettajan kanssa tunnistettavien kappaleiden tyypistä ja lopputuloksen muodosta. Tunnistettavaksi kappaleeksi valittiin kuvassa 8 esitetty kappale. Algoritmin tulokset on lähetettävä tietyssä muodossa tuotantorobotin ohjelmointi-interfeisiin käyttäen tietokoneverkkoa. Robotin ohjelmoija lähettää kyselyn verkon välityksellä tunnistuslaitteeseen, laite ottaa valokuvan, analysoi sen ja lähettää takaisin tulokset. Robotti vaatii tulokset muodossa ”[x-koordinaatti]:[y-koordinaatti]:[kulma x:n suhteen]”. Sovittiin myös, että työssä käytetään Raspberry Pi-tietokonetta.



Kuva 8. Tunnistettavaksi valittu kappale

3.2 Suunnittelu

3.2.1 Raspberry Pi

Työtä varten hankittiin Raspberry Pi ja asennettiin Raspbian Linux-laitteeseen. Raspbian asentaminen on kuvailtu Raspberryn kotisivulla. Ensiksi ladataan kotisivulta Raspbian Linux ja sitten, jos tekee Linux-ympäristöstä, syötetään seuraava komento: ***dd bs=4M if=asennettava_tiedosto.img of=/dev/sd[X]***

Komennon selitys:

- dd – Linux-komento, joka tarkoittaa kopiointia bitin tarkkuudella
- bs – kopioitavan osamäärän koko (tässä 4 megatavua)
- if – lähde, eli mistä kopioidaan (tässä Raspbian järjestelmän tiedosto)
- of – tulo, eli mihin kopioidaan (/dev/sd[X] sd-kortin nimi Linux-muodossa).

Kun käyttöjärjestelmä on asennettu, voidaan ohjata Raspberrya toisesta tietokoneesta ”ssh”-ohjelman avulla. Ainut ehto on se, että pitää etukäteen tietää Raspberryn ip-osoite. Tässä ja jatkossa oletetaan, että kaikki toiminnot suoritetaan Linux-ympäristössä. Toisella kuin Raspberry-tietokoneella suoritetaan seuraava komento: ***ssh pi@[ip_osoite] (esim. ”pi@192.168.1.100”)***

Tässä ”pi” tarkoittaa etätietokoneen käyttäjänimeä, meidän tapauksessa Raspbian koneella käyttäjänimi on ”pi”. Verkko-osoite kirjoitetaan [ip_osoite] kohdan sijaan. Seuraavaksi käyttäjältä kysytään luottaako hän etäkoneeseen. Tähän kysymykseen pitää vastata ”yes” ja sen jälkeen syöttää salasana. Nyt raspbian-tietokone on valmis käytettäväksi ja seuraavaksi asennetaan OpenCV-kirjasto.

3.2.2 OpenCV-asennus

Saadaksemme OpenCV-kirjasto syötetään selaimen osoitekenttään ”https://opencv.org” ja siinä painetaan ”releases”-linkki. Avatusta listasta valitaan viimeisin versio. Käyttäjä voi valita, joko ladata valmiiksi tehdyt kirjastot tai ladata ne raakoina ohjelmakoodeina ja tehdä kirjastot itse. Meidän työ on riittävän helppo eikä vaadi mitään ylimääräisiä fuktioita, joten voidaan ladata valmiiksi tehdyt. Kirjastojen valmistaja suosittelee kuitenkin tekemään ne itse. Kirjastotiedostojen tekemistä ei käsitellä, koska prosessi on varsin monimutkainen eikä kuulu tämän työn aiheeseen. Kannattaa mainita vain se, että voidaan aina ladata viimeisin mahdollinen versio, joka sisältää aina edellisten versioiden funktiot.

Asennuksen aikana Linux-järjestelmällä toimiva kone kysyy mihin asennetaan valmiit kirjastot. Polku, johon sijoitetaan kaikki tiedostot pitää muistaa, koska ne pitää kytkeä C++ -ohjelmointiympäristöön.

3.2.3 Työympäristö

Varakkaammat toteuttajat voivat ostaa itselleen C++ -ohjelmointiympäristön, esimerkiksi Microsoft Visual Studion ja kätevästi ohjata koko ohjelmointiprosessia tällä ohjelmapaketilla. Tässä projektissa ei ole käytetty mitään varsinaista pakettia, vaan sovellettu Linux-järjestelmän perustyökaluja.

```
g++ $(pkg-config –libs –cflags opencv) –lraspicam –lraspicam_cv –o output
```

Tunnistusohjelma on kirjoitettu käyttäen tavallista Notepad-ohjelmaa ja kompilointi tehdään käyttäen Linux-komentoikkunaa. Komentoikkunan käyttö on helppoa kompiloidessa. Meillä on esimerkiksi main.cpp-tiedosto, joka sisältää ohjelman, silloin kompiloidaksemme pitää syöttää:

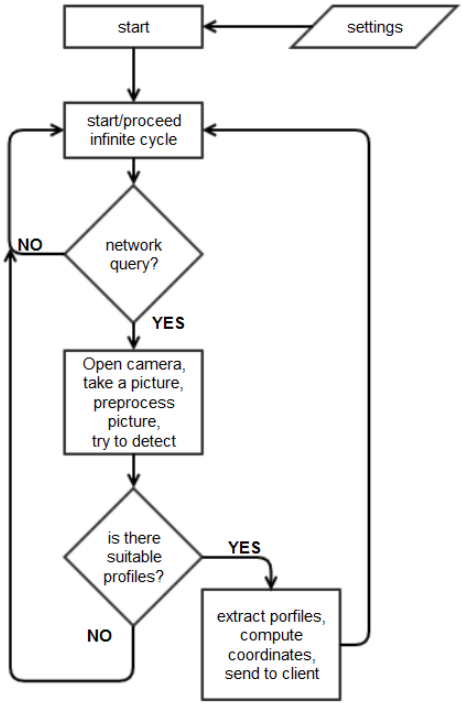
- g++ - Linux-järjestelmän C++-kielen käyttäjä
- $(pkg-config –libs –cflags opencv) – määrittelee muuttujan, johon tallentuu kaikka OpenCV:een liittyvät kirjastot
- -lraspicam –lraspicam_cv – kytketään lisää muut kirjastot
- -o [output] – käännetyn tiedoston nimi

Täytyy myöntää, että tällaisen rivin kirjoittaminen ei ole nopein eikä mukavin mahdollinen, mutta Linux-käyttäjät yleensä ovat tottuneita tähän. Kerran kirjoitettuna, tämän rivin voi toistaa vain painamalla näppäimistön nuolta ”ylös”.

3.2.4 Ohjelma

Tehtävänä on sijoittaa kameran näkyville yksi kuvassa 8 esitetty objekti, määritellä sen koordinaatit ja pyörimiskulma. Lisäksi tämän pitää toimia verkon välityksellä. Raspberry-koneessa oleva ohjelma toimii taustalla ja odottaa kyselyä tietyllä

portilla. Ohjelma pitää laatia joustavaksi niin, että käyttäjä voi määritellä itse ohjelmakohtaisia asetuksia mm. verkkoportin numero, ip-osoitteet ja muut (niitä käsittelemme myöhemmin). Jos kysely asiakkaalta on tullut, ohjelma ottaa kameralla valokuvan ja lataa sen ohjelmaan. Ohjelma kutsuu funktiot, jotka esivalmistavat kuvan prosessointiin, määrittelevät ja suodattavat mahdolliset ”tutut” muodot. Suodattimena käytetään kuvassa olevien suljettujen profiilien pinta-aloja. Eräänä ongelmana tunnistamisessa on myös se, että ohjelma voi löytää ylimääräisiä pieniä kohtia. Näiden kohtien ilmestymiseen voi pahasti vaikuttaa esimerkiksi huono valaistus. Nämä kohdat ovat yleensä pienikokoisia, joten voidan välttää ne pinta-alojen laskemisen avulla. Pinta-alat ilmaistaan aluksi pikseleissä. Jatkossa voidaan muuntaa ne senttimetriksi, joko laskemalla yhden pikselin mitta tietyllä kameran etäisyydellä tai määrittämällä jokin vertauskappale. Tässä projektissa käytetään vertauskappaletta, jonka mitat on määritelty etukäteen. Oikeat profiilit saatua, ohjelma suorittaa laskutoimitukset ja palauttaa tulokset verkon välityksellä asiakasohjelmalle (client application) muodossa: x_koordinaatti:y_koordinaatti:kulma. Kulma taas ilmaistaan vertauskappaleen



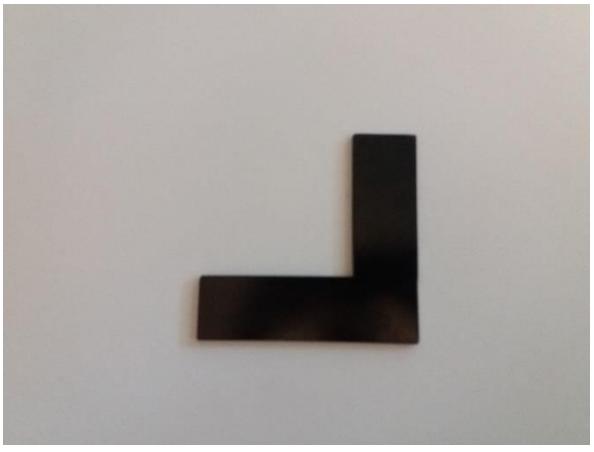
Kuva 9. Ohjelman toimintaperiaatekaava

mukaan. Kuvasta 9 näkyy kaikki ohjelman askeleet. Jokaisen tämän kaavan askeleen voi purkaa yksityiskohtaisemmin, mutta puretaan jokainen vaihe seuraavissa työn osissa.

3.3 Ohjelman esiasetukset

3.3.1 Muuttujatyypit

Kannattaa aluksi mainita lyhyesti muuttujien tyypit. Ohjelmointikielessä C++ käytetään muuttujien tyyppeja. Jos esimerkiksi halutaan käyttää desimaalityyppistä muuttujaa, se määritellään float, double-tyyppiseksi. Kirjastossa OpenCV on taas omia tyyppejä. Ensimmäinen ja tärkein näistä on tyyppi ”Mat”. ”Mat” esittää matriisin ja sitä määritettäessä täytyy osoittaa matriisin koot, esim: Mat matx = Mat::zeros (2,1,CV_32FC1). Tämä määritelmä luo matrisiin, jonka kaikki arvot ovat nolla (zeros). Matriisin koko on 2 riviä ja 1 sarake. Merkintä ”CV_32FC1” tarkoittaa, että matriisi tulee käyttämään 32-bittisiä (32) ”float”-tyyppisiä (F) lukuja ja matriisi on yksikerroksinen. Tässä tyypissä oletetaan, että matriisiin ladataan jokin kuva. Värillinen kuva koostuu kolmesta kerroksesta R-kerros (RED), G-kerros (GREEN) ja B-kerros (BLUE). Jokainen kerros sisältää kyseisen värin arvot 0-255, jotka esittävät tämän värin sävyt. Jos käyttäjä, kuten meidän esimerkissämme, käyttää yksikerroksista matriisia, tämä tarkoittaa, että kuva on mustavalkoinen ja sisältää vain mustan sävyt. Seuraava tyyppi on Scalar. Tällainen muuttuja sallii neljän eri lukujen säilyttämisen. Vectortyyppiset muuttujat sisältävät matemaattiset pystyvektorit. Vector-muuttuja toimii Mat-muuttujan periaatteella tarkoittaen, että määritellessä pitää osoittaa sisällä olevien arvojen tyypit. Vectorin esimerkki on: vector<float> m.



Kuva 10. Vertailukappale

3.3.2 Ohjelman rakenne

Suunnitteluosassa mainittiin vertailukappale. Vertailukappale on hyödyllistä ottaa käyttöön, sillä käyttäjän ei tarvitse kalibroida koko järjestelmää kameran siirtäessä ylösalaisin. Vertailukappaleeksi otettiin kuvan 10 mukainen muoto, koska se esittää kätevästi x- a y-akselit. Tämän vertailukappaleen korkeus ja leveys on ennältä mitattu ja tallennettu tiedostoon ”settings.json”. Myös tämän kappaleen etuna on se, että käyttäjä voi vaihtaa tämän toiseksi samanmuotoiseksi kappaleeksi, mitata sen ja kirjoittaa ”settings”-tiedostoon mitat ja ohjelma jatkaa tavallista toimintaa ilman muutoksia. Tutkitaan ”settings.json”-tiedosto tarvittaessa.

```
{
    "axis_x_mm" : 132.20,
    "axis_y_mm" : 137.90,
```

Tässä on tiedoston ”settings.json” osa, johon vertailukappaleen mitat ovat tallennettu.

Ohjelman kirjoittaminen aloitetaan kytkemällä kaikki tarvittavat C++ -kielen kirjastot.

```
#include <iostream>

#include <stdio.h>
#include <sys/types.h>
#include <sstream>
#include <string>
#include <sys/socket.h>
#include <opencv2/core.hpp>
#include <opencv2/imgcodecs.hpp>
#include <opencv2/imgproc.hpp>
#include <opencv2/highgui.hpp>
#include <opencv2/opencv.hpp>
#include <netinet/in.h>
#include <raspicam/raspicam_cv.h>
#include <unistd.h>
#include <jsoncpp/json/json.h>
#include <fstream>
```

Katsotaan tärkeimpien kirjastojen vastuualueet:

- iostream – C++-kielen peruskirjasto, joka vastaa näyttämisestä käyttäjälle tiedot ja syöttämällä erilaiset arvot

- stdio – tämän avulla voidaan muokata tekstitietoa käyttäjälle esittämiseen
- sys/soket.h – kirjasto, joka mahdollistaa ohjelman verkkotoimintaa
- opencv2/core.hpp – sisältää konenäön kirjaston perusfunktiot
- opencv2/imgcodecs.hpp – tähän kirjastoon on tallennettu kaikki funktiot, jotka osaavat käsitellä kuvat
- opencv2/imgproc.hpp – sisältää kuvien käsittelyn laskennalliset funktiot mm. geometriset muunnokset, suodatukset jne.
- opencv2/highgui.hpp – meidän projektissa tätä ei tarvita, joten jätettiin tulevaa kehitystä varten. Tässä voidaan luoda ja säätää erilaiset käyttöjärjestelmän ikkunat
- opencv2/opencv.hpp – sisältää kaikki perusfunktiot ja matriisilaskentafunktiot
- raspicam/raspicam_cv.h – kirjasto, jonka funktioiden avulla voidaan säätää Rpi-kamera
- jsoncpp/json/json.h – tästä kirjastosta otetaan funktiot, jotka tarvitaan ”settings.json”-tiedostoa varten.

