RASPBERRY PI REAALIAIKAISEEN KUVANTUNNISTUKSEEN

Verneri Keskinen

Opinnäytetyö
Toukokuu 2014
Tietotekniikan koulutusohjelma
Sulautetut järjestelmät ja elektroniikka

TAMPEREEN AMMATTIKORKEAKOULU
Tampere University of Applied Sciences

TIIVISTELMÄ

Tampereen ammattikorkeakoulu
Tietotekniikan koulutusohjelma
Sulautetut järjestelmät ja elektroniikka

VERNERI KESKINEN:
Raspberry Pi reaaliaikaiseen kuvantunnistukseen

Opinnäytetyö 46 sivua, joista liitteitä 7 sivua
Toukokuu 2014

Tässä opinnäytetyöraportissa käsitellään kuinka Raspberry Pi:tä saadaan hyödynnettyä kuvantunnistussovelluksissa. Työn tavoitteena oli tutustua itse Raspberry Pi -tietokoneeseen, kuvantunnistukseen ja ohjelmointiin Linux-ympäristössä. Työn lopullinen tavoite oli saada aikaiseksi jokin reaaliaikainen kuvantunnistussovellus Raspberry Pi:llä.

Työ aloitettiin tutustumalla Raspberry Pi:n ja sen Raspbian-käyttöjärjestelmän käyttöön sekä erilaisiin reaaliaikaisiin kuvantunnistussovelluksiin mitä muut Raspberry Pi:n käyttäjät olivat toteuttaneet. Tutustumisvaiheessa vertailtiin lukuisia erilaisia kameravaihtoehtoja ja kuvantunnistusmenetelmiä, mutta lopulta työssä päädyttiin käyttämään Raspberrry Pi:lle valmistettua kameramoduulia ja OpenCV-kirjastoa. Ohjelmointikieleksi valittiin C++.

Työssä haasteeksi muodostui ohjelmointi ja erityisesti ohjelmakoodin kääntäminen Linux-ympäristössä. Raspbianin gcc/g++-kääntäjän käyttöä jouduttiin opiskelemaan jonkin aikaa ennen kuin ohjelmakoodin kääntäminen alkoi luonnistumaan. Hyötynä tästä kääntäjän opiskelusta oli se, että kääntäjien toiminnasta tiedetään nyt huomattavasti enemmän, kuin ennen työn aloittamista.

Toiseksi haasteeksi muodostui se, ettei OpenCV-kirjasto tukenut suoraan Raspberry Pi:n kameramoduulia, mutta tämäkin ongelma ratkesi pienen opiskelun jälkeen. Kun Raspberry Pi oltiin saatu siihen kuntoon, että tarvitsi vain kirjoittaa toteutettavalle reaaliaikaiselle kuvantunnistussovellukselle ohjelmakoodi, niin tutkittiin vielä erilaisia sovellusvaihtoehtoja. Työssä päädyttiin toteuttamaan kohteen seuranta -sovellus, joka kykenee värien perusteella seuraamaan reaaliajassa videokuvasta useita määriteltyjä kohteita.

Kohteen seuranta saatiin toimimaan ja työlle asetetut tavoitteet katsottiin saavutetuksi. Työssä tuli selväksi kuinka monipuolinen pieni laite Raspberry Pi on. Se kykenee lukemattomiin erilaisiin käyttötarkoituksiin ja on täten erinomainen työkalu kenelle tahansa ohjelmoinnista kiinnostuneelle tai huokeaa tietokonetta sovellukseen etsivälle yritykselle.

Asiasanat: raspberry pi, reaaliaikainen, kuvantunnistus, opencv, tietokonenäkö

ABSTRACT

Tampereen ammattikorkeakoulu
Tampere University of Applied Sciences
Degree Programme in ICT Engineering
Embedded systems and Electronics

VERNERI KESKINEN:
Raspberry Pi for real time image recognition

Bachelor's thesis 46 pages, appendices 7 pages
May 2014

This thesis examines how Raspberry Pi can be used for real-time image recognition applications. The goal of the thesis was to get acquainted with Raspberry Pi, its Raspbian operating system and programming in Linux environment. Final goal of the thesis was to accomplish some kind of real-time image recognition application with Raspberry Pi.

The project began with getting acquainted with Raspberry Pi, its Raspbian operating system and image recognition applications that other Raspberry Pi users had created. In this phase of the project several camera and image recognition options were weighted but in the end camera module specifically made for Raspberry Pi and OpenCV image recognition library were chosen.

In the project programming and especially compiling code on Linux became a challenge. Raspbian's gcc/g++-compiler had to be studied for a while before code compilation succeeded. As a result a lot was learned about how compilers work in general.

Another challenge was that OpenCV didn't support Raspberry Pi camera module natively but this problem was solved with little bit of studying. After getting Raspberry Pi to that stage where the only thing left to do was to write code for the image recognition application different application options were weighted. Object tracking application that could track multiple objects based on color was decided.

Object tracking application succeeded and the goals of the thesis were accomplished. In this project it became clear that Raspberry Pi is a very versatile little device and is capable for countless amount of applications. Therefore Raspberry Pi is an excellent tool for anyone interested in programming or for a company looking for a cheap computer to use in their project.

Key words: raspberry pi, real time, image recognition, opencv, computer vision

SISÄLLYS

ERITYISSANASTO

ARM	Advanced RISC Machines. 32-bittinen mikroprosessoriarkkitehtuuri.
CMake	Usealla alustalla toimiva käännösjärjestelmä.
CSI	Camera Serial Interface. Kuvasensorien tiedon siirtämiseen käytetty väylä.
DSI	Display Serial Interface. Sisältää sarjaliikenneväylän sekä protokollan kuvatiedon lähteen ja kohteen välistä kommunikointia varten.
Ethernet	Pakettipohjainen lähiverkkoratkaisu.
Git	Versionhallintaohjelmisto, joka on suunniteltu toimimaan hajautetusti ja mahdollisimman tehokkaasti.
GNU	GNU's Not Unix. Richard Stallmanin vuonna 1983 käynnistämä projekti, jonka tavoitteena oli kehittää täysin vapaa käyttöjärjestelmä.
GPIO	General Purpose I/O. Yleiskäyttöinen portti mikroprosessoreissa.
HDMI	High Definition Multimedia Interface. Kuvan ja monikanavaäänen siirtämiseen suunniteltu digitaalinen näyttölaitteiden liitäntästandardi
HSV	Hue Saturation Value. Värimalli jossa värit kuvataan sävyn, kylläisyyden ja kirkkauden arvoina.
I²C	Yksinkertainen kaksisuuntainen ohjaus- ja tiedonsiirtoväylä.
I²S	Digitaalisten äänilaitteiden yhdistämiseen käytetty väylä.
I/O	Input/Output. Tietokoneen ja ulkomaailman välistä kommunikointia.
IP	Internet Protocol. Internet protokolla, joka huolehtii tietoliikennepakettien toimittamisesta perille pakettikytkentäisessä verkossa.
Kernel	Käyttöjärjestelmän ydin, joka hallinnoi ohjelmien I/O-pyyntöjä ja kääntää ne prosessorille sekä muille tietokoneen elektronisille komponenteille sopiviksi käskyiksi.
Make	Työkalu erilaisten tiedostojen luonnin automatisointiin.

MMAL	Multi-Media Abstraction layer. Runko jota käytetään tarjoamaan yksinkertainen matalan tason rajapinta VideoCore-näytönohjaimella ajettaville multimediakomponenteille.
MMC	MultiMediaCard. Flash-muistikorttityyppi.
MMX	Intelin käskykanta laskentayksikölle, joka suorittaa synkronoidusti samaa ohjelmakoodia eri data-alkioille.
NOOBS	New Out Of Box Software. Raspberry Pi -säätiön kehittämä käyttöjärjestelmän asennustyökalu.
RCA	Radio Corporation of American kehittämä audio- ja videolaitteissa käytetty liitäntätapa.
RGB	Värimalli jossa eri värejä muodostetaan sekoittamalla keskenään punaista, vihreää ja sinistä.
SD	Secure Digital. Yleisesti käytetty muistikorttityyppi.
SDIO	Secure Digital Input Output. SD-standardiin lukeutuva muistikorttityyppi, joka tukee myös I/O-toimintoja.
SPI	Serial Peripheral Interface. Synkronoitu sarjaväylä.
SSE	Streaming SIMD (Single Instruction, Multiple Data) Extensions. Intelin käskykantalaajennus IA-32 -käskykantaan.
UART	Universal Asynchronous Receiver Transmitter. Rinnakkaismuotoista tietoa sarjamuotoiseksi ja päinvastoin muuntava sarjaliikennepiiri.
Unix	Laitteistoriippumaton käyttöjärjestelmä.
USB	Universal Serial Bus. Sarjaväyläarkkitehtuuri oheislaitteiden liittämiseksi tietokoneeseen.
Userland	Userland-termi viittaa erilaisiin ohjelmistoihin ja kirjastoihin, joita käyttöjärjestelmä käyttää keskustelemaan kernelin kanssa.
V4L	Video4Linux. Linuxille rakennettu ohjelmointirajapinta ja ajurirunko videon kaappaamista ja tuottamista varten.
YUV	Ihmisnäön huomioon ottava väriavaruus, joka koostuu kolmesta komponentista.
YUV I420	YUV-väriavaruutta käyttävä kuvaformaatti, jossa yhtä pikseliä kohden on 12-bittiä

1 JOHDANTO

Tämän opinnäytetyön tarkoituksena oli tutustua Raspberry Pi -tietokoneeseen, siinä käytettävään Linux-pohjaiseen Raspbian-käyttöjärjestelmään, kuvantunnistukseen ja lopullisena tavoitteena oli toteuttaa kyseisellä laitteella jokin reaaliaikainen kuvantunnistussovellus. Sovellus tuli toteuttaa käyttäen Raspberry Pi -tietokoneelle kehitettyä kameramoduulia ja avoimen lähdekoodin kuvantunnistuskirjastoa nimeltä OpenCV.

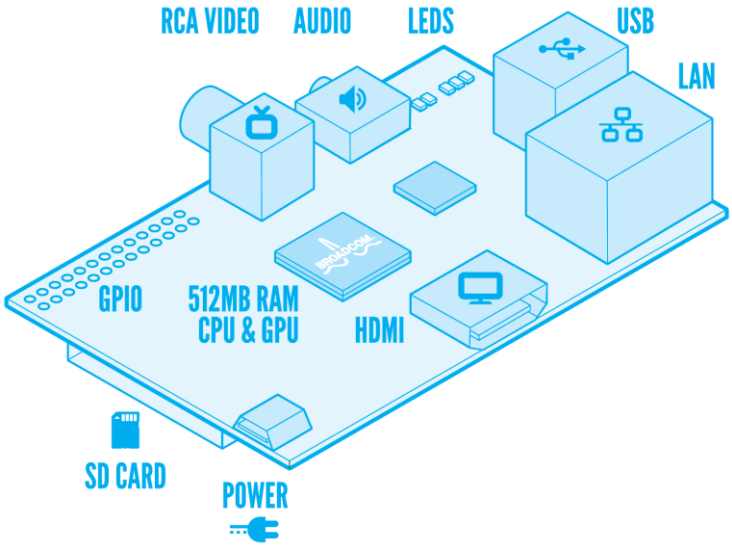
Raspberry Pi on suunnilleen luottokortin kokoinen tietokone Linux-käyttöjärjestelmällä. Raspberry Pi tuli myyntiin helmikuussa 2012 ja on sen jälkeen saavuttanut suuren suosion tietotekniikan harrastajien ja ammattilaisten keskuudessa. Suuren suosion syynä on edullinen hinta, pieni koko, ekologisuus ja monipuolisuus. Näiden etujensa takia Raspberry Pi soveltuu lukemattomiin erikoiskäyttötarkoituksiin, joihin eivät perinteiset tietokoneet sovellu suuren kokonsa, hintansa ja virrankulutuksensa takia.

Kuvantunnistus ja etenkin konenäkö, eli sovellukset, joissa tietokone on ohjelmoitu havainnoimaan ympäristöstänsä erinäisiä asioista kameran avulla, käyttäen monimutkaisia kuvantunnistusalgoritmeja, ovat yleistyneet viime vuosina tietokoneiden prosessoritehon kasvamisen ansiosta. Raspberry Pi ei tosin ole prosessoritehonsa puolesta erityisen tehokas ja tästä syystä työssä täytyi sovellusta toteutettaessa ottaa erityisesti huomioon Raspberry Pi:n hyvin rajalliset resurssit.

Tässä työssä perehdytään aluksi itse Raspberry Pi -tietokoneeseen, käyttöjärjestelmään, kameramoduuliin sekä kuvantunnistuksen kautta OpenCV-kirjastoon. Tämän opinnäytetyön pääpaino on kuitenkin toteutetussa kohteen seuranta -sovelluksessa sekä Raspberry Pi:n käyttöönotossa kuvantunnistussovelluksia varten.