Kirjastojen liittämisen jälkeen alustetaan kaikki päämuuttujat. Ensiksi määritetään JSON-muuttuja, joka vastaa ”settings.json” tiedostosta:

```
Json::Value settings;
```

Haetaan heti ”settings.json”-tiedostosta vertailukappaleen mitat:

```
ifstream ifs("settings.json", ifstream::binary);
ifs >> settings;
ifs.close();

float X_AXIS_LENGTH = settings["axis_x_mm"].asFloat();
float Y_AXIS_LENGTH = settings["axis_y_mm"].asFloat();
```

Nyt muuttujat X_AXIS_LENGTH ja Y_AXIS_LENGTH sisältävät vertailukappaleen leveyden ja korkeuden. Oman mukavuuden vuoksi määritetään:

```
Scalar green(0,255,0), blue(255,0,0), red(0,0,255);
```

Tämä rivi antaa mahdollisuuden kirjoittaa ”green”, ”blue”, ”red” sen sijaan, että joka kerta kirjoitettaisiin värin arvot. Viimeisenä esitetään C++-kielen rakenteen muuttuja:

```
struct proc_data

{
    float x;
    float y;
    float angle;
};
```

Muuttujassa ”proc_data” säilytetään vertailukappaleen pysty- ja vaakareunan tiedot, sekä etsittävän kappaleen keskipiste ja kulma. Tärkein muuttuja on Mat-tyyppinen ”img”. Tähän muuttujaan tallennetaan kameralta tullut valokuva.

3.4 Ohjelman funktiot

Kielen C++ säännöt vaativat sen, että ohjelmassa on aina oltava ”main”-niminen funktio, joten tässä ohjelmassa ensimmäisenä on ”main”-funktio. Tässä funktiossa suoritetaan kierros, joka odottaa kyselyä asiakasohjelmasta. Jos kysely on tullut, ”main” kutsuu muut apufunktiot.

Seuraava on ”raspi_cam()”-funktio, joka valmistaa Rpi:n kameran toimintaan ja ottaa valokuvan.

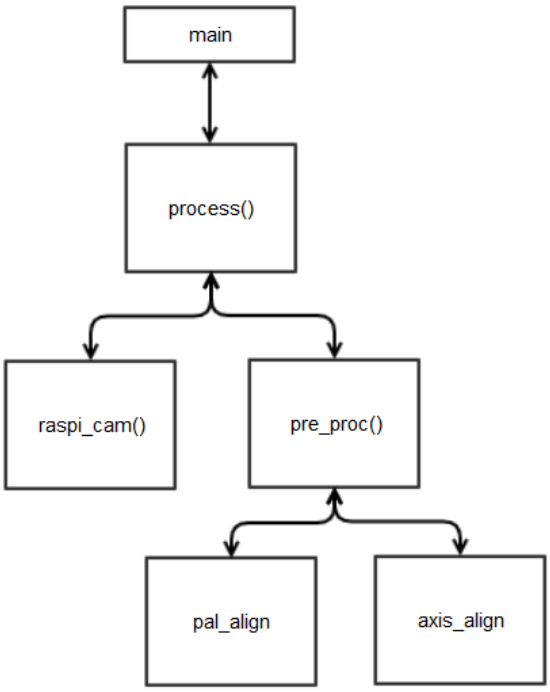
Funktio ”pre_proc (const Mat &img)” valmistaa kameralta tulleen kuvan. Ennen tunnistamista kuva pitää muuttaa mustavalkoiseksi, vähentää kuvan tausta, ottaa kuvasta ns. melu pois ja korostaa kaikki kuvassa olevat profiilit.

Kaikki funktiot välittävät jonkin arvon siihen, mistä funktio on kutsuttu. Täten jokaista funktiota varten pitää määrittää palautettavan arvon tyyppi. Se on samanlainen kuin muuttujien tyyppi. Funktion tyypit voivat olla ”void” (tyhjä), ”int” (kokonaisluku) jne. Joskus funktiot palauttavat muistin osoitteen, mistä palautettava arvo voidaan hakea. Tämä on hyödyllistä käyttää, jos haluaa palauttaa esimerkiksi taulukon. Seuraava funktio käyttää täsmälleen tätä periaatetta. Funktio ”float * process()” suorittaa matemaattiset laskutoimitukset, joiden tuloksena on

tekstirivi. Tekstirivi on C++-kielen mukaan taulukko, jota ei voida suoranaisesti palauttaa funktiosta. Ongelman ratkaisuna jouduimme palauttamaan osoite tietokonemuistista, johon tekstirivi on tallentunut. Tähti kertoo sen, että palautetaan muistin osoite, joka sisältää float-tyyppiset arvot.

Funktiot ”axis_align(vector<Point> crv)” ja ”pal_align(vector<Point> crv)” rakentavat matemaattiset vektorit löydetyistä vertailu- ja etsittävästä kappaleesta.

Kaikki yllä mainitut funktiot vuorovaikuttavat keskenään. Funktioiden yhteistyön kaava on esitetty kuvassa 11.



Kuva 11. Funktioiden yhteistyön kaava

3.4.1 Ohjelman pääfunktio ”main()”

Kielen C++ säännöt vaativat sen, että jokaisessa ohjelmassa pitää olla funktio nimeltään ”main()”. Tämä on kaikkien muiden funktioiden juuri. Tästä jokainen C++ ohjelma alkaa.

Projektissa tämä funktio valmistaa palvelimen toimintaa ja järjestää verkkotoiminnan. Kaikkien esivalmistuksien jälkeen silmukka käynnistyy ja palvelin alkaa kuuntelemaan verkkoportin numeroa 3425. Verkkoportin numeron voi vaihtaa tiedostossa ”settings.json”. Tämän portin läpi tullut kysely kutsuu funktion ”float * process()”. Funktio ”process()” palauttaa funktioon ”main” kappaleiden koordinaatit. Tämän jälkeen funktio ”main” lähettää nämä koordinaatit asiakasohjelmaan, palvelimen komentoriviin ja jatkaa kuuntelua.

Tässä päättötyön osassa esitetään koodin tärkeimmät osat. Kokonainen koodi sijoittuu työn liitteeseen.

```
0  while(1)
1      {
2          listener = accept(server, NULL, NULL);
3          if(listener < 0)
4          {
5              perror("accept");
6              exit(3);
7          }
8          bytes_read = recv(listener,buf,1024,0);
9          if(bytes_read <= 0) break;
10         cout << "Calling main processing function" << endl;
11         float *p = process();
12         stringstream s_d;
13         s_d << p[0] << ":" << p[1] <<":" << p[2];
14         //
15         char to_snd[100];
16         sprintf(to_snd,"%.3f:%.3f:%.3f", p[0], p[1], p[2]);
17         cout<<"==================================" << endl;
18         cout << to_snd << endl;//to_send << endl;
19         cout<<"==================================" << endl;
```

```
20          send(listener, to_snd, sizeof(to_snd),0);
21      }
22      close(listener);
23
```

Rivissä ”0” alkaa loputon silmukka, jossa palvelin kuuntelee verkosta tulleet kyselyt. Yhdeksäs rivi tarkistaa porttiin tulleet tiedot. Jos tavujen määrä on tyhjä katkeaa tämä iteraatio ja silmukka aloittaa alusta. Jos tavujen määrä on enemmän kuin nolla, funktio ”process()” kutsutaan. ”Processin” palauttamat tiedot sijoitetaan ”rivi”-tyyppiseen (String) muuttujaan (rivi 12) ja lähetetään asiakasohjelmaan (rivi 20).

3.4.2 Funktio ”raspi_cam()”

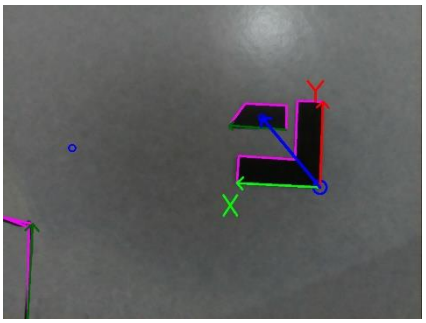
Ennen valokuvan ottamista järjestelmän on säädettävä kamera. Tästä vastaa funktio ”Mat raspi_cam()”. ”Mat” tässä tarkoittaa, että funktio palauttaa matriisityyppisen arvon. Tämä matriisi sisältää otetun valokuvan.

```
Mat raspi_cam()
0 {
1     raspicam::RaspiCam_Cv Camera;
2     Camera.set(CV_CAP_PROP_FORMAT, CV_8UC3);
3     if(!Camera.open()){cerr << "Camera is not working!" << endl;
exit(1);}
4     sleep(2);
5     Camera.grab();
6     Mat img;
7     Camera.retrieve(img);
8     Camera.release();
9     return img;
}
```

Työn suunnitteluvaiheessä käytettiin OpenCV:n sisäisiä funktioita kameran käyttämiseen, mutta selvisi, että OpenCV ei osaa säätää kameraa omilla toiminnoillaan. Ratkaisu löytyi eräältä espanialaiselta kehittäjältä /4/, joka kirjoitti C++ API-kirjaston Rpi:n kameraa varten. Toisen rivin komento säätää kameran ja määrittelee matriisin tyypin, johon tuleva kuva sijoittuu. Kameran avaamisen jälkeen tarvitaan jonkin aikaa, jotta siihen ehtisi tulla valo- ja väritiedot. Ellei tehdä tätä askelta, valokuva nähdään erittäin mustana.

Kuvissa 12 ja 13 on huomioitu mitä tapahtuu, jos ei anneta kameralle aikaa heräämiseen. Rivillä 4 oleva ”Sleep(2)”-toiminto pysäyttää ohjelman kahdeksi sekunniksi, jotta kamera heräisi kunnolla.

Kuva 12. Ilman ”Sleep()”-toimintoa

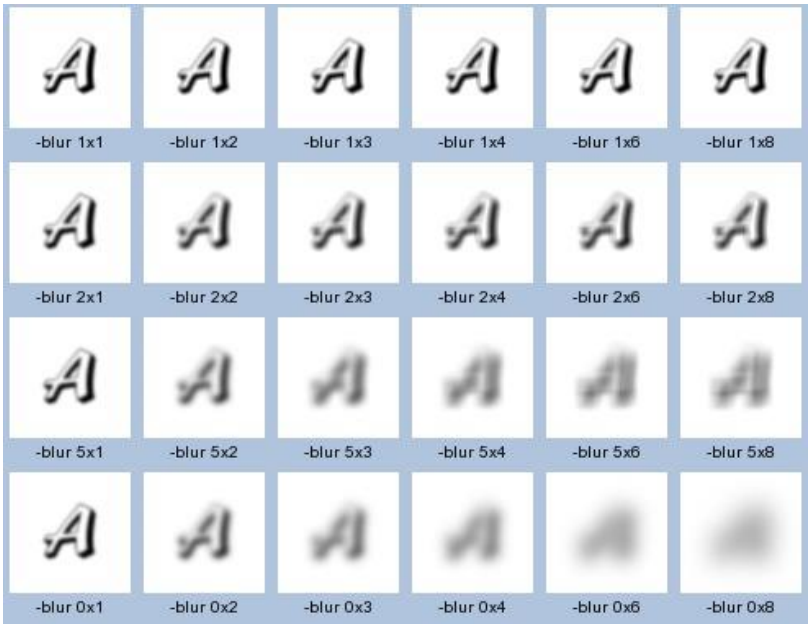


Kuva 13. ”Sleep()”-toiminnolla

3.4.3 Funktio ”pre_proc()”

Ennen objektien etsimistä valokuva tulee optimoida. Oikeissa olosuhteissa ohjelmoijan on määriteltävä tausta sekä tehtävä muutama taustakuva eri valaistusolosuhteissa. Taustakuvien avulla kehittäjä voi erottaa varsinaiset etsittävät objektit taustasta. Koska tässä työssä oletetaan, että taustana on aina jokin tasaisesti värjätty ja kiinteä taso, voidaan toteuttaa taustan erottaminen ilman erillisiä valokuvia. Graafisissa ohjelmissa on olemassa toiminto nimellä ”Blur”. Blur-toiminto hajaannuttaa kuvan piirteet, tekee kuvan epämääräiseksi.

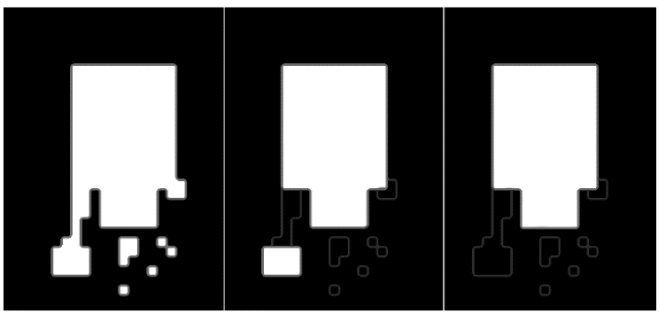
Blur-toiminto rakentaa erään matriisin ja käy koko kuvan läpi sovittaen matriisin arvot jokaiseen pikseliin. Ainut arvo, joka pitää tietää ennen Blurin soveltamista on matriisin koko. Jos syötetään matriisin kooksi riitävän iso arvo, katoavat kuvan piirteet, mutta kuvan tausta jää. Tätä menetelmää sovelletaan taustan määrittämiseksi. Otetaan matriisin kooksi kuvan kolmasosa vaakasuorassa ja kuvan kolmasosa pystysuorassa, sillä saadaan matriisi, joka kattaa melkein koko kuvan. Suoritetusta Blur-toiminnosta saadaan kaksi matriisia: toisessa on tavallinen valokuva ja toisessa on tämän kuvan tausta. Jos vähennetään taustakuvasta tavallinen kuva, saadaan tulokseksi kuvassa 18 olevat profiilit. Nämä profiilit ovat



Kuva 14. Blur-toimintaperiaate

jo hyvin erotettavissa ympäristöstä. Seuraavaksi tulos muutetaan mustavalkoiseksi. Mustavalkoinen kuva on matriisi, joka sisältää mustan värin arvot 0-255, jossa 0 tarkoittaa täysin musta ja 255 tarkoittaa valkoinen. Arvot tästä välistä ovat mustan vivahteita **(Kuva 19**.). Parhaan tuloksen saavuttamiseksi sovelletaan ns. binarisointiprosessi. Binarisointiprosessissa mustavalkoisen matriisin arvot muutetaan joko nollaksi tai 255:ksi. Tätä varten asetetaan raja, jonka ulkopuolella olevat arvot muuttuvat 255:ksi ja sisäpuolella 0:ksi. Tiedostossa ”settings.json” oleva ”threshold_bound”-arvo määrittelee tämän rajan. Tässä tapauksessa raja on 30. Toisin sanoen, matriisin arvot alle 30 muuttuvat nolliksi ja arvot enemmän kuin 30 muuttuvat 255:ksi (**Kuva 20**.).