2 RASPBERRY PI -TIETOKONE

Raspberry Pi on edullinen luottokortin kokoinen tietokone, johon on mahdollista kytkeä lukuisia oheislaitteita kuten; hiiri, näppäimistö, näyttö ja kaiuttimet. Se on hyvin monipuolinen pieni laite, joka mahdollistaa kaiken ikäisten ihmisten tutkia tietokoneita ja opiskella erilaisia ohjelmointityökaluja kuten Scratch ja Python. Raspberry Pi kykenee kaikkeen mihin tavallinen pöytätietokonekin kykenee. Tästä huolimatta Raspberry Pi on kuitenkin jotain ihan muuta mitä nykypäivän modernit pöytätietokoneet ovat. Tässä luvussa on käsitelty lyhyesti Raspberry Pi -tietokoneen historiaa ja tarkemmin tekniset ominaisuudet. Alla olevassa kuvassa (kuva 1) on esitetty Raspberry Pi -tietokoneen B-malli. [1]



KUVA 1. Raspberry Pi:n B-malli. [1]

2.1 Historiaa

Raspberry Pi -tietokoneen on kehittänyt vapaaehtoisvoimin englantilainen Raspberry Pi Foundation niminen hyväntekeväisyysjärjestö, joka alunperin perustettiin edistämään tietotekniikan opetusta kouluissa. Järjestön perustajajäsenet huomasivat 2000-luvulla aloittelevien tietotekniikan opiskelijoiden tietokonetaitojen laskeneen huomattavasti 1990-lukuun verrattuna, jolloin suurin osa tietotekniikkaa opiskelemaan hakevista nuorista olivat kokeneita harrastavia ohjelmoijia. [1]

Järjestön perustajajäsenet tunnistivat joukon tilanteeseen johtaneista ongelmista ja yksi näistä oli modernien PC:iden yleistyminen, jotka korvasivat vanhan ajan Amigat, BBC Microt, Spectrum ZX ja Commodore 64 -tietokoneet, joilla aiemmat sukupolvet oppivat aikanaan ohjelmoimaan. Moderneista tietokoneista oli tullut niin kalliita ja monimutkaisia, että nuorten vanhemmat kielsivät niillä ohjelmointikokeilut. Järjestön jäsenet halusivat toteuttaa alustan, joka vastaisi vanhan ajan kotitietokoneita, jotka pystyivät käynnistymään suoraan ohjelmointiympäristöön. [1]

2.2 Tekniset ominaisuudet

Aikaisimmat prototyypit perustuivat Atmelin ATmega644-mikrokontrolleriin, mutta lopulta mobiiliprosessoreiden kehittymisen ansiosta syntyi nykypäivän ARM-prosessoriin pohjautuva Raspberry Pi. Raspberry Pi:stä on kahta erilaista versiota, toinen on karsittu 25 dollarin A-versio ja toinen ominaisuuksiltaan parempi 35 dollarin B-versio. Tässä työssä käytettiin Raspberry Pi -tietokoneen B-mallia, joten työssä tullaan keskittymään siihen. [1]

Raspberry Pi sisältää kattavan määrän ominaisuuksia pakattuna hyvin edulliseen hintaan. Raspberry Pi on rakennettu Broadcomin BCM2835-järjestelmäpiirin ympärille, mikä sisältää 700 MHz ARM11-perheeseen lukeutuvan prosessorin ja 250 MHz kellotaajuudella toimivan Broadcomin VideoCore IV -näytönohjaimen. Muistia B-mallissa on 512 megatavua ja se on jaettu näytönohjaimen kanssa. [2]

Lisälaitteita varten Raspberry Pi:ssä on mm. USB 2.0 -portti, joka mahdollistaa esimerkiksi näppäimistön, hiiren tai bluetooth-adapterin liittämisen Raspberry Pi -tietokoneeseen. Tätä USB-porttia on mahdollista jatkaa USB-hubilla, joka mahdollistaa useamman lisälaitteen käyttämisen. On suositeltavaa käyttää erillisellä virtalähteellä varustettua USB-hubia, ettei Raspberry Pi:stä lopu virta kesken.

Kuvaa varten Raspberry Pi:ssä on RCA-komposiittiliitäntä sekä HDMI, joka tukee myös ääntä. Erityisesti ääntä varten on 3,5 mm jakki. Raspberry Pi:ssä ei itsessään ole minkäänlaista massamuistia vaan SD-muistikorttipaikka. Tälle muistikortille asennetaan käyttöjärjestelmä, joka vaatii tilaa noin kaksi gigatavua käyttöjärjestelmästä riippuen, jolloin neljän gigatavun muistikortti on suositeltava.

Elektroniikkaharrastajien iloksi Raspberry Pi -tietokoneessa on kaksi rivistöä täysin ohjelmoitavia GPIO-pinnejä, joiden avulla erilaisten elektronisten laitteiden ohjaaminen Raspberry Pi:n avulla on mahdollista. Näiden GPIO-pinnien sekaan on upotettu pinnejä erilaisia rajapintoja varten kuten; SPI, UART ja I²C. Nämä rajapinnat ovat oletuksena pois päältä ja tarvittaessa ne täytyy manuaalisesti kytkeä käyttöjärjestelmän kautta päälle erinäisillä komennoilla. [1]

Lopuksi Raspberry Pi:ssä on kameramoduulia varten CSI-liitäntä, mahdollista TFT-näyttöä varten DSI-liitäntä ja verkkosovittimena RJ45-liitännällä varustettu Ethernet-portti. Tarkemmat tekniset tiedot vielä molemmista Raspberry Pi:n malleista on esitetty alla olevassa taulukossa (taulukko 1).

TAULUKKO 1. Raspberry Pi:n tekniset tiedot. [1], [2]

	Malli A	**Malli B**
Hinta	25 $	35 $
Järjestelmäpiiri	Broadcom BCM2835	
Prosessori	700 MHz ARM1176JZF-S core	
Näytönohjain	Broadcom VideoCore IV @ 250 MHz	
Muisti	256 MB	512 MB
USB 2.0 -portit	1	2
Videosisääntulot	CSI-liitäntä	
Videoulostulot	Komposiitti RCA, HDMI, DSI	
Ääni	HDMI, 3,5 mm jakki	
Massamuisti	SD / MMC / SDIO -muistikortti	
Verkkosovitin	Ei ole	10/100 Ethernet (RJ45)
Matalan tason oheis-laitteet	8x GPIO, UART, I²C, SPI ja I²S audio	
Virrankulutus	300 mA (1,5 W)	700 mA (3,5 W)
Virtalähde	5 V MicroUSB	
Koko	85,60 mm x 56 mm	
Paino	45 g	
Käyttöjärjestelmä	Raspbian wheezy, Arch Linux ARM, Android, muut ARMv61 arkkitehtuuria tukevat käyttöjärjestelmät	

3 KÄYTTÖJÄRJESTELMÄ

Raspberry Pi on kehitetty Linux-ydintä käyttävät Unixin kaltaiset käyttöjärjestelmät mielessä. Linux on etenkin palvelimissa mahdollisesti maailman käytetyin käyttöjärjestelmä. Linuxin nimi tulee alun perin Linus Torvaldsin vuonna 1991 kehittämästä Linux-ytimestä. Linuxista käytetään myös nimeä GNU/Linux, sillä sekä käyttöjärjestelmä että sovellusohjelmistot sisältävät GNU-projektin tuottamia ohjelmistoja. [4]

Linux levitetään tuotteistettuina Linux-jakelupaketteina palvelin- ja työpöytäkäyttöön. Tällainen jakelupaketti on kokoelma erilaisia ohjelmistoja, jotka muodostavat kokonaisen käyttöjärjestelmän. Linux-jakelupaketti sisältää itse Linux-ytimen, käyttöjärjestelmäalusta ja erilaisia kirjastoja. Jakelupaketin tuottaja vastaa sen kehityksestä ja käyttäjätuesta. Suosittuja Linux-jakelupaketteja ovat mm. Debian, Ubuntu, Fedora ja openSUSE.