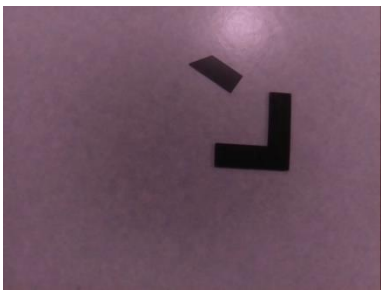
Kuvan esikäsittely ei pääty tähän. Valaistuksen epätäydellisyydestä binarisoituun kuvaan jää melko usein pieniä valkoisia pikseleitä, jotka voivat pilata lopputuloksen. Tässä projektissa on käytetty geometrisiä primitiiveja, joten pienet valkoiset pikselit edustavat ylimääräisiä tietoja, joita ei tarvita. Kirjastossa on funktiot, jotka auttavat voittamaan tämän ongelman. Funktioiden nimet ovat ”erode()” ja ”dilate()” /5, s. 276-280/. Yleensä näitä funktioita käytetään yhdessä ja niiden käyttöjärjestys riippuu halutusta tuloksesta. Tässä työssä käytetään erode() ensiksi ja dilate() toiseksi. Tällainen järjestys poistaa ylimääräiset pienet alueet kuvasta yhdistäen isommat profiilit. Toista järjestystä käytetään silloin kun tarvitaan, esimerkiksi kun lasketaan mikroskopilla tehdyissä kuvissa solujen määrä. Silloin on aika tärkeä erottaa jokainen pieni pikseli.



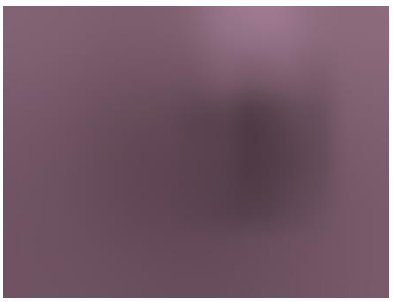
Kuva 15. erode-dilate algoritmi /6, s. 282/

Kuva 13 havainnollistaa sen, mitä erode-dilate-toiminnot saavuttavat. Kuva on jaettu kolmeen osaan. Ensimmäinen osa (vasen) näyttää lähtötilanteen. Toinen osa näyttää erode-toiminnon tulokset ja kolmas – dilate.

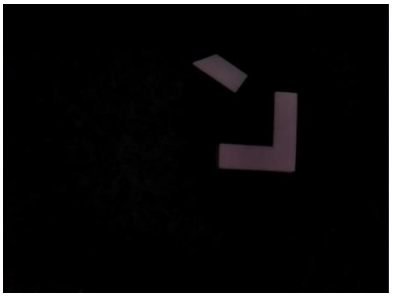
Lopuksi tulos pehmennetään jälleen kerran Blur-toiminnolla pienellä matriisikoolla ja palautetaan takaisin funktioon ”process()”. Funktion pre_proc() koodi on esitetty seuraavalla sivulla.



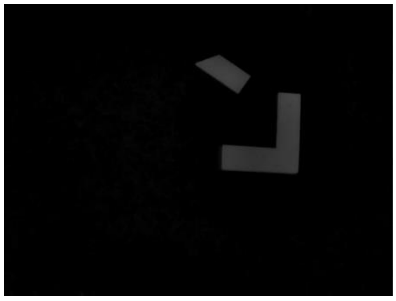
Kuva 16. Tavallinen kuva



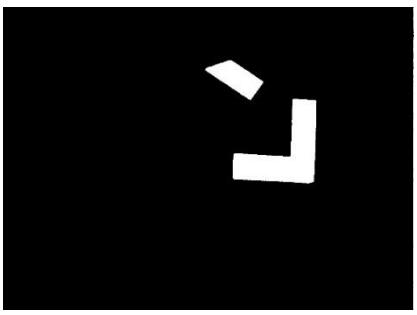
Kuva 17. Kuvan tausta



Kuva 18. Taustan vähennys



Kuva 19. Väritön kuva



Kuva 20. Binarisoinnin tulos

```
Mat pre_proc(const Mat &img)

{
    Mat im_proc = Mat::zeros(img.rows,img.cols,CV_8UC1);
    Mat bgr, test;
    blur(img, bgr,Size(img.rows/3,img.cols/3));
    im_proc = bgr-img;//background remove
    cvtColor(im_proc,im_proc,COLOR_BGR2GRAY);
    threshold(im_proc,im_proc,settings["thresh-
old_bound"].asUInt(),255,CV_THRESH_BINARY);
    int st_element = settings["structuredElement"].asUInt();
    Mat element = getStructuringElement(MORPH_ELLIPSE,Size(st_ele-
ment,st_element));//was 21 21
    erode(im_proc,im_proc,element);
    dilate(im_proc,im_proc,element);
    int blr = settings["blur"].asUInt();
    blur(im_proc,im_proc,Size(blr,blr));//was 5 5
    return im_proc;
}
```

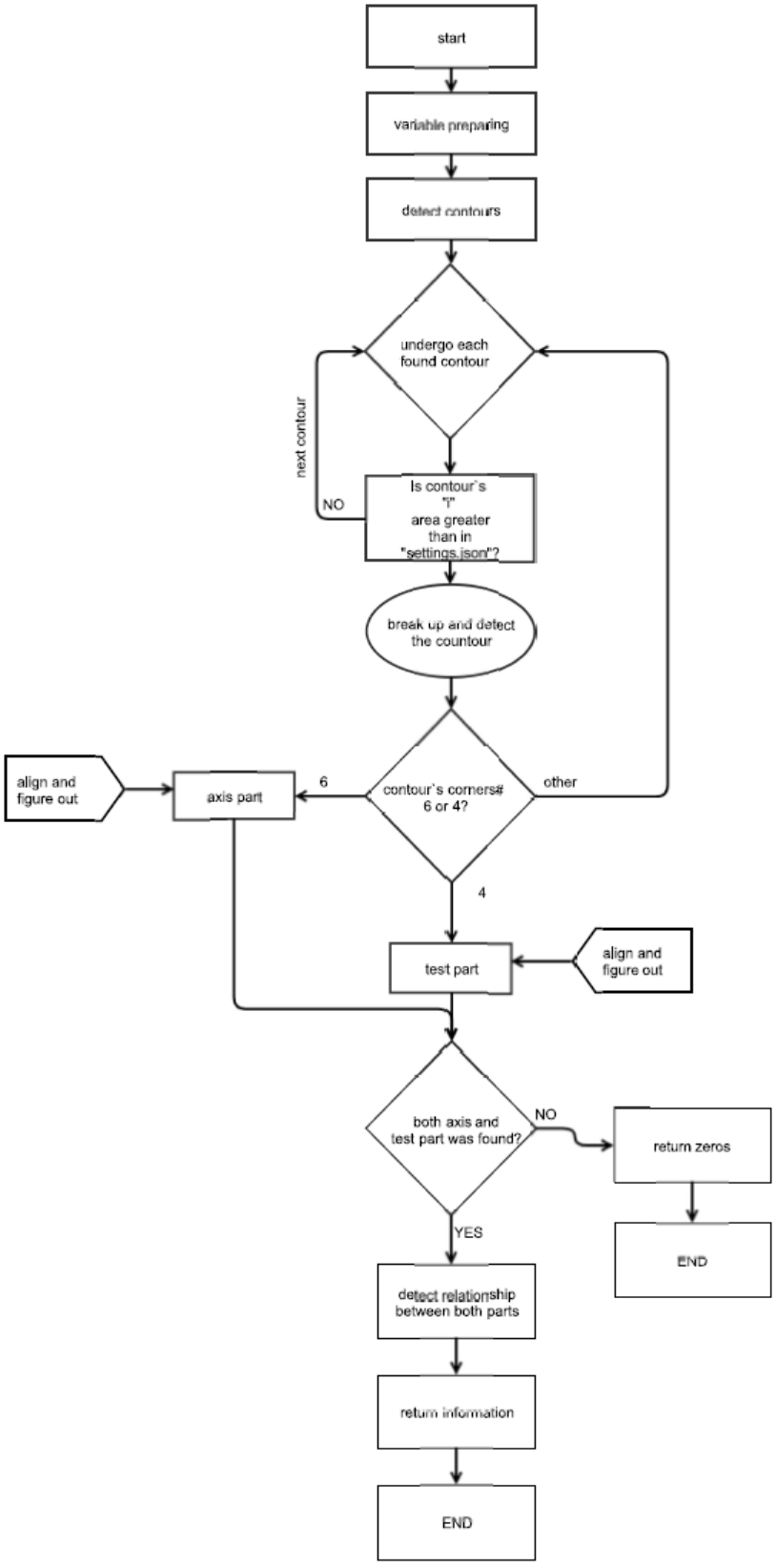
3.4.4 Funktio ”float * process()

Tämä funktio on koko ohjelman sydän. Tässä analysoidaan esivalmistettu kuva, erotetaan objektien profiilit toisistaan, sovelletaan lineaarialgebraa ja palautetaan tulokset funktioon ”main()”. Tässä kappaleessa ei esitetä koko funktion koodia, vaan tärkeimmät osat, koska se on liian iso. Koodi on työn liitteessä.

```
Point pal_center;
Mat x_ax = Mat::zeros(2,1,CV_32FC1); Mat y_ax = Mat::zeros(2,1,
    CV_32FC1); Mat obj_pal = Mat::zeros(2,1, CV_32FC1);//9,8,2017
Mat pal_ang = Mat::zeros(2,1,CV_32FC1);
proc_data obj, axis;
float pr_d[3];
float *pr_ptr;
pr_ptr = pr_d;

img = raspi_cam();
Mat im_proc = pre_proc(img);
vector<vector<Point> > features;
    findContours(im_proc,features,RETR_EXTERNAL,CV_CHAIN_AP-
PROX_SIMPLE);
cout << "contours found " << features.size() << endl;
bool ax, pal; ax = false; pal = false;
```

Rivit yllä ovat funktion alkuosa, jossa valmistetaan kaikki tarvittavat muuttujat ja suoritetaan profiilien etsintä. Ensimmäisen rivin Point-tyyppinen muuttuja ”pal_center” tulee sisältämään etsittävän kappaleen keskipisteen koordinaatit.



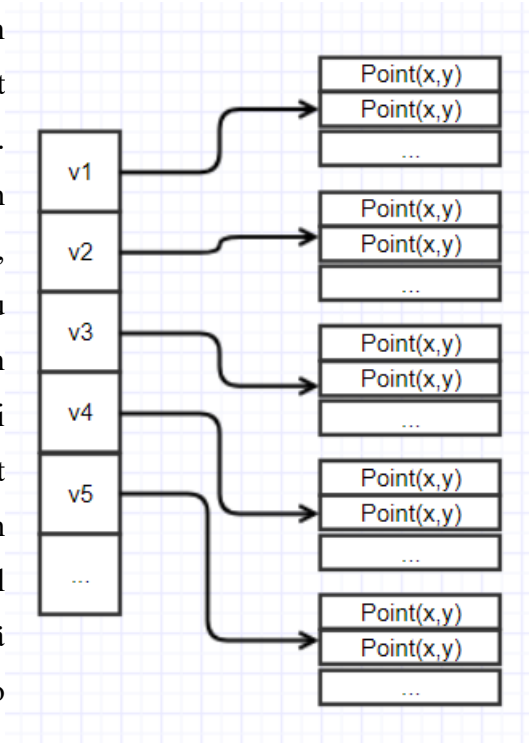
Kuva 21. Funktion ”process()” algoritmi

Tyyppi ”Point” on OpenCV:n sisältämä muuttujien tyyppi. Tämäntyyppisiin muuttujiin voi tallentaa kaksi arvoa. Point sopii mainiosti koordinaattien tallentamiseen. Rivillä 2 määritetään kolme matriisia, jotka matemaattisesti sanoen ovat vektoreita. Vektori on matriisi kooltaan Nx1 tai 1xN. Tällä hetkellä vektorit sisältävät vain nolla-arvot, mutta ohjelman edetessä ne täytetään arvoilla. Matriisi-tyyppinen vektori-muuttuja pal_ang saa testikappaleen kulman vektoriarvot. Rivi ”proc_data obj, axis” ilmoittaa muuttujat, joita käytetään ohjelman testia varten, nämä voi ottaa ohjelmasta pois.

Tämä funktio eroaa muista niin, että se palauttaa taulukon ”main()” fukntioon. Yleensä kielessä C++ ei ole olemassa tavallisia menetelmiä, jotka sallisivat taulukoiden palautuksen. Käytetään erästä menetelmää, jotta voitaisiin kuitenkin saada funktiosta taulukko. Tietokoneen muistiin muuttujat tallentuvat tietynkokoiseen soluun. Esimerkiksi kokonaisluvuntyyppinen (int, Integer) varaa itselleen 4 tavua. Tietäen sen, voidaan hakea muistin osoite, josta taulukko alkaa ja noutaa nelitavuisia muistin osia muuttujiin tai toiseen taulukkoon. Tätä lähestymistapaa hyödynnetään, jotta saadaan funktiosta taulukko ulos. Onneksi C++-kielen kehittäjät helpottivat tätä tilannetta. Sen sijaan, että laskettaisiin muistin soluja, voidaan ilmoittaa muuttuja, joka osoittaa tiettyyn muistin osoitteeseen ja käyttää tätä muuttujaa kontekstista riippuen. Meidän tapauksessa luodaan taulukkomuuttuja ”float pr_d[3]” ja heti sen jälkeen luodaan muuttuja ”float *pr_ptr”. Tähti edellisessä muuttujassa tarkoittaa sitä, että tämä muuttuja sisältää muistin osoitteen, josta alkaa float-tyyppisten arvojen jono tai taulukko. Ohjelman käydessä täytetään taulukko pr_d normaalisti ja funktion lopussa palautetaan osoitemuuttuja.

Määritetään muuttuja ”vector<vector<Point> > features”. Kuten aikaisemmin on mainittu, ”vector” on pystysuora matriisi kooltaan Nx1. Muuttuja ”features” ilmoitetaan siinä merkityksessä, että se sisältää vektorit, joissa ovat sisävektorit täytettynä Point-tyyppisillä arvoilla. Vektor-tyyppiset muuttujat kielellä C++ ovat ”loputtomia”, yksisarakkeisia taulukoita. Kirjaston OpenCV-funktio ”findContours” palauttaa kaikki arvot juuri vektortyyppiseen muuttujaan. /7, s. 410-413/ Jos kuvitellaan, että kuvassa on kaksi objektia, niin algoritmi sijoittaa

ensimmäisen löydetyn objektin koordinaatit ensimmäiseen ylempään vektoriin jokaisen pisteen koordinaatit sisävektorin muodossa. Toisen objektin koordinaatit sijoittuvat toiseen päävektoriin jne. Kuvassa 22 on visualisoitu tämän muuttujan rakenne. Täytyy huomioida, että funktio ”findContours” on tarkoitettu käytettäväksi vain ja ainoastaan binarisoitujen kuvien kanssa, eli parametriksi se ottaa vastaan vain matriisit tyyppiä CV_8UC1. Viimeiset funktion ”process” globaalit muuttujat ovat ”bool ax, pal”. Nämä muuttujat ovat tyyppiä ”bool”, mitkä voivat ottaa arvot joko ”tosi” tai ”epätosi”. Nämä muuttujat asetetaan tilaan ”tosi” silloin, kun kuvasta ilmestyy joko akseliosa tai testiosa.



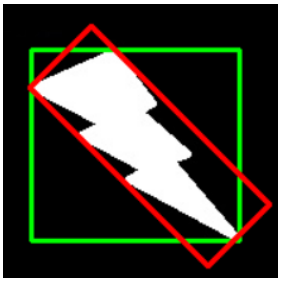
Kuva 22. vector<vector<Point> >

```
float * process()
{
    Point pal_center;
    Mat x_ax = Mat::zeros(2,1,CV_32FC1);
    Mat y_ax = Mat::zeros(2,1, CV_32FC1);
    Mat obj_pal = Mat::zeros(2,1, CV_32FC1);
    Mat pal_ang = Mat::zeros(2,1,CV_32FC1);
    proc_data obj, axis;
    float pr_d[3];
    float *pr_ptr;
    pr_ptr = pr_d;
    img = raspi_cam();
    Mat im_proc = pre_proc(img);
    vector<vector<Point> > features;
    findContours(im_proc,features,RETR_EXTERNAL,CV_CHAIN_AP-
PROX_SIMPLE);
    cout << "contours found " << features.size() << endl;
    bool ax, pal; ax = false; pal = false;
```

Haettua kaikkien objektien profiilit, kuvasta järjestetään kierros, joka käy jokaisen profiilin läpi, tasoittaa ja muuttaa viivoiksi. Jokainen profiili tähän asti on pisteiden

vektori. Jokaisen profiilin käsittely aloitetaan vasta silloin, kun sen pinta-ala ylittää tietyn arvon (”settings.json”, ”pixel_area_bound”). Tasoittaminen ja viivoittaminen seuraa profiilin suuntaa ja muuttaa suorat pisteiden suunnat viivoiksi. Tämän jälkeen ne voidaan käsitellä viivoina, joilla on alku- ja loppupiste. Tärkeimmät OpenCV:n funktiot tässä osassa ovat minAreaRect, approxPolyDP ja polylines (**Taulukko 1.**).

Taulukko 1. OpenCV:n funktiot, profiilin määrittäminen /8/

minAreaRect	Muodostaa pienimmän mahdollisen neliön löydetyn profiilin ympäri. Punainen neliö on funktion lopputulos.	
approxPolyDP	Tarkentaa funktion minAreaRect tuloksen, saa aikaan profiilin tarkan muodon.	
polylines	Piirtää viivat visualisointia varten.	

Polylines-funktiota on käytetty testeissä, joten ei ehkä kuulu tärkeimpiin funktioihin. Funktion ”polylines” tulos on sama kuin taulukossa olevan funktion ”approxPolyDP” kuvaesimerkki.

Tähän asti algoritmi on suorittanut kaikki löytämänsä profiilien muunnokset ja tietää näiden profiilien yksityiskohdat. Nämä yksityiskohdat ovat mm. kulma-asento, profiilin reunojen määrä, mitat pixeleinä jne.

```
for(int i=0; i<features.size(); i++)
{

   Mat mask = Mat::zeros(img.rows,img.cols,CV_8UC1);
   drawContours(mask,features,i,Scalar(1),FILLED,LINE_8);
   Scalar area = sum(mask);
   if(area[0] > settings["pixel_area_bound"].asUInt())
   {
      RotatedRect r = minAreaRect(features[i]);
      int lng = arcLength(features[i],true);
      vector<Point> curve;
      approxPolyDP(features[i], curve, lng*0.04,true);
      Rect bound = boundingRect(Mat(curve));
      polylines(img,curve,true,Scalar(255,0,255),6);
      stringstream ss;
```

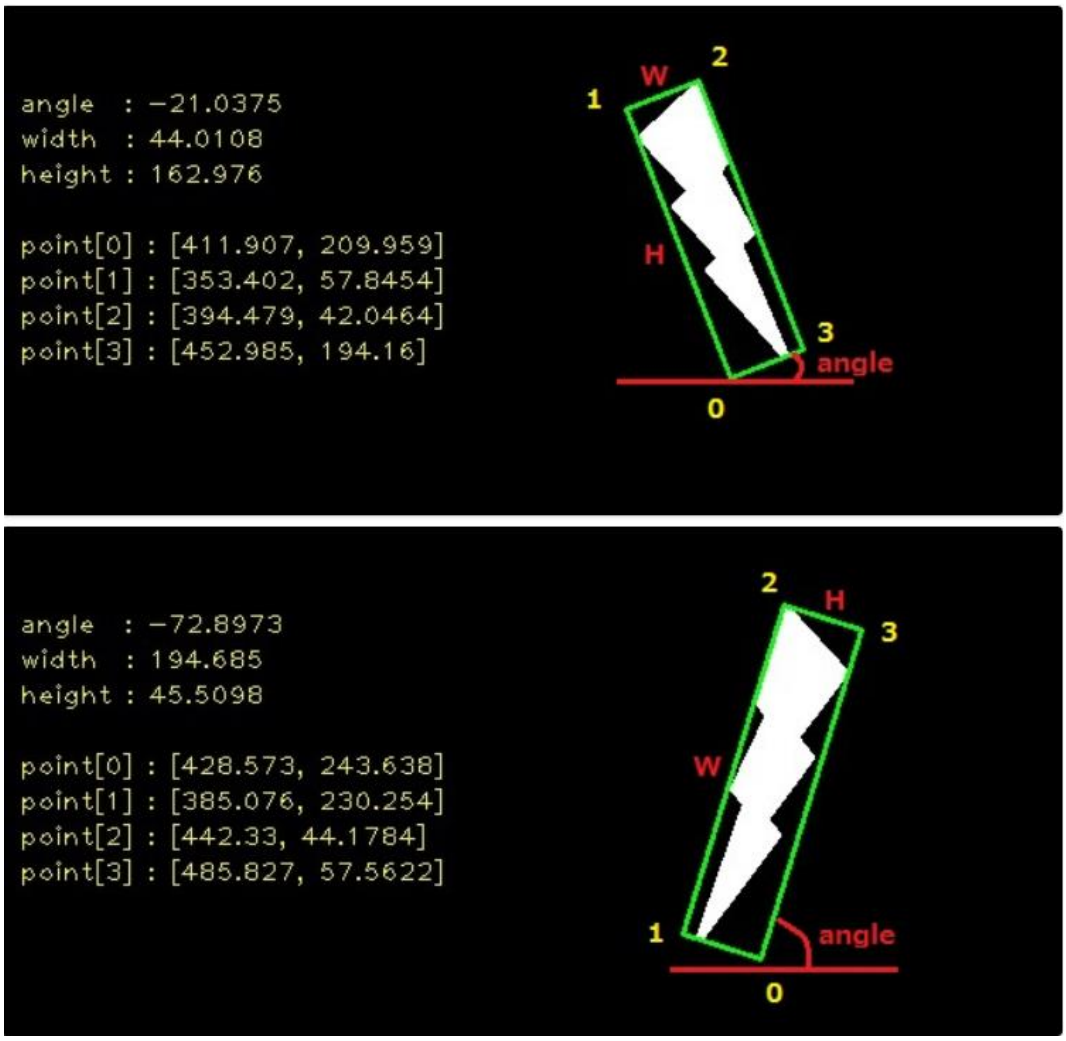
Ylempi on koodin osa, jossa aloitetaan kierros jokaisen profiilin läpi ja rakennetaan siitä “pyörivä neliö”. Profiilit suodatetaan pinta-alan perusteella. Pinta-ala lasketaan matriisin ”mask” avulla. Matriisiin ”mask” täytetään profiilin vastaavat luvut ykkösillä ja lopuksi lasketaan kaikki ykkösarvoiset pikselit yhteen. Pyrörivä neliö on pienin mahdollinen neliö, johon kyseinen profiili mahtuu kokonaan. Myös tämä neliö toimittaa sovituksen jälkeen neliön kulma-asentotiedot. Haettaessa geometrian kulma-asentoa, on pakko tehdä kulman muunnos, koska OpenCV toimittaa kulman erikoisessa muodossa. OpenCV:n kulmatietojen vaikeuden selitti eräs ”Abhishek from India” omassa blogissaan /9/. Käytännössä selviää, että hänen johtopäätökset pitävät paikkansa. Vaikka kulman tietoja tässä muodossa on käytetty vain testimerkityksessä, kannatta selittää lyhyesti miten muodostuu kulma. Samat selitykset löytyy Abhishekin blogista yksityiskohtaisemmin.

Rakentaessaan neliön erään muodon ympärille OpenCV määrittelee neliön kulmat tietyssä järjestyksessä: matalin piste on neliön nollapiste. Seuraavat pisteet saavat järjestysnumerot myötäpäivään. OpenCV on sitä mieltä, että korkeus on jänä 0. ja 1. pisteiden välillä tai 2. ja 3. välillä. Leveys on vastaavasti 1. ja 2. tai 0. ja 3 pisteiden välillä. Kulma muodostuu kuvan vaakasuoran reunan ja neliön ensimmäisen reunan väliin. Tämän neliön ensimmäisenä reunana voi olla sekä korkeus että leveys (**Kuva 23**). Tämän ajattelumallin takia joudutaan suorittamaan kulman muunnos seuraavassa koodipätkässä.

Nyt tietäen jokaisen löydetyn geometrian reunojen määrät ja kulmat, voidaan suodattaa ne muodot, jotka tarvitaan. Akseliosalla on 6 reunaa ja testikappaleella

on 4. Etsitään siis kaikki geometriat, joiden reunojen määrä vastaa 6:tta ta 4:ää. Ensiksi katsotaan kaikki, joilla on 6 reunaa. Olettaen, että työpäydällä ei voi olla muita kappaleita kuin askeli- ja testikappale, emme suorita mitään ylimääräisiä tarkistuksia, vaan luotetaan siihen, että 6 reunaa omaava kappale on meidän akseli.

Seuraavassa koodissa katsotaan kuinka napataan ja käsitellään akselikappale. Koodissa akselin uloimmat reunat määritetään ja niiden alku- ja loppupisteistä



Kuva 23. Geometrian kulman periaate

rakennetaan matemaattiset vektorit. Myös akselikappaleen vaakasuorin reuna pidetään X-akselina. Riippumatta siitä, missä asennossa akselikappale on, algoritmi määrittelee vaakasuorimman osan ja sijoittaa sen päälle X-akselin. Tästä toiminnosta vastaa itsetekemä funktio ”axis_align()”.

```
if(curve.size() == 6)
{
	double angle = r.angle;
	//if-rakenteessa tehdän yllä mainittu kulman muunnos
	if(r.size.width < r.size.height)
	{
		angle +=180;
	}
	else
	{
		angle+=90;
	}
	//axis_align(curve) etsii akselin pisimmät ja uloimmat
reunat) ja palauttaa ne ”axis_lines”-vektoriin
	vector<Point> axis_lines = axis_align(curve);
	x_ax.at<float>(0,0) = axis_lines[1].x - axis_lines[0].x;
	x_ax.at<float>(1,0) = axis_lines[1].y - axis_lines[0].y;
	y_ax.at<float>(0,0) = axis_lines[3].x - axis_lines[2].x;
	y_ax.at<float>(1,0) = axis_lines[3].y - axis_lines[2].y;
	Mat vec_vrtx(2,1,CV_32FC1); //vector for comparing
	vec_vrtx.at<float>(0,0) = 0 - img.cols;
	vec_vrtx.at<float>(1,0) = 0;//length of comparing vector
	(bottom border of the picture
	float cos_ang =
	x_ax.dot(vec_vrtx)/(sqrt(pow(x_ax.at<float>(0,0),2)+pow(x_ax
	.at<float>(1,0),2))*sqrt(pow(vec_vrtx.at<float>(0,0),2)+pow(
	vec_vrtx.at<float>(1,0),2)));
	cos_ang = acos(cos_ang)*(180/CV_PI);
	if(cos_ang > 45 and cos_ang < 135)
	{
		std::swap(x_ax, y_ax);
		std::swap(axis_lines[1], axis_lines[3]);
		std::swap(X_AXIS_LENGTH, Y_AXIS_LENGTH);
	}
	axis.x = axis_lines[0].x; axis.y = axis_lines[0].y; axis.an-
	gle = angle;
	cout << "AXIS ANGLE IS: " << axis.angle << endl;
	circle(img,axis_lines[0],20,Scalar(255,0,0),6);
	arrowedLine(img,axis_lines[0],axis_lines[1],Scalar(0,255,0),
	5);//x
	putText(img,"X",Point(axis_lines[1].x -50,
	axis_lines[1].y+100),FONT_HERSHEY_SIMPLEX,3,Sca-
	lar(0,255,0),5);
	arrowedLine(img,axis_lines[2],axis_lines[3],Scalar(0,0,255),
	5);//y
	putText(img,"Y",Point(axis_lines[3].x - 50,
	axis_lines[3].y),FONT_HERSHEY_SIMPLEX,3,Scalar(0,0,255),5);

	ax = true;
	}
```