3.1 Raspbian-jakelupaketti

Raspberry Pi tukee periaatteessa kaikkia sen prosessoriarkkitehtuuria tukevia käyttöjärjestelmiä kuten; Arch Linux ARM, RISC OS, Android, openSUSE ja Debian ARM, mutta tässä työssä käytettiin Raspberry Pi -säätiön suosittelemaa Raspbian-jakelupakettia. Raspbian on Raspberry Pi:lle Debian wheezystä johdettu täysin ilmainen käyttöjärjestelmä. [5]

Kyseinen käyttöjärjestelmä sisältää lukusia perusohjelmia ja apuohjelmia, joiden avulla Raspberry Pi:n ominaisuudet ovat heti käytettävissä käyttöjärjestelmän asennuksen jälkeen. Näihin sovelluksiin lukeutuu erilaisia arkikäyttöön tarkoitettuja perusohjelmia esim. verkkoselain ja ohjelmointiin tarkoitettuja erikoisohjelmia kuten Python.

Raspbian asennetaan FAT32 alustetulle SD-muistikortille kirjoittamalla Raspbianin kuvatiedosto suoraan muistikortille käyttäen kuvatiedoston kirjoitusohjelmistoa. Windows-käyttöjärjestelmällä suositellaan käytettäväksi Win32diskimager-ohjelmaa ja Mac-käyttöjärjestelmällä Raspberry Pi SD Card Builder -ohjelmaa. Muistikortin alustamiseen Raspberry Pi -säätiö suosittelee SD Association -yhdistyksen SD Formatter

4.0 -ohjelmistoa. Tässä työssä käyttöjärjestelmän asentamiseen käytettiin NOOBS nimistä asennusohjelmistoa, joten Raspbianin asentamista ei käsitellä tämän tarkemmin. NOOBS-ohjelmistosta ja sen asennusvaiheista on kerrottu tarkemmin seuraavassa kappaleessa 3.2. [1]

3.2 NOOBS-ohjelmisto

NOOBS tulee englannin kielen sanoista "New Out Of Box Software", joka karkeasti suomennettuna tarkoittaa heti käyttövalmista ohjelmistoa. NOOBS on Raspberry Pi -säätiön kehittämä asennustyökalu helpottamaan yhden tai useamman käyttöjärjestelmän samanaikaista asentamista aloittelijoille. [1]

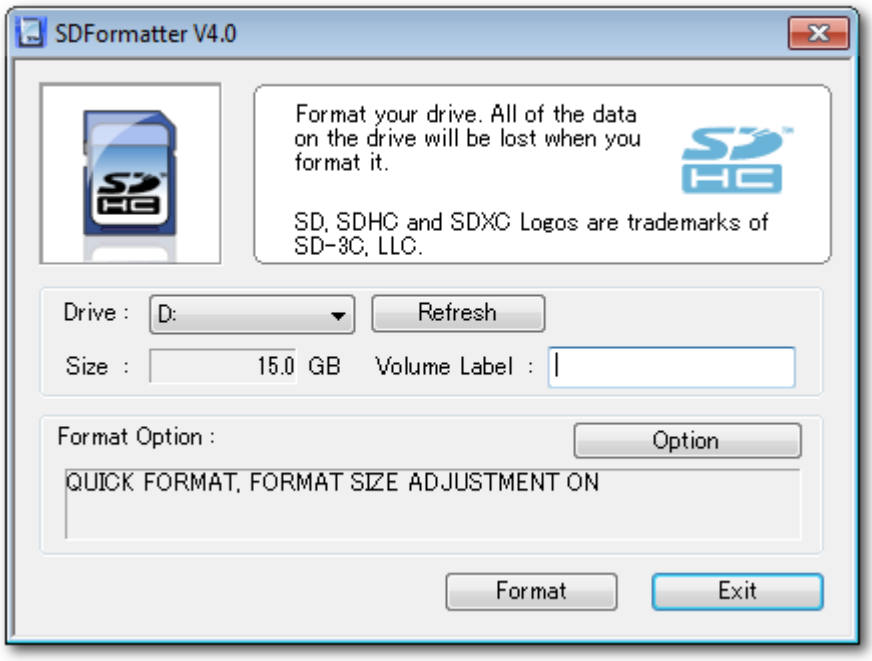
Kirjoitushetkellä NOOBS sisältää asennustyökalut; Archlinuxin, openELEC:n, Pidoran, RaspBMC:n, Raspbianin ja Risc OS:n asentamiseen. Käyttöjärjestelmän asennuksen jälkeen NOOBS jää muistikortin käynnistysosiolle ja tarjoaa laitteen käynnistysvaiheessa palautustilan, kun näppäimistöltä pidetään shift-näppäin pohjassa. Palautustilan avulla käyttöjärjestelmä pystytään helposti vaihtamaan toiseen käyttöjärjestelmään tai asentamaan uudelleen mahdollisen vikatilanteen sattuessa. Palautustila tarjoaa myös kätevän työkalun asennetun käyttöjärjestelmän config.txt sekä cmdline.txt -tiedostojen muokkaamiseen ja jopa verkkoselaimen.

3.2.1 Asentaminen

Ennen NOOBS-ohjelmiston asentamista tarvitaan vähintään neljän gigatavun muistikortti mielellään mahdollisimman nopeaa nopeusluokkaa. Suositeltu miniminopeus muistikortille on class 4, mikä tarkoittaa 4 MB/s kirjoitusnopeutta. Tässä työssä käytettiin 16 gigatavun class 10 (10 MB/s) SDHC-muistikorttia. Asentamiseen tarvitaan myös Windows tai Mac -pohjainen PC muistikortinlukijalla tai USB:hen liitettävällä muistikorttiadapterilla.