Tässä osassa matriisi x_ax ja y_ax täyttyvät akselikappaleen pisimpien reunojen vektoriarvoilla. Vektoriarvo on esimerkiksi viivan x-loppupiste miinus x-alkupiste

ja y-loppupiste miinus y-alkupiste. Saadan vektoriarvo muodossa ai±bj tai matriisimuodossa: x_ax = $\begin{matrix} x-arvo\ (x_{loppupiste}-x_{alkupiste}) \\ y-arvo(y_{loppupiste}-y_{alkupiste}) \end{matrix}$ (1)

Otetaan kuvan alareuna vertailuvektoriksi (vec_vrtx), koska tehtävänä tällä on määrittää akselikappaleen oikea kulma-asento. Suoritetaan vektoripistetulo ja saadaan kosini x_ax ja vec_vrtx vektorien välillä. Vektorien pistetulo on:

$$\cos\alpha = \frac{x\_ax \cdot vec\_vrtx}{\|x\_ax\| \times \|vec\_vrtx\|} \quad (2)$$

Tällä kulmalla voidaan jatkossa korjata testiosan kulma. Jos x-akselin kulma on välissä ”45º < kulma < 135º” vaihdetaan x:ää ja y:tä edustavien matriisien ja muuttujien arvot keskenään. Lopuksi piirretään kaikki vektorit ja kirjoitetaan merkinnät kuvaan visualisointia varten. Muuttuja nimeltään ”ax” (tyyppi Boolean) saa arvon ”true” tarkoittaen, että tällä kierroksella akselikappale löytyi. Koko algoritmin yksi tärkeimmistä ehdoista on se, että akselikappaleita saa olla vain yksi.

Testikappaleen reunojen määrä on 4, joten lisätään toinen ehto, jossa käsitellään kaikki profiilit, joiden reunojen määrät ovat 4. Nämä ovat meidän testikappaleet.

```
else if(curve.size()==4)
{
      RotatedRect minEllips = fitEllipse(features[i]);
      circle(img,minEllips.center,10, Scalar(255,0,0),3);
      float ang = r.angle;
      pal_center = minEllips.center;
      vector<Point> pal_line = pal_align(curve);
      arrowedLine(img, pal_line[0], pal_line[1],Scalar(0,100,0),
      5);
      pal_ang.at<float>(0,0) = pal_line[1].x - pal_line[0].x;
      pal_ang.at<float>(1,0) = pal_line[1].y - pal_line[0].y;
      pal = true;

 }
```

Tämä osa toimii samalla periaatteella kuin edellinen. Funktio ”pal_align()” etsii testikappalleen pisimmän reunan, palauttaa arvot takaisin ja pisin reuna muunnettuna vektorimuotoon tallentuu matriisiin ”pal_ang”. Testikappaleen keskipiste tallentuu muuttujaan ”pal_center”. Keskipisteen tiedot ovat haettavissa OpenCV:n tuomasta funktiosta ”minAreaRect/fitEllipse”.

Funktion ”float * process()” viimeisin osa on kaikkien kerättyjen tietojen käsittely. Tämä osa alkaa ehdolla ”if(ax == true and pal == true)”, mikä tarkoittaa ”jos testikappale löytyi ja akselikappale löytyi, suorita laskutoimitukset”.

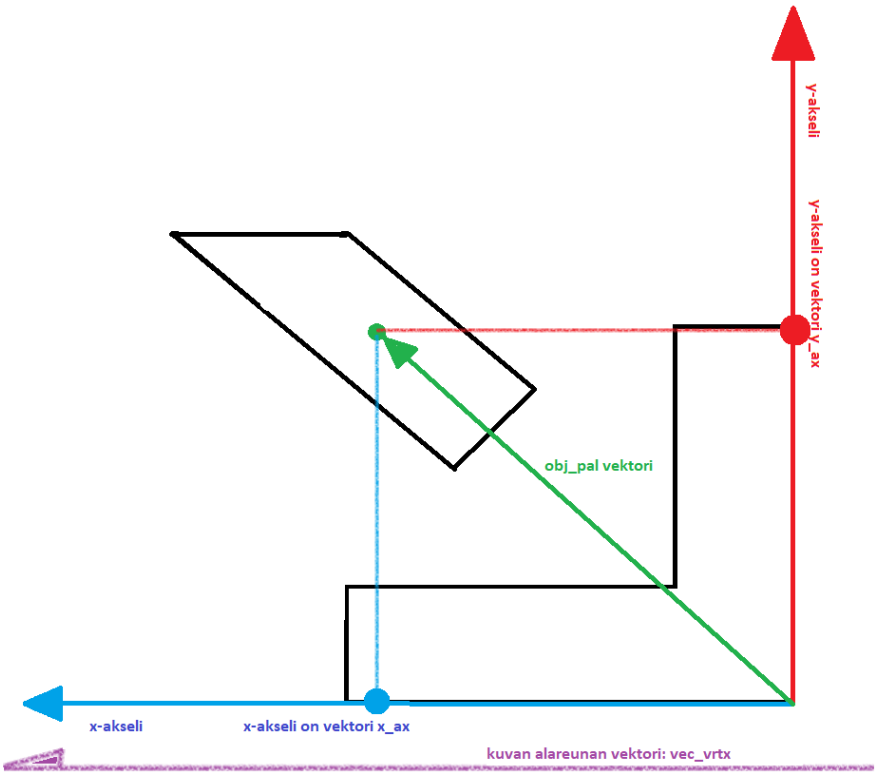
```
if(ax == true and pal == true)
{
	arrowedLine(img, Point(axis.x, axis.y), pal_center,Sca-
	lar(255,0,0), 8);
	obj_pal.at<float>(0,0) =(float) pal_center.x - axis.x;
	obj_pal.at<float>(1,0) =(float) pal_center.y - axis.y;
	float x_ax_lng =
	sqrt(pow(x_ax.at<float>(0,0),2)+pow(x_ax.at<float>(1,0),2));
	float y_ax_lng =
	sqrt(pow(y_ax.at<float>(0,0),2)+pow(y_ax.at<float>(1,0),2));
	float pal_side_lng =
	sqrt(pow(pal_ang.at<float>(0,0),2)+pow(pal_ang.at<float>(1,0
	),2));
	float obj_pal_lng =
	sqrt(pow(obj_pal.at<float>(0,0),2)+pow(obj_pal.at<float>(1,0
	),2));
	float ang_cos = obj_pal.dot(x_ax)/(x_ax_lng*obj_pal_lng);
	float ang_sin = obj_pal.dot(y_ax)/(y_ax_lng*obj_pal_lng);
	float pal_cos = pal_ang.dot(x_ax)/(pal_side_lng*x_ax_lng);
	float pal_deg_x = acos(pal_cos)*(180/CV_PI);
	float ang_deg_x = acos(ang_cos)*(180/CV_PI);
	float ang_deg_y = acos(ang_sin)*(180/CV_PI);
	cout << "PALIKAN KULMA ON: " << pal_deg_x << endl;
	cout << "length along X in pix is: " << obj_pal_lng*ang_cos
	<< endl;
	cout << "length along Y in pix is: " <<
	obj_pal_lng*sin(ang_cos) << endl;
	cout << "length along X in mm is: " << (set-
	tings["axis_x_mm"].asFloat()*(obj_pal_lng*ang_cos))/x_ax_lng
	<< endl;
	cout << "length along Y in mm is: " << (set-
	tings["axis_y_mm"].asFloat()*obj_pal_lng*ang_sin)/y_ax_lng
	<< endl;
	pr_d[0] = (X_AXIS_LENGTH*(obj_pal_lng*ang_cos))/x_ax_lng;
	pr_d[1] = (Y_AXIS_LENGTH*obj_pal_lng*ang_sin)/y_ax_lng;
	pr_d[2] = pal_deg_x;
}
else if(!(ax == true and pal == true))
{
	pr_d[0]=0; pr_d[1]=0; pr_d[2]=0;
}

	putText(img,ss.str(),Point(bound.x,
	bound.y+(bound.height/2)),FONT_HERSHEY_SIMPLEX,0.8, Sca-
	lar(0,0,255),2);

}
}
	imwrite("test.jpg", img);
	return pr_ptr;
}
```

Ideana tässä osassa on se, että lasketaan (ja myös piirretään) vektori akselikappaleen sisemmästä nurkasta testikappaleen keskipisteeseen. Sitten tämä vektori projisoidaan akselikappaleen x-akselivektoriin ja y-akselivektoriin saaden näin testikappaleen keskipisteen koordinaatit akselikappaleen x- ja y-akseleiden suhteen ja riippumatta akselikappaleen kulma-asennosta.

Seuraavalla sivulla toteutetaan esimerkkilasku, jolla selitetään matemaattinen idea. Kuvassa 24 on piirretty kappaleiden esimerkkisijoitus pöydälle. Laskutoimituksissa vektorien arvot ovat satunnaisia vain esitystä varten.



Kuva 24. Vektorien sijoitus jokaista kuvan objektia kohden.

Suoritetaan esimerkkilasku selittäen koko idea
1. Rakennetaan akseliosan akselivektorit (sininen on X; punainen on Y)
2. Rakennetaan vektori akselin nurkasta testikappaleen keskiosaan.
Otetaan esimerkiksi:

$$\overrightarrow{x\_ax} := \begin{pmatrix} 500 \\ 0 \end{pmatrix} \quad \overrightarrow{y\_ax} := \begin{pmatrix} 0 \\ 500 \end{pmatrix} \quad \overrightarrow{obj\_pal} := \begin{pmatrix} 175 \\ 250 \end{pmatrix}$$

Näiden vektorien pistetulo antaa noin 0, jos ne olisi täydellisen kohtisuorassa, niin olisi tasan nolla. Pistetulo jaettuna vektorien pituuksien tulolla antaa vektorien välinen kosini. Eli:

$$x\_ax\_cos := \frac{500 \cdot 175 + 0 \cdot 250}{\sqrt{175^2 + 250^2} \cdot \sqrt{500^2}} \qquad y\_ax\_cos := \frac{175 \cdot 0 + 250 \cdot 500}{\sqrt{175^2 + 250^2} \cdot \sqrt{500^2}}$$

$$x\_ax\_cos = 0.573 \qquad y\_ax\_cos = 0.819$$

Nyt tietäen kulman kosini välillä obj_pal ja x_ax JA obj_pal ja y_ax voidaan projsoida obj_pal kumpaankin akselin päälle ja saada riittävän tarkat koordinaattiarvot pikseleinä. Kerrotaan obj_pal vektorin pituus vastaavan akselin kosinilla.

$$pituus\_x := \sqrt{250^2 + 250^2} \cdot x\_ax\_cos \qquad pituus\_y := \sqrt{250^2 + 250^2} \cdot y\_ax\_cos$$

$$pituus\_x = 202.75 \qquad pituus\_y = 289.642$$

Tiedetään nyt pituudet pikseleinä. Jos mitataan akselikappaleen todelliset mitat x ja y suunnassa, voidaan kääntää pikselit millimetreihin.

$$aks\_pit\_mm := 132.2 \qquad aks\_kork\_mm := 137.9$$

$$\frac{aks\_pit\_mm}{\sqrt{500^2}} \cdot pituus\_x = 53.607 \qquad \frac{aks\_kork\_mm}{\sqrt{500^2}} \cdot pituus\_y = 79.883$$

Koordintatit X-pitkin(mm): 53.607

Koordinatit Y-pitkin(mm): 79.883

Koko funktion ”process()” lopuksi täytetään taulukko pr_d[] koordinaatti- ja kulma-arvoilla ja palautetaan funktioon ”main()”, joka palauttaa tiedot asiakkaalle muodossa ”x_koordinaatti:y_koordinaatti:kulma”.

4 JATKOKEHITYS

Tämä projekti on toteutettu rajoittuen tunnistettavan osan muodon. Jatkokehitykseksi voidaan laajentaa tunnistettavien osien muotoja käyttämällä esimerkiksi ”SURF”-algoritmia. Tämä algoritmi, kuten melkoinen osa tätä työtä, on kuvailtu OpenCV:n kotisivulla. Jos vaihdetaan tunnistusperiaate, voitaisiin tunnistaa monimutkaisimpiakin kappaleita, esimerkiksi yrityksen logo. Silloin tätä sovellusta on mahdollista käyttää myös laadun tarkkailussa.

Minun mielestäni tämä sovellus voidaan kirjoittaa uudelleen Java-ohjelmointikielellä, jolloin se muuttuu monialusteiseksi. Tämä mahdollistaa ohjelman toimimisen sekä Linux- että Windows-ympäristössä. Ongelmaksi voi tulla ohjelman tehokkuus, koska C++ on koneresurssien näkökulmasta paljon tehokkaampi, mutta tämä ongelma voidaan välttää, mikäli Javan asiantuntija tekee koodin. Jos päästään tehokkuusongelmasta, sovellusta voi suorittaa jopa tavallisella älypuhelimella sijoittamalla se halutulla tavalla työpöydän ylle.

Tällä hetkellä Rpi pitää sijoittaa niin, että USB tai pistorasia olisi lähellä. Kokeilisin jonkinlaista akun käyttöä, jotta laite muuttuisi täysin langattomaksi. Paras vaihtoehto olisi jokin virtapankki. Foruumeissa Internetissä väitetään, että 6 AA paristojen varauksella Rpi voi pyöriä 16 tuntiin asti.

Projektin heikko kohta piilee referenssikappaleessa. Referenssikulman korkeus ja leveys eroavat noin 5 cm. Tämä voi aiheuttaa epätarkkuutta, jos kulma sijoitetaan väärinpäin. Tämä kulma olisi hyvä valmistaa niin, että korkeus ja leveys ovat samanmittaisia. Tässä tapauksessa ei tarvitse edes muokata ohjelmaa.

Tässä työssä voidaan kokonaan päästä referenssikulmastaa eroon, mikäli sovelletaan OpenCV:n sisältämiä kalibrointifunktioita. Ainut vaikeus on silloin referenssipiste, josta robotin on laskettava koordinaattiarvot ja kulma-asento. Referenssipisteeksi voidaan muutenkin käyttää vaikka kolikkoa. Robotin liikkeet ovat tarkasti suoraviivaisia, joten kolikosta lähtevä mielivaltainen vaakasuora viiva esittää X-akselia ja pystysuora Y-akselia.

Viimeisenä kehitysajatuksena on Rpi:n näyttö. Rpi:n valmistaja myy pieniä tätä tietokonetta varten räätälöityjä kosketusnäyttöjä. Tämä näyttö ei tietenkään riitä varsinaiseen ohjelman muokkamiseen, mutta se voi olla hyödyllinen, jos halutaan nähdä viimeksi tehty kuva kaikkine tietoineen, ja muutenkin prosessin visualisointi helpottaisi loppukäyttäjän toimintaa.

Kehittämäni ohjelma on runko seuraaville konenäköprojekteille, joten kehitysajatuksia voi olla useita.

5 YHTEENVETO

Työ osoittautui odotettua helpommaksi. Syynä voi olla työn lopputuloksen asettelu. Tunnistettavaksi kappaleeksi otettiin geometrisesti yksinkertainen esimerkki. Tutkin perusteellisesti OpenCV-kirjastoa. Tämän kirjaston funktiot auttavat pääsemään liian syvällisestä matematiikasta eroon, mikä on huomattava etu konenäköjärjestelmän ohjelmoijalle. Kirjaston tekijä on laatinut melko selkeät dokumentaatiot varsin laajalle funktioiden valikoimalle. Tästä kirjastosta löytyy ratkaisuja jokaiseen konenäköongelmaan. Työn haasteellisin osa oli robottijärjestelmän ohjelmoinnin niukka osaaminen. Jouduin ”opettelemaan” RAPID-kieltä erilaisilta foruumeilta Internetissä. RAPID-kielen osaaminen ei kuitenkaan ollut projektin tärkein osa, koska minun tehtävänä oli saada toimittaa verkkoon tietyt tiedot, joita toinen robotiikan puoli voisi käsitellä haluamansa tavalla.

LÄHTEET

/1/ OpenCV:n virallinen sivu, About. Viitattu 1.3.2018. https://opencv.org/about.html

/2/ Kuivanen R. 1999. Robotiikka, Vantaa, Talentum Oyj/Metallitekniikka

/3/ Kaehler A.& Bradski G. 2016. Learning OpenCV 3, USA, Sebastopol, O`Reilly Media

/4/ Rafael Muñoz Salinas, Raspicam: C++ API for using Raspberry camera with/without Opencv. Viitattu 1.3.2018. https://www.uco.es/investiga/grupos/ava/node/40

/5/ Kaehler A.& Bradski G. 2016. Learning OpenCV 3, USA, Sebastopol, O`Reilly Media

/6/ Kaehler A.& Bradski G. 2016. Learning OpenCV 3, USA, Sebastopol, O`Reilly Media

/7/ Kaehler A. & Bradski G. 2016. Learning OpenCV 3, USA, Sebastopol, O`Reilly Media

/8/ OpenCV, Wiki. Viitatttu 1.3.2018. *https://docs.opencv.org/3.1.0/dd/d49/tutorial_py_contour_features.html*

/9/ Abhishek. Namkeen Man`s Blog. OpenCV: Determine angle of RotatedRect/minAreaRect. Viitattu 1.3.2018. https://namkeenman.wordpress.com/2015/12/18/open-cv-determine-angle-of-rotatedrect-minarearect/

LIITTEET

Ohjelman esiasettelu

```
#include <iostream>
#include <stdio.h>
#include <sys/types.h>
#include <sstream>
#include <string>
#include <sys/socket.h>
#include <opencv2/core.hpp>
#include <opencv2/imgcodecs.hpp>
#include <opencv2/imgproc.hpp>
#include <opencv2/highgui.hpp>
#include <opencv2/opencv.hpp>
#include <netinet/in.h>
#include <raspicam/raspicam_cv.h>
#include <unistd.h>
#include <jsoncpp/json/json.h>
#include <fstream>


using namespace std;
using namespace cv;

struct proc_data
{
    float x;
    float y;
    float angle;
};

Json::Value settings;
float X_AXIS_LENGTH = settings["axis_x_mm"].asFloat();
```

```
float Y_AXIS_LENGTH = settings["axis_y_mm"].asFloat();

Mat img;
Scalar green(0,255,0), blue(255,0,0), red(0,0,255);
Mat pre_proc(const Mat &img);
Mat raspi_cam();
float * process();
vector<Point> axis_align(vector<Point> crv);
vector<Point> pal_align(vector<Point> crv);
```

Funktio “main()”

```
ifstream ifs("settings.json", ifstream::binary);
    ifs >> settings;
    ifs.close();
    cout << settings;
    char buf[1024];
    int bytes_read;
    int server = socket(AF_INET, SOCK_STREAM, 0);
    int listener;
    if(server < 0)
    {
        perror("socket");
        exit(2);
    }
    struct sockaddr_in addr;
    addr.sin_family = AF_INET;
    addr.sin_port = htons(settings["port"].asUInt());
    addr.sin_addr.s_addr = htonl(INADDR_ANY);
    if(bind(server, (struct sockaddr *)&addr,sizeof(addr))<0)
    {
        perror("accept");
        exit(3);
```

```
    }
    listen(server, 1);
    while(1)
    {
        listener = accept(server, NULL, NULL);
        if(listener < 0)
        {
            perror("accept");
            exit(3);
        }
        bytes_read = recv(listener,buf,1024,0);
        if(bytes_read <= 0) break;
        cout << "Calling main processing function" << endl;
        float *p = process();
        stringstream s_d;
        s_d << p[0] << ":" << p[1] <<":" << p[2];
        //
        char to_snd[100];
        sprintf(to_snd,"%.3f:%.3f:%.3f", p[0], p[1], p[2]);
        cout << "================================================================================" << endl;
        cout << to_snd << endl;
        cout << "================================================================================" << endl;
        send(listener, to_snd, sizeof(to_snd),0);
    }
    close(listener);
    return 0;
}
```

Funktio “float * process()”

```
Point pal_center;
    //vectors filled with zeros for holding information about the axis and the detail
    Mat x_ax = Mat::zeros(2,1,CV_32FC1);
```

```
Mat y_ax = Mat::zeros(2,1, CV_32FC1);
Mat obj_pal = Mat::zeros(2,1, CV_32FC1);//9,8,2017
Mat pal_ang = Mat::zeros(2,1,CV_32FC1);
//float AX_WIDTH_PIX, AX_HEIGHT_PIX;
proc_data obj, axis;
float pr_d[3];
float *pr_ptr;//pointer do returning data array
pr_ptr = pr_d;
//raspicam code was here
img = raspi_cam();
Mat im_proc = pre_proc(img);
vector<vector<Point> > features;
findContours(im_proc,features,RETR_EXTERNAL,CV_CHAIN_APPROX_SIMPLE);
cout << "contours found " << features.size() << endl;
bool ax, pal; ax = false; pal = false;
for(int i=0; i<features.size(); i++)
{

    vector<Point> btline; vector<Point> rtline;//axis`s bottom and right-most line as "axis"
    Mat mask = Mat::zeros(img.rows,img.cols,CV_8UC1);
    drawContours(mask,features,i,Scalar(1),FILLED,LINE_8);
    Scalar area = sum(mask);
    cout << "Countour " << i << " with area: " << area << endl;
    if(area[0] > settings["pixel_area_bound"].asUInt())
    {
        RotatedRect r = minAreaRect(features[i]);
        int lng = arcLength(features[i],true);
        vector<Point> curve;
        approxPolyDP(features[i], curve, lng*0.04,true);
        Rect bound = boundingRect(Mat(curve));
        polylines(img,curve,true,Scalar(255,0,255),6);
        stringstream ss;
        if(curve.size() == 6)
        {
            double angle = r.angle;
            if(r.size.width < r.size.height)
```

```
                {
                    angle +=180;
                }
                else
                {
                    angle+=90;
                }
                vector<Point> axis_lines = axis_align(curve);
                x_ax.at<float>(0,0) = axis_lines[1].x - axis_lines[0].x; x_ax.at<float>(1,0) = axis_lines[1].y
- axis_lines[0].y;
                y_ax.at<float>(0,0) = axis_lines[3].x - axis_lines[2].x; y_ax.at<float>(1,0) = axis_lines[3].y
- axis_lines[2].y;
                Mat vec_vrtx(2,1,CV_32FC1);
                //vector for comparing
                vec_vrtx.at<float>(0,0) = 0 - img.cols; vec_vrtx.at<float>(1,0) = 0;//length of comparing vec-
tor (bottom border of the picture
                float cos_ang =
x_ax.dot(vec_vrtx)/(sqrt(pow(x_ax.at<float>(0,0),2)+pow(x_ax.at<float>(1,0),2))*sqrt(pow(vec_vrtx.at<float>(0,0
),2)+pow(vec_vrtx.at<float>(1,0),2)));
                cos_ang = acos(cos_ang)*(180/CV_PI);
                if(cos_ang > 45 and cos_ang < 135)
                {
                    std::swap(x_ax, y_ax);
                    std::swap(axis_lines[1], axis_lines[3]);
                    std::swap(X_AXIS_LENGTH, Y_AXIS_LENGTH);
                }
                axis.x = axis_lines[0].x; axis.y = axis_lines[0].y; axis.angle = angle;
                cout << "AXIS ANGLE IS: " << axis.angle << endl;

                circle(img,axis_lines[0],20,Scalar(255,0,0),6);
                arrowedLine(img,axis_lines[0],axis_lines[1],Scalar(0,255,0), 5);//x
                putText(img,"X",Point(axis_lines[1].x -50, axis_lines[1].y+100),FONT_HERSHEY_SIMPLEX,3,Sca-
lar(0,255,0),5);
                arrowedLine(img,axis_lines[2],axis_lines[3],Scalar(0,0,255), 5);//y
                putText(img,"Y",Point(axis_lines[3].x - 50, axis_lines[3].y),FONT_HERSHEY_SIMPLEX,3,Sca-
lar(0,0,255),5);
```

```
                ax = true;
            }
            else if(curve.size()==4)
            {
                RotatedRect minEllips = fitEllipse(features[i]);
                circle(img,minEllips.center,10, Scalar(255,0,0),3);
                float ang = r.angle;
                pal_center = minEllips.center;
                vector<Point> pal_line = pal_align(curve);
                arrowedLine(img, pal_line[0], pal_line[1],Scalar(0,100,0), 5);
                pal_ang.at<float>(0,0) = pal_line[1].x - pal_line[0].x; pal_ang.at<float>(1,0) = pal_line[1].y
- pal_line[0].y;
                pal = true;

            }
            if(ax == true and pal == true)
            {
                arrowedLine(img, Point(axis.x, axis.y), pal_center,Scalar(255,0,0), 8);
                obj_pal.at<float>(0,0) =(float) pal_center.x - axis.x;
                obj_pal.at<float>(1,0) =(float) pal_center.y - axis.y;
                float x_ax_lng = sqrt(pow(x_ax.at<float>(0,0),2)+pow(x_ax.at<float>(1,0),2));//length of the x
axis vector
                float y_ax_lng = sqrt(pow(y_ax.at<float>(0,0),2)+pow(y_ax.at<float>(1,0),2));//length of the y
axis vector
                float pal_side_lng = sqrt(pow(pal_ang.at<float>(0,0),2)+pow(pal_ang.at<float>(1,0),2));//length
of the object side vector
                float obj_pal_lng = sqrt(pow(obj_pal.at<float>(0,0),2)+pow(obj_pal.at<float>(1,0),2));//length
of the vector from 0,0 to pal.center
                float ang_cos = obj_pal.dot(x_ax)/(x_ax_lng*obj_pal_lng);//cosine between x axis and pal_vector
                float ang_sin = obj_pal.dot(y_ax)/(y_ax_lng*obj_pal_lng);//cosine between y axis and pal_vector
                float pal_cos = pal_ang.dot(x_ax)/(pal_side_lng*x_ax_lng);//cosine between pal_side_vector and
x axis
                float pal_deg_x = acos(pal_cos)*(180/CV_PI);
                float ang_deg_x = acos(ang_cos)*(180/CV_PI);//cosine conversion into degrees
                float ang_deg_y = acos(ang_sin)*(180/CV_PI);//--------//----------//-------//
```

```
                cout << "PALIKAN KULMA ON: " << pal_deg_x << endl;
                cout << "length along X in pix is: " << obj_pal_lng*ang_cos << endl;
                cout << "length along Y in pix is: " << obj_pal_lng*sin(ang_cos) << endl;
                cout << "length along X in mm is: " << (set-
tings["axis_x_mm"].asFloat()*(obj_pal_lng*ang_cos))/x_ax_lng << endl;
                cout << "length along Y in mm is: " << (set-
tings["axis_y_mm"].asFloat()*obj_pal_lng*ang_sin)/y_ax_lng << endl;
                pr_d[0] = (settings["axis_x_mm"].asFloat()*(obj_pal_lng*ang_cos))/x_ax_lng;//along X-axis in
millimeters
                pr_d[1] = (settings["axis_y_mm"].asFloat()*obj_pal_lng*ang_sin)/y_ax_lng;
                pr_d[2] = pal_deg_x;
            }
            else if(!(ax == true and pal == true))
            {
                pr_d[0]=0; pr_d[1]=0; pr_d[2]=0;
            }

            putText(img,ss.str(),Point(bound.x, bound.y+(bound.height/2)),FONT_HERSHEY_SIMPLEX,0.8, Sca-
lar(0,0,255),2);

        }
    }
    imwrite("test.jpg", img);
    return pr_ptr;
}
```