Ensimmäiseksi SD-muistikortti alustetaan käyttäen SD Association -yhdistyksen kehittämää SD Formatter -työkalua, jonka voi ladata heidän kotisivuiltaan. [6] SD Formatter -työkalun käynnistyttyä ilmestyy seuraavalla sivulla olevan kuvan (kuva 2) mukainen

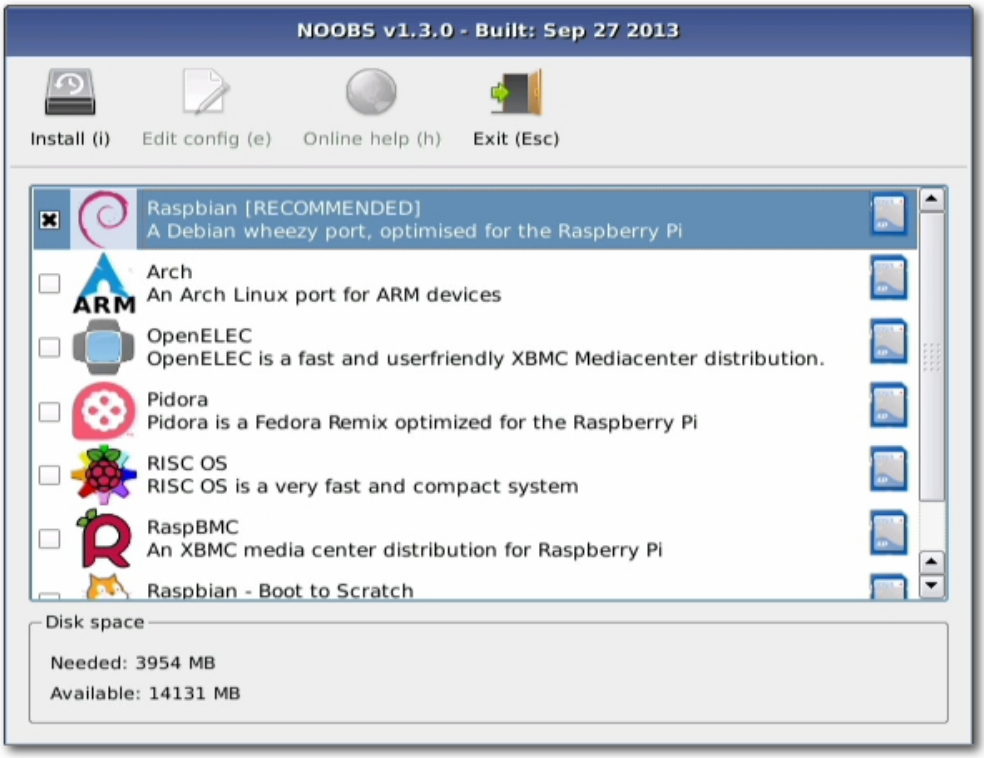
ikkuna, josta alustamista varten valitaan aseman kirjain (SD-muistikortti) "Drive"-kohtaan ja "Option"-valikosta valitaan "Format Size Adjustment ON". Tämän jälkeen klikataan "Format"-näppäintä ja SD-muistikortti on käyttövalmis.



KUVA 2. SD Formatter 4.0 -työkalun käyttöliittymä. [7]

Seuraavaksi ladataan Raspberry Pi -säätiön kotisivuilta NOOBS-ohjelmiston asennustiedosto, joko verkon ylitse tapahtuva asentaja tai ns. verkoton valmispaketti, joka sisältää koko NOOBS:n pakattuna yhteen zip-tyyppiseen tiedostoon. [1] Tässä työssä käytettiin verkotonta ratkaisua. Ladattu zip-tiedosto puretaan alustetun muistikortin juureen. Tämän jälkeen NOOBS on Raspberry Pi -tietokonetta varten valmis.

Muistikortti asetetaan virrattomaan Raspberry Pi -tietokoneeseen ja tässä vaiheessa on hyvä varmistaa että Raspberry Pi:hin on kytketty vähintään näppäimistö ja näyttö, jotta asennusvalikossa voidaan navigoida. Raspberry Pi:n käynnistyttyä näytöllä näkyy Raspberry Pi:n logo ja pieni laatikko, jossa asentaja ajaa automaattisesti asennuksen kannalta tarpeellisia toimintoja. Tämän jälkeen näytölle tulee näkyviin asennusvelho, jossa on listattuna kaikki käyttöjärjestelmät, jotka ovat kyseisessä NOOBS:n versiossa asennettavissa. NOOBS:n asennusvelho on esitetty seuraavalla sivulla olevassa kuvassa (kuva 3).

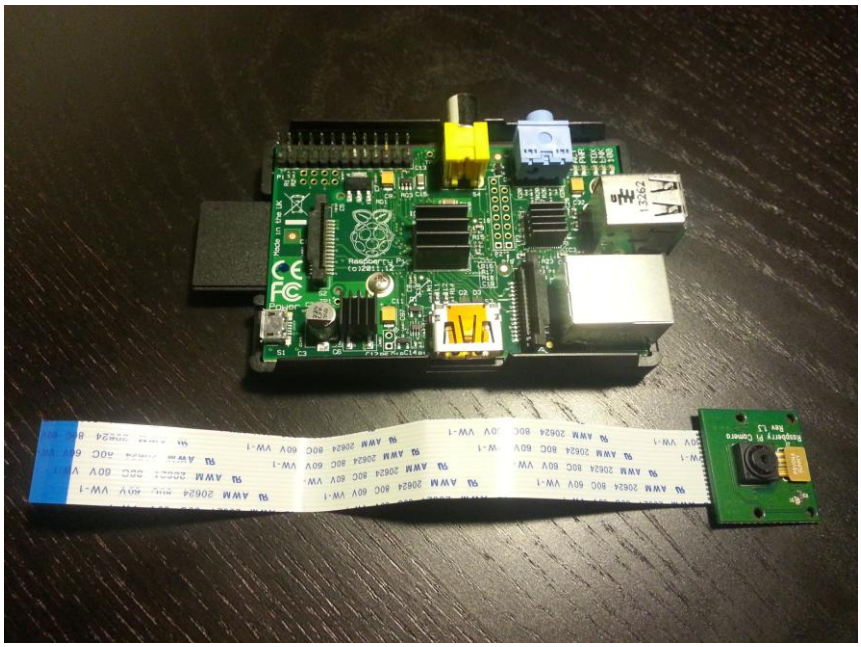


KUVA 3. NOOBS-ohjelmiston asennusvelho. [7]

Asennusvelhosta laitetaan ruksi niiden käyttöjärjestelmien kohdalle mitkä halutaan asentaa. Tässä työssä asennettiin ainoastaan Raspbian. Asennusvelhon alareunassa näkyy valituille käyttöjärjestelmille vaadittu tila ja käytettävissä oleva tila muistikortilla. Kun käyttöjärjestelmä tai käyttöjärjestelmät on valittu, niin asennusvelhon yläreunasta painetaan "Install"-näppäintä ja ilmestyvään varmistusviestiin painetaan "Yes"-näppäintä. Tämän jälkeen asentaja aloittaa käyttöjärjestelmän asennusprosessin ja tässä vaiheessa yleensä kestää useampikin minuutti. Asennuksen valmistuttua Raspberry Pi on käyttövalmis.

4 KAMERA

Raspberry Pi:lle on kehitetty aloittelijoille helppokäyttöinen kameramoduuli, joka tarjoaa myös edistyneille käyttäjille runsaasti toimintoja. Kyseisellä kameramoduulilla kyetään ottamaan teräväpiirtotason videoita ja kuvia. Tässä työssä päädyttiin käyttämään kyseistä kameramoduulia, koska kameramoduulin tuottama kuva prosessoidaan Raspberry Pi:n näytönohjaimella toisin kuin perinteisten USB-kameroiden tuottama kuva, joka prosessoidaan prosessorilla. Tällöin Raspberry Pi:n prosessori pystyttiin keskittämään laskemaan kuvantunnistusalgoritmeja. Raspberry Pi:n kameramoduuli on esitetty alla olevassa kuvassa (kuva 4).



KUVA 4. Raspberry Pi:n kameramoduuli (alhaalla) ja Raspberry Pi (ylhäällä).

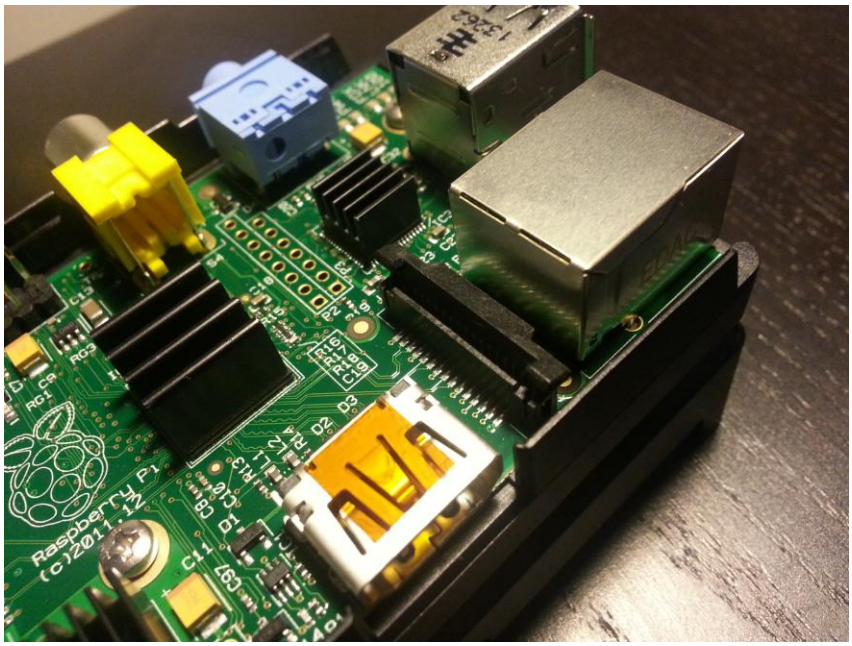
Kameramoduuli sisältää Omnivisionin valmistaman 5-megapikselin OV5647-kuvasensorin. Kyseisen kuvasensorin avulla kameramoduuli tukee 1080p-tasoista videokuvaa 30 ruutua/sekunti, 720p-tasoista videokuvaa 60 ruutua/sekunti ja VGA90-videotilaa, jonka avulla voidaan ottaa videokuvaa 90 ruutua/sekunti. Kameramoduuli tukee myös valokuvia. Suurin mahdollinen resoluutio kameramoduulilla on 2592x1944 1-15 ruutua/sekunti riippuen otetaanko videota vai valokuvia. [1]

Kameramoduuli liitetään Raspberry Pi:n CSI-porttiin 15 cm nauhakaapelilla ja Raspberry Pi:llä sitä käytetään MMAL tai V4L -ohjelmointirajapintojen avulla. Kameramoduu-

lille on olemassa myös lukuisia kolmannen osapuolen kirjastoja esimerkiksi PiCamera-kirjasto kameramoduulin ohjaamiseen Pythonin avulla. [1]

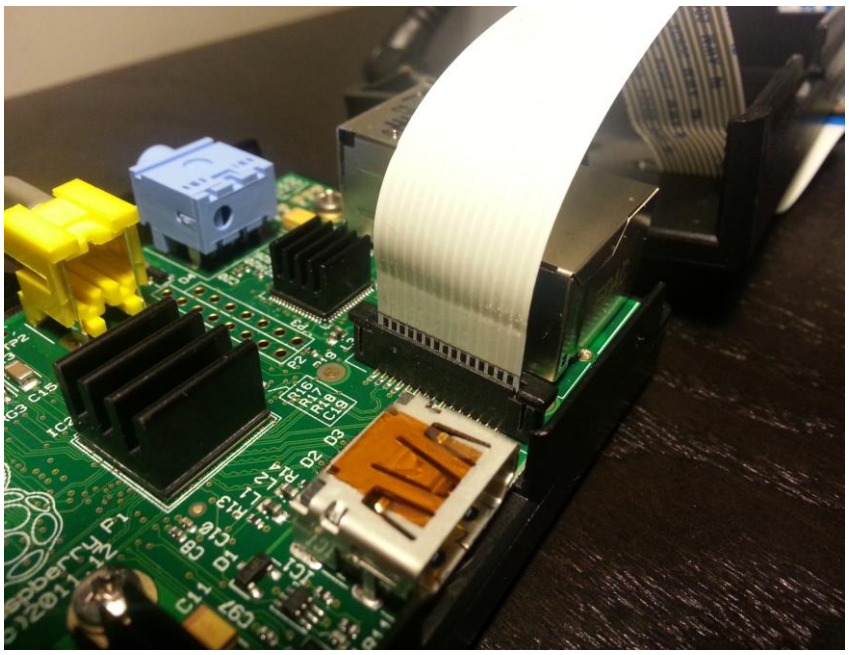
4.1 Asennus ja käyttöönotto

Raspberry Pi:n kameramoduuli on herkkä staattiselle sähkölle, joten sitä tulee käsitellä varoen. Kameramoduuli liitetään virrattoman Raspberry Pi:n ethernet- ja HDMI-portin välissä sijaitsevaan CSI-porttiin. Portin liittimessä on lukitusmekanismi, joka avataan nostamalla liittimen molemmilla puolilla olevista hakasista yhtaikaa ylöspäin. Portin lukitusmekanismi tällöin nousee ja taittuu hieman sivulle. Aukaistu CSI-portin liitin on esitetty alla olevassa kuvassa (kuva 5).



KUVA 5. Aukaistu CSI-portin liitin (keskellä).

Kameramoduulin kaapeli asetetaan liittimeen siten, että kaapelin foliot ovat HDMI-porttiin päin. Kaapeli tulee pitää paikallaan ja kaapelin foliot täysin suorassa suhteessa liittimeen samalla kun liittimen lukitusmekanismi suljetaan painamalla molempia hakasia yhtaikaa alaspäin. Jos jokaisesta kaapelin foliosta on asennuksen jälkeen tasaisesti näkyvissä noin puoli millimetriä, niin asennus on onnistunut. CSI-porttiin asennettu kameramoduulin kaapeli on esitetty seuraavalla sivulla olevassa kuvassa (kuva 6).



KUVA 6. Asennettu kameramoduulin kaapeli CSI-porttiin.

Jotta kameramoduulia voidaan käyttää asennuksen jälkeen Raspberry Pi:llä, niin Raspbianin komentorivin kautta mennään asetuksiin komennolla "sudo raspi-config". Asetuksista valintaan "Enable/Disable Camera Addon Support" asetetaan "Enable" ja Raspberry Pi käynnistetään uudelleen. Tämän jälkeen kameramoduuli on toimintakunnossa.

Kameramoduulia voidaan oletuksena ohjata Raspbianissa komentorivillä käyttäen raspistill ja raspivid -komentoja. Raspistill tarjoaa työkalut valokuvien ottamiseen ja raspivid videokuvan ottamiseen. Asennuksen jälkeen on hyvä testata kameran toimivuutta esimerkiksi komennolla "raspistill -v -o test.jpg", jolloin näytöllä näkyy viiden sekunnin ajan kuvaa kameralta samalla näyttäen komentorivillä statistiikkaa kameran toiminnallisuudesta, jonka jälkeen kamera ottaa kuvan ja tallentaa sen nimellä "test.jpg".

5 KUVANTUNNISTUS

Kuvantunnistus on menetelmä, jonka avulla määritellään sisältääkö kuva jonkin tietynlaisen objektin, ominaisuuden tai toimintaa. Kuvantunnistus sujuu luonnostaan ihmisiltä, mutta tietokoneille se on edelleen nykypäivänäkin suuri haaste, vaikka tietokoneet ovat kehittyneet huomattavasti. Tietokoneen näkemisen kehitystä on hidastanut näön biologisen mekanismin tuntemattomuus, joka tekee sen jäljittelemisestä hankalaa. Ihminen kykenee katsomaan kuvaa ja tunnistamaan helposti mitä se esittää, vaikka kuva olisi puutteellinen.

Tietokoneet eivät ole lähellekään näin hyviä mitä tulee kuvantunnistukseen, mutta toisaalta tietokoneet pystyvät näkemään erikoiskameroilla sellaisia valon aallonpituuksia mitä ei ihmissilmä kykene näkemään, kuten infrapuna. Tästä syystä tietokoneilla on nykypäivänä automatisoitu lukuisia erilaisia teollisuuden sovelluksia, joissa täytyy esimerkiksi tarkistaa kappaleista poikkeamia, joita ei ihmissilmä kykenisi näkemään luotettavasti. Tällaista tietokoneen ja kuvantunnistuksen kokonaisuutta kutsutaan tavallisesti tietokone- tai konenäöksi.

Tyypillisessä kuvantunnistussovelluksessa hankitaan aluksi kameralta kuva, joka muunnetaan sellaiseen muotoon, jota voidaan ohjelmallisesti käsitellä. Käsitellystä kuvasta suodatetaan sellainen tieto mitä ei haluta, kuten kohina. Suodatuksen jälkeen kuvasta haetaan haluttu asia siihen soveltuvalla algoritmilla ja ohjelmallisesti ilmaistaan, että kuvasta löydettiin haettu asia.

Hyvä esimerkki tyypillisestä kuvantunnistussovelluksesta arkipäiväisessä käytössä on digitaalikameroiden kasvojen tunnistus, jossa kamera piirtää näytölle suorakulmion tunnistettujen kasvojen ympärille. Tällainen kasvojentunnistussovellus hakee algoritmilla kuvasta tiettyjä kasvoille ominaisia kuvioita määrittääkseen onko kuvassa kasvot vai ei. Kasvojentunnistusalgoritmit ovat laskennallisesti hyvin vaativia varsinkin jos kuvasta täytyy vielä tunnistaa tietty henkilö. Tällöin tietokoneen täytyy vertailla jokaista kuvaa tietokannassa määritettyjen henkilöiden kuviin, joita yhdestä henkilöstä on tavallisesti useampi. Muita kuvantunnistusmetodeja ovat mm. kohteentunnistus, liikkeentunnistus ja reunantunnistus.

6 OPENCV-KIRJASTO

OpenCV tulee englannin kielen sanoista "Open Source Computer Vision Library" ja suomennettuna tarkoittaa avoimen lähdekoodin tietokonenäkökirjastoa. OpenCV kehitettiin tarjoamaan yleinen infrastruktuuri tietokonenäkösovelluksia varten ja kiihdyttämään konenäön käyttöä kaupallisissa tuotteissa. Tietokonenäön kehittäminen kun on hyvin työlästä, jos kehittäjän täytyy itse suunnitella ja toteuttaa tarvittavat algoritmit. Tässä työssä käytettiin OpenCV:tä toteutetussa kohteen seuranta -sovelluksessa. [8]

6.1 Yleistä OpenCV:stä

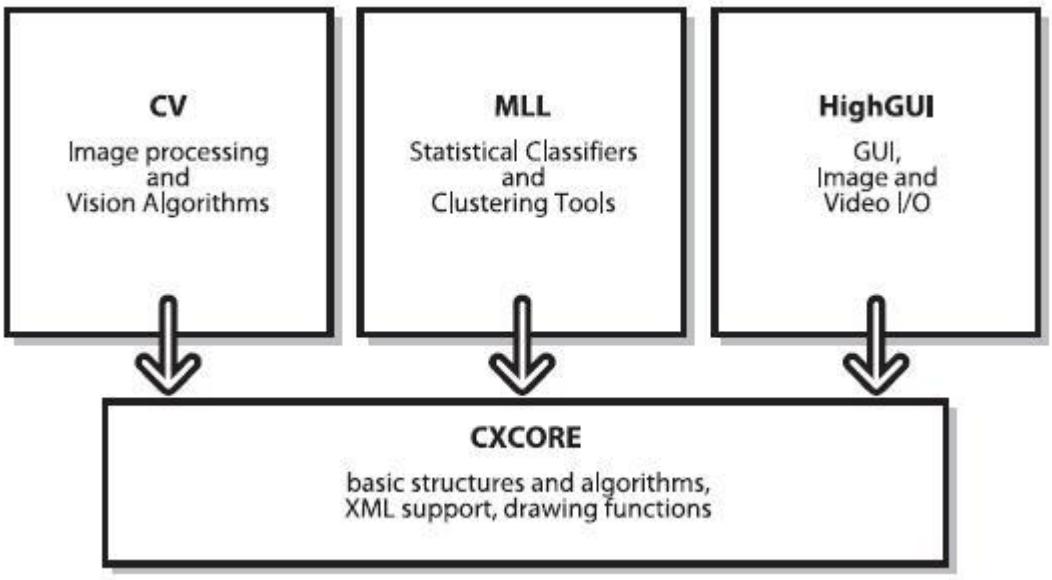
Kyseiseen kirjastoon sisältyy yli 2500 optimoitua algoritmia, jotka sisältävät kattavan joukon sekä klassisia että viimeisintä tekniikkaa edustavia konenäön ja koneoppimisen algoritmeja. Näitä algoritmeja voidaan käyttää tunnistamaan kasvoja, tunnistamaan esineitä, luokittelemaan ihmisten eleitä videosta, jäljittämään kameran liikkeitä, jäljittämään liikkuvia kohteita, purkamaan esineestä 3D-mallin, etsimään yhtäläisyyksiä kuvatietokannasta, poistamaan punasilmät valokuvasta, jäljittämään silmien liikettä, tunnistamaan maisemia ja niin edelleen. [8]