Funktio "Mat pre_proc(const Mat &img)"

```
    Mat im_proc = Mat::zeros(img.rows,img.cols,CV_8UC1);
    Mat bgr, test;
    blur(img, bgr,Size(img.rows/3,img.cols/3));
    im_proc = bgr-img;//background remove
    cvtColor(im_proc,im_proc,COLOR_BGR2GRAY);
    threshold(im_proc,im_proc,settings["threshold_bound"].asUInt(),255,CV_THRESH_BINARY);
    int st_element = settings["structuredElement"].asUInt();
    Mat element = getStructuringElement(MORPH_ELLIPSE,Size(st_element,st_element));
```

```
erode(im_proc,im_proc,element);
dilate(im_proc,im_proc,element);
int blr = settings["blur"].asUInt();
blur(im_proc,im_proc,Size(blr,blr));
return im_proc;
```

Funktio “Mat raspi_cam()”

```
raspicam::RaspiCam_Cv Camera;
Camera.set(CV_CAP_PROP_FORMAT, CV_8UC3);
if(!Camera.open()){cerr << "Camera is not working!" << endl; exit(1);}
sleep(2);
Camera.grab();
Mat img;
Camera.retrieve(img);
Camera.release();
return img;
```

Funktio “vector<Point> axis_align(vector<Point> crv)”

```
{
    vector<Point> to_ret;
    Point lngst1[4];
    float longest = 0;
    Mat vecx(2,1,CV_32FC1);
    Mat vecy(2,1,CV_32FC1);
    Mat vec_vrt(2,1,CV_32FC1);
    int pos = 0;
    float length = 0;
    for(int i = 0; i<crv.size(); i++)
    {
        vecx.at<float>(0,0) = crv[(i+1)%6].x - crv[i].x; vecx.at<float>(1,0) = crv[(i+1)%6].y - crv[i].y;
        //may be used simple vector length formula
        length = hypot(vecx.at<float>(0,0),vecx.at<float>(1,0));
        if(length > longest)
        {
            longest = length;
            lngst1[0] = crv[i]; lngst1[1] = crv[(i+1)%6];
```

```
            pos = i;
        }
    }

    length = 0; longest = 0;
    for(int i=0; i<crv.size(); i++)
    {
        vecx.at<float>(0,0) = crv[(i+1)%6].x - crv[i].x; vecx.at<float>(1,0) = crv[(i+1)%6].y - crv[i].y;
        //may be used simple vector length formula
        length = hypot(vecx.at<float>(0,0),vecx.at<float>(1,0));
        if(length > longest)
        {
            if(i==pos) continue;
            longest = length;
            lngst1[2] = crv[i]; lngst1[3] = crv[(i+1)%6];
        }
    }
    //looking for similar points and rearranging if needed
    for(int i=0; i<=1; i++)//In order that both vectors starts from origo!!!
    {
        for(int j = 3; j>=2; j--)
        {
            if(lngst1[i].x == lngst1[j].x && lngst1[i].y == lngst1[j].y)
            {
                if(i==1)
                {
                    Point temp1 = lngst1[1];
                    lngst1[1]=lngst1[0];
                    lngst1[0] = temp1;
                }
                if(j==3)
                {
                    Point temp1 = lngst1[3];
                    lngst1[3]=lngst1[2];
                    lngst1[2] = temp1;
                }
```

```
            }
        }
    }
    for(int i=0; i<=3; i++)
    {
        to_ret.push_back(lngst1[i]);
    }
    return to_ret;
}
```

Funktio “vector<Point> pal_align(vector<Point> crv)”

```
    float length = 0, longest = 0;
    vector<Point> ln;
    Point to_ret[2];
    int size = crv.size();
    Mat vcr(2,1,CV_32FC1);
    for(int i=0; i<crv.size(); i++)
    {
        vcr.at<float>(0,0) = crv[(i+1)%size].x - crv[i].x; vcr.at<float>(1,0) = crv[(i+1)%size].y - crv[i].y;
        length = hypot(vcr.at<float>(0,0), vcr.at<float>(1,0));
        if(length > longest)
        {
            longest = length;
            to_ret[0] = crv[(i+1)%size]; to_ret[1] = crv[i];
        }
    }
    ln.push_back(to_ret[0]); ln.push_back(to_ret[1]);
    if(ln[0].y < ln[1].y) std::swap(ln[0], ln[1]);
    return ln;
```

Compiling in Linux, g++

```
//compiling: g++ $(pkg-config --libs --cflags opencv) -lraspicam -lraspicam_cv -ljsoncpp -o output_bin file.cpp
```

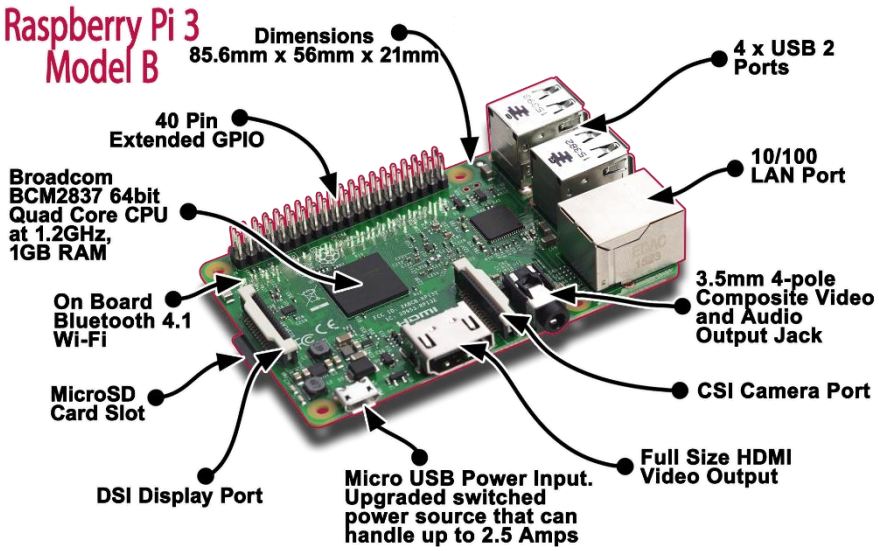
Note that libraries opencv, raspicam, jsoncpp should be compiled and installed before

Käyttöohje

Tämä ohje opastaa käyttäjää Raspberry Pi:n, OpenCV:n ja tunnistusohjelman asennuksessa. Tässä ohjeessa oletetaan, että käyttäjällä on Linux-järjestelmien perusosaaminen. Ohjeen lukija tarvitsee Raspbery Pi model 3 tietokoneen (kuva 1.), Raspberry Pi:in sopivan kameran (esim. kuva 2.) ja jonkin näytön. Ohjeen kirjoittaja on käyttänyt kotitelevisiota HDMI-liitännällä. Tähän tietokoneeseen tulee kytkeä sopiva kamera kuvassa 1. mainittuun liitoskohtaan ”CSI Camera Port”. Kamera voidaan hankkia joko Raspberryn virallisesta kaupasta, jolloin sen hinta ylitsee 20 euroa, tai Kiinasta, jossa kameran hinta on noin 1,5 – 10 euroa.

Virtajohtimena sopii tavallinen älypuhelimien latausjohto. Se kytketään kuvassa 1. olevaan ”Micro USB Power Input” liitäntään.

Käyttöjärjestelman asennusta varten tarvitaan MicroSD muistikortti. Suositellaan hankkimaan vähintään 8 GB:n kokoinen kortti, koska muistikorttiin pitää ladata myös OpenCV paketti, jonko koko on noin 4-5 GB:a mukaan lukien asennuspaketti. Muistikortti asennetaan ”MicroSD Card Slot” kohtaan kuvan mukaisesti (kuva 1.).



Kuva 26. Rasperry Pi model 3

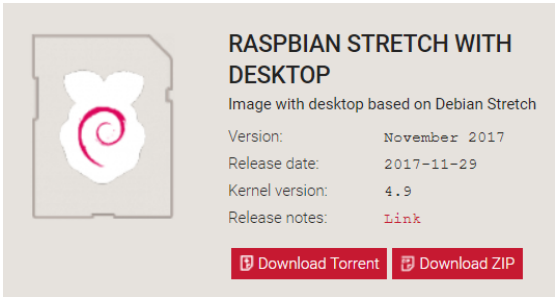
Kuva 25. Raspberry Pi:n kamera

Käyttöjärjestelmän lataaminen ja asennus

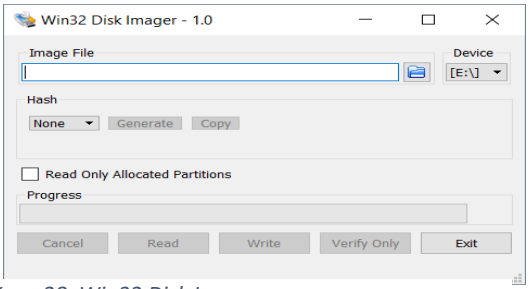
Käyttöjärjestelmä ladataan Raspberry Pi:n valmistajan sivulta. Avataan raspberryn kotisivu, ”Download”-osa[2] ja painetaan ”Download ZIP” (kuva 3.). Kun tiedosto nimella ”[nimi].img” on latautunut se tulee kirjoittaa MicroSD muistikortille. Tätä varten käytetään ohjelmaa ”Win32 Disk Imager”. Voidaan käyttää muitakin ohjelmia asennuspaketin käsittelemiseen. Asetetaan muistikortti tavalliseen tietokoneeseen ja ladataan ohjelma, asennetaan ja avataan (Kuva 4.). Kenttään ”Image File” syötetään osoite, jossa Raspbian ”img”-tiedosto sijaitsee. Valikossa ”Device” valitaan muistikorttia vastaava kirjain ja painetaan ”Write” painike. Tiedoston siirtäminen muistikortille alkaa (kuva 5.).

Siirron jälkeen muistikortti on käyttövalmis ja se voidaan asettaa Raspberry-tietokoneeseen. Tällä hetkellä kannattaa kytkeä Raspberry-kone johonkin näyttöön, esimerkiksi HDMI-interfeissia käyttäen, ja näppäimistöön, koska aluksi Raspbian ei ole säädetty hallittavaksi verkon välityksellä. Raspberry-tietokone käynnistyy heti, kun virtajohto on kytketty. Käyttöjärjestelmän käynnistyksen jälkeen liitytään joko Wifi- tai langalliseen verkkoon. Tämä tehdään painamalla Wifi-kuvaketta oikeassa ylänurkassa.

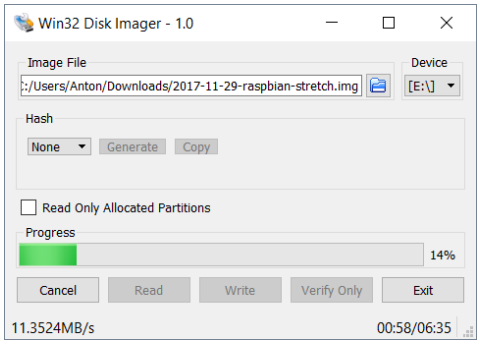
Aluksi säädetään Raspbian toimimaan verkon välityksellä. Tätä varten käytetään ”ssh”-teknologia. Avataan komentoikkuna ja syötetään ”***sudo raspi-config***”. Ilmestyneestä valikosta painetaan ”Interfacing Options”. Etsitään luettelosta ”SSH”, valitaan se ja painetaan ”yes”. Lopuksi painetaan ”Finish”-painike. Samalla tavalla annetaan käyttöjärjestelmälle lupa käyttämään etäpöytäyhteyttä, minkä jälkeen voidaan liittyä Raspbianiin etäpöytä-palvelun avulla käyttämällä esimerkiksi ohjelmaa ”TightVNC”. Tavallisesti käyttöjärjestelmän käyttäjä on ”pi” ja salasana on ”raspberry”. Nämä suositellaan vaihtaa heti asennuksen jälkeen. Ennen SSH:n käyttöä täytyy selvittää Raspbianin saama IP-osoite.



Kuva 27. Raspbian lataaminen



Kuva 28. Win32 Disk Imager



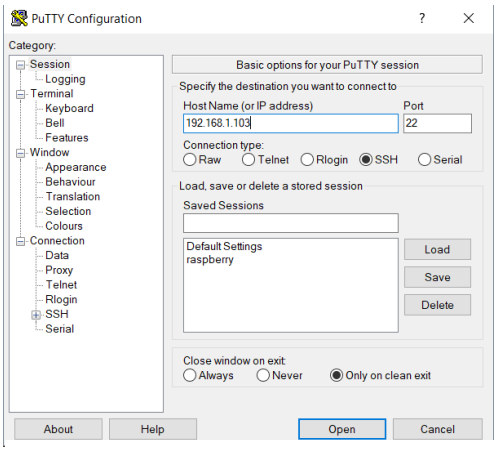
Kuva 29. Asennuspaketin siirto

[2] https://www.raspberrypi.org/downloads/raspbian/

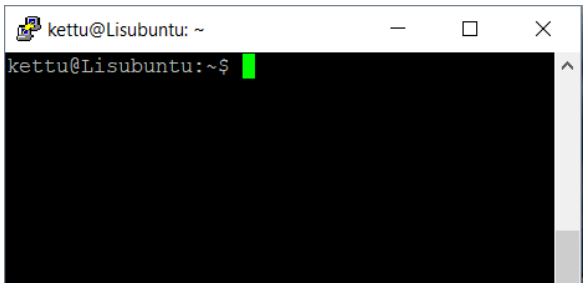
Raspberry-koneesta voidaan nyt ottaa HDMI-johto ja näppäimistö pois. Mikäli käytetään SSH-palvelua Raspbianin hallintaan suositellaan käyttää ”Putty”-ohjelmaa[3]. Tämä ohjelma antaa mahdollisuuden hallita Raspbiania komentokehotteen avulla verkon kautta.

Siirrytään toiselle tietokoneelle ja avataan Putty. Ohjelman kenttään ”Host Name” syötetään Raspbian-koneen IP-osoite ja painetaan ”Open” (kuva 6.). Seuraavaan ikkunaan syötetään käyttäjänimi (”pi”) ja salasana (”raspberry”). Kirjainten koko on tärkeä, joten kaikki pienin kirjaimin. Jos kaikki on tehty oikein ilmestyy kuvan (kuva 7.) mukainen ikkuna. Tämä on Raspbianin komentoikkuna, jolla voidaan hallita tietokonetta.