OpenCV:llä on yli 47000 käyttäjän yhteisö ja latauskertojen arvioidaan ylittäneen 7 miljoonaa latausta. Kirjastoa käytetään nykypäivänä kattavasti yritysten, tutkimuslaitosten ja julkisten laitosten toimesta. Käytössä olevat sovellukset kattavat sellaisia sovelluksia kuten katukuvien yhdistäminen toisiinsa, tunkeutumisten tunnistaminen videolta Israelissa, kaivosvarusteiden monitorointi Kiinassa, robottien navigoinnin helpottaminen Willow Garagella, uima-altaiden hukkumisonnettomuuksien tunnistaminen Euroopassa, interaktiivista taidetta Espanjassa sekä New Yorkissa, roskien tunnistaminen jalkakäytäviltä Turkissa, tuotetarrojen tarkistaminen tehtaissa ympäri maailmaa ja nopeaa kasvojentunnistusta Japanissa. [8]

OpenCV:llä on rajapinnat C++, C, Python sekä MATLAB -ohjelmointiympäristöjä varten ja se tukee Windows, Linux, Android sekä Mac OS -käyttöjärjestelmiä. OpenCV painottuu reaaliaikaisiin tunnistussovelluksiin ja se kykenee käyttämään hyväksi MMX ja SSE -käskykantoja. [8]

6.2 OpenCV:n rakenne

OpenCV on rakennettu modulaariseksi, jolloin voidaan hyödyntää sen eri tasojen toimintoja. OpenCV koostuu viidestä pääkomponentista, joista neljä tärkeintä on esitetty alla olevassa kuviossa (kuvio 1).



KUVIO 1. OpenCV:n rakenne. [14]

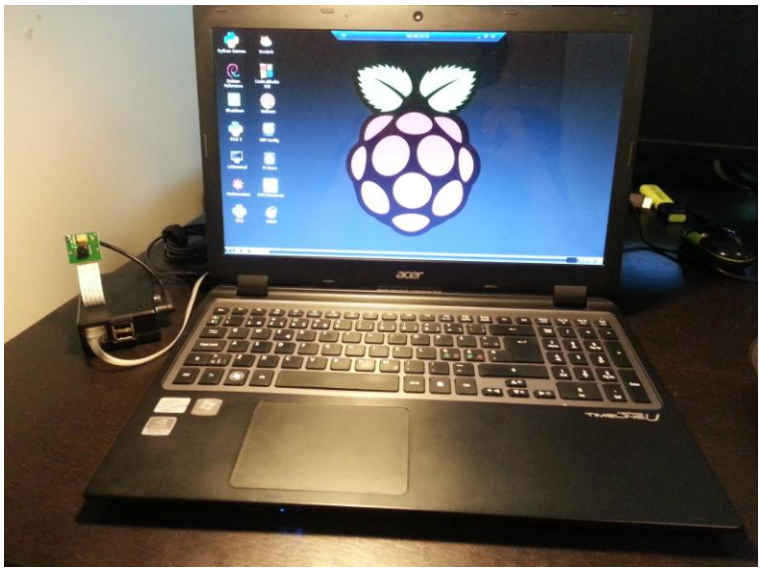
CV-moduuli sisältää kuvankäsittelyn perustoimintoja ja korkean tason tietokonenäön algoritmeja. MLL on koneoppimisen kirjasto ja se sisältää monia tilastollisia luokittelijoita ja klusterointityökaluja. HighGUI sisältää I/O-rutiinit ja toiminnot videon sekä kuvien tallentamiseen että lataamiseen. CXCore sisältää tietorakenteet ja sisältöä tietotyyppien käytön helpottamiseksi. Viides moduuli, mitä ei yllä olevassa kuviossa (kuvio 1) ole sisällytetty, on CvAux, joka sisältää kokeellisia ja toimimattomia algoritmeja. Dokumentointi CvAux-moduulista on hyvin vähäistä, eikä sitä suositella käytettäväksi muuta kuin OpenCV:n kehitystyöhön. [14]

7 KOHTEEN SEURANTA

Työssä toteutettiin kuvantunnistussovellus, joka kykenee tunnistamaan ja seuraamaan videokuvasta ennalta määriteltyjä kohteita värien perusteella. Työssä päädyttiin käyttämään C++-ohjelmointikieltä, koska Raspberry Pi:n kameramoduulin ohjaamisen käytetyt ohjelmistot oli kirjoitettu kyseisellä ohjelmointikielellä. Tässä luvussa on käyty läpi kohteen seurannan toimintaan saamisen kannalta kaikki oleelliset asiat ja selvitetty sovelluksen toimintaperiaate.

7.1 Laitteistokokoonpano

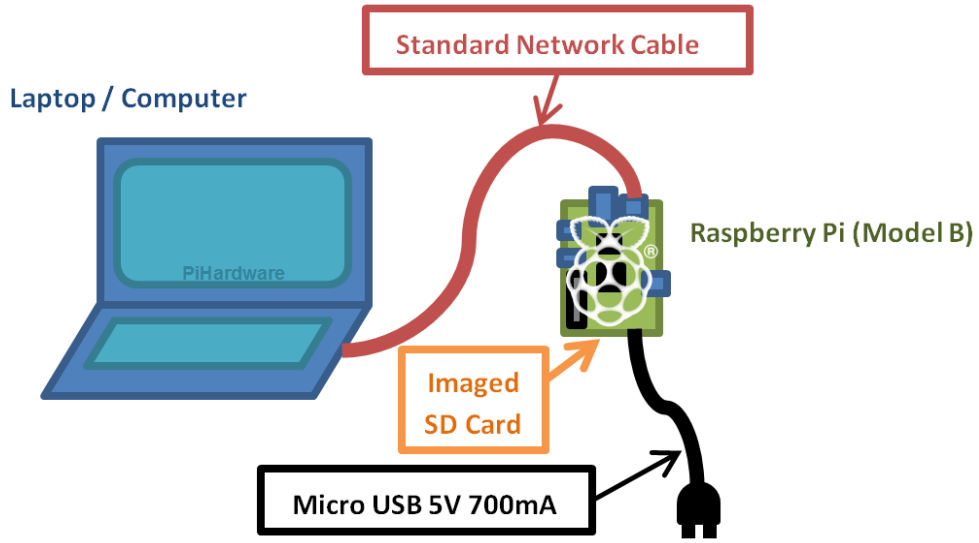
Sovelluksen laitteistokokoonpano koostui; Raspberry Pi -tietokoneen B-mallista