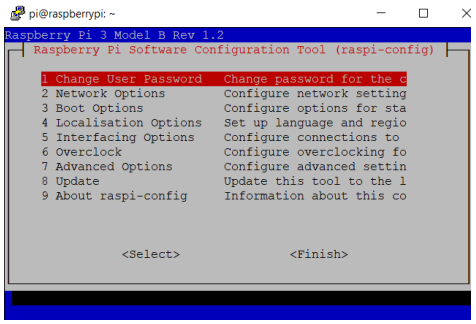
Tehdään viimeinen käyttöjärjestelmän säätö käyttämällä Putty-komentoikkunaa. Ensiksi asennetaan kamera Raspberry:iin siihen tarkoitettuun liitoskohtaan. Syötetään ikkunaan komento ”***sudo bash***” saadaksemme oikeudet säätämään käyttöjärjestelmää. Seuraavaksi kirjoitetaan ikkunaan ”***raspi-config***”. Valitaan ”Interfacing Options” (kuva 8.) ja seuraavassa näkymässä ”P1 Camera” (kuva 9.). Vastataan kysymykseen ”Yes” ja ”Ok” (kuva 10.). Edellisillä toimenpiteillä annettiin käyttöjärjestelmälle lupa käyttämään kamera.



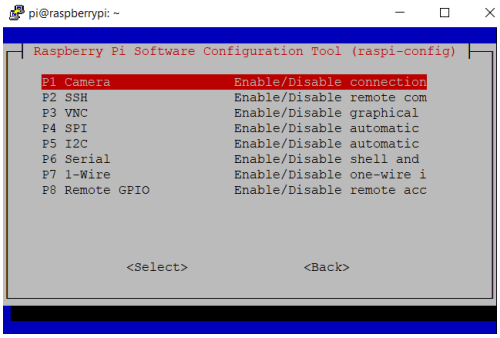
Kuva 30. PuTTY aloitusnäkymä



Kuva 31. Komentoikkuna

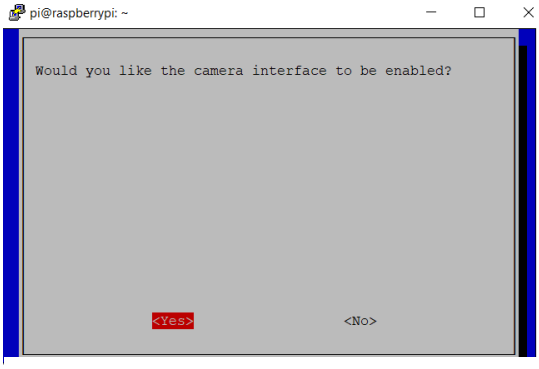


Kuva 32. Interfacing Options



Kuva 33. Camera enable

[3] https://www.chiark.greenend.org.uk/~sgtatham/putty/latest.html



Kuva 34. Varmistus

Lopuksi käyttöjärjestelmä kannattaa päivittää. Syötetään komentoikkunaan kaksi peräkkäistä komentoa erottaen ne ”;” merkillä ”***apt-get update; apt-get upgrade***”.

OpenCV:n lataus ja asennus

OpenCV on kirjasto, jonka päättötyön ohjelma tarvitsee. Seuraavissa askelissa oletetaan, että Raspberry-koneella on pääsy Internettiin. Avataan komentoikkuna, siirrytään kansioon ”Downloads” komennolla ”***cd /home/pi/Downloads***” ja ladataan OpenCV syöttämällä ”***wget https://github.com/opencv/opencv/archive/3.4.1.zip***”. Kun tiedosto on latautunut se tulee purkaa komennolla ”***unzip 3.4.1.zip***”. Jos halutaan käyttää muita tunnistusalgoritmeja (”SURF”, ”SIFT” yms.) täytyy ladata OpenCV:n ”opencv_contrib” lisäosa. Tämän osan saa syöttämällä komento ***”wget https://github.com/opencv/opencv_contrib/archive/master.zip”***. Lisäosa pitää purkaa samalla komennolla ”unzip [tiedoston_nimi]”.

OpenCV myös tarvitsee muutama kirjasto. Asennetaan nämä OpenCV:n latauksen aikana. Avataan toinen komentoikkuna, saadaan kaikki oikeudet ”***sudo bash***”-komennolla ja syötetään alla olevat komennot.

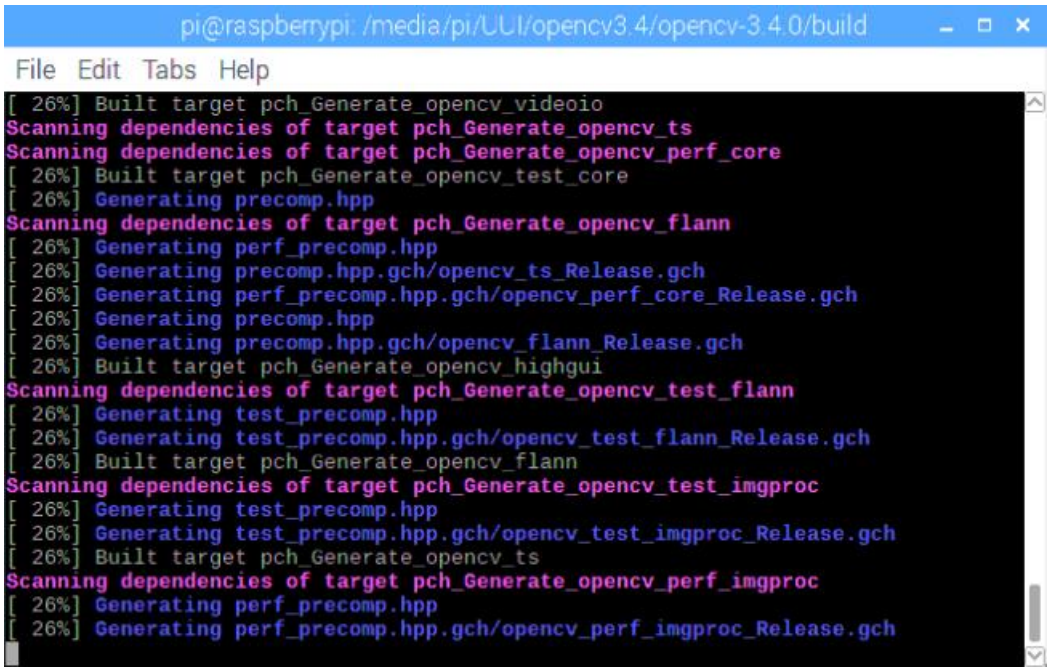
apt-get install build-essential cmake pkg-config libjpeg-dev libtiff5-dev libjasper-dev libpng12-dev
apt-get install libavcodec-dev libavformat-dev libswscale-dev libv4l-dev libxvidcore-dev libx264-dev
apt-get install libgtk2.0-dev
apt-get install libatlas-base-dev gfortran
apt-get install python2.7-dev python3-dev

Latauksen ja lisäkirjastojen asennuksen jälkeen voidaan aloittaa OpenCV:n kääntämisen. Syötetään seuraava koodi komentoikkunaan:

```
$cd ~/Downloads/opencv-3.4.1  //siirrytään kansioon ”Download/opencv…”
$mkdir build                  //luodaan kansio nimeltään ”build”
$cd build                     //siirrytään äsken luomaamme kansioon
cmake -D CMAKE_BUILD_TYPE=RELEASE \
    -D CMAKE_INSTALL_PREFIX=/usr/local \
    -D OPENCV_EXTRA_MODULES_PATH=~/Downloads/[lisäosien_kansio]/modules \
    -D BUILD_EXAMPLES=ON ..
```

[lisäosien kansio] on kansio, johon ”opencv_contrib” on purettu. Oletetaan, että tämä kansio on käyttäjän kotiliittymän ”Downloads”-kansiossa. Syöttäkää ”[lisäosien_kansio]” tilalle kansion nimi, johon ”contrib” on purettu. Komento ”cmake…” valmistaa kaikki tarvittavat tiedostot/kirjastot ennen varsinaista kääntämistä sekä generoi tiedoston, jonka avulla kääntäminen käynnistyy. Jos käyttäjä kokee komentoikkunan interfeissi vaikeaksi on mahdollista käyttää ohjelma ”cmake_gui”. Tässä ohjelmassa kaikki kääntämisen asivalmistelun asetukset ovat selkeästi näkyvillä ja ne voi valita vain rastitamalla. Ohjelman käyttö-ohje löytyy Internetista.

Kun ”cmake” on tehnyt työnsä onnistuneesti on aika kääntää. Joskus kääntäminen ei onnistu eri syistä. Silloin kääntäjä näyttää viestin, jossa lukee, minkä takia kääntämisprosessi on katkennut. Kääntäminen aloitettaan syöttämällä komento: ”make –j4”. Parametri ”-j” ja sen arvo ”4” kertoo kääntäjällä kuinka monta CPU:n ydintä saa käyttää kääntäessä. Tämän arvon voi vaihtaa riippuen käyttäjän CPU:n ydinten määrästä. Komennon jälkeen kääntäminen alkaa välittömästi. Komentoikkunan näkymä on kuvan 11 mukainen.



Kuva 35. OpenCV:n kääntäminen on menossa

Kääntämisen valmistuttua komentoikkunaan ilmestyy ”[100%] …” vihreällä kirjoitettuna. Kääntäminen voi viedä aika paljon aikaa riippuen OpenCV:n lisäosien määrästä. Tavallisesti tämä prosessi etenee noin 2 tuntia.

Kun kääntäminen on valmis käyttäjän on vain asennettava kaikki kompiloidut kirjastot järjestelmään. Tämä tehdään komennolla ”make install” poistumatta nykyisestä kansiosta. Asennuksen jälkeen suoritetaan komento ”ldconfig”, joka päivittää järjestelmässä rekisteröityjen kirjastojen luettelo. HUOM.: ”make install” ja ”ldconfig” tule suorittaa ”root”-käyttäjänä. Joko käytetään komennon lisäys ”sudo” tai kirjaudutaan komentoikkunan kauttaa ”root”-ympäristöön. OpenCV-kirjasto on nyt käyttövalmis.

Tässä projektissa on myös käytetty pieni kirjasto ”raspicam”. Kirjasto ohjaa Raspberryyn liitetyn kameran toiminnan. On toki mahdollista ohjata kameraa OpenCV:n kirjaston kautta, mutta kirjoittaja kokee raspicam-interfeissin mukavammaksi.[

Raspicam-kirjasto[4] on ladattavissa osoitteesta: https://sourceforge.net/projects/raspicam/files/. Käyttöohjeen kirjoitushetkellä viimesin versio oli 0.1.6. Ladataan viimeisin versio käyttäjän haluamaan kansioon, puretaan ”.zip”-tiedosto ja siirrytään purettuun kansioon. Komennolla ”***mkdir build***” luodan uusi kansio ja siirrytään siihen. Syötetään vuorotellen seuraavat komennot:

```
cmake                       //Kääntämisen esiasetukset
make                        //käännä
[sudo] make install         //asenna
[sudo] ldconfig             //päivitä järjestelman kirjastoluet-
                            telo
```

[4] https://www.uco.es/investiga/grupos/ava/node/40

Viimeisin projektissa käytetty lisäosa on ”jsoncpp”. Tämän krijaston avulla ohjatan ”.json”-tiedostoja, joita on hyvä käyttää asetusten tallentamiseen. Tämä kirjasto ei vaadi kääntämistä. Tämän kirjaston saa Linuxin sovelluspalvelimelta syöttämällä komento: ***”[sudo] apt-get install libjsoncpp-dev***”.
Tämän päätätyön kohteena oleva projekti täytyy myös kääntää. Luodaan jokin kansio, esim. ”***mkdir ~/Pixel_Works***”. Kopioidaan tähän kansioon kaksi projektin tiedostoa: ”main.cpp” ja ”settings.json”. Pitää seurata, että molemmat tiedostot pysyvät yhdessä samassa kansiossa aina. Seuraavaksi syötetään komento: ”***g++ $(pkg-config –libs –cflags openc) –lraspicam –ljsoncpp –lraspicam_cv –o [ohjelman_haluttu_nimi] main.cpp***”. Jos kääntäminen loppui onnistuneesti, kansioon ilmestyy tiedosto nimellä ”[ohjelma_haluttu_nimi]”. Leikitään, että ”ohjelman_haluttu_nimi” on ”my_test.bin”. Ohjelma käynnistyy komennolla: ”.\my_test.bin”. Heti käynnistyksen jälkeen ohjelma näyttää komentoikkunaan sen asetukset ja siirtyy ”kuuntelu”-tilaan. Tässä tilassa ohjelma odottaa kyselyjä ”settings.json”-tiedostossa määritettyyn prottiin. Täytyy mainita, että ohjelma odottaa mitä vaan kyselyä, vaikka ”terve”. Tärkein on se, että kysely ei ole tyhjä. Oletuksena ”settings.json”-tiedostossa määritetty porttinumero on 3425. Tällä hetkellä oletetaan, että kamera on asennettu ja suunnattu tunnistettavalle aueelle.
Siirrytään toiselle tietokoneelle tai robotti-interfeisille. Jos käytetään robotti-interfeissiä ohjelman, joka lähettää kyselyjä socketin avulla täytyy olla tehty. Jos testataan toiselta tietokoneelta (Linux) tarkistetaan ohjelman toimintaa komennolla: ***echo ”jotakin” | netcat [raspberry_koneen_ip_osoite] [raspbian_koneen_porttinumero]***. Esimerkiksi: ”***echo ”jotakin” | netcat 192.168.0.1 3425***”.
HUOM. Robotin ja raspbian-koneen tulee olla samassa verkkoalueessa, eli niiden ip-osoitteiden ero näkyy vain kolmannen pisteen jälkeen. Esim. IP-osoite1: 192.168.0.1 ja IP-osoite2: 192.168.0.129. Tässä esimerkissä molemmat ovat samassa verkossa. Jos tätä vaatimusta ei voida toteuttaa pyytäkää verrkoadministraattorinne avuksi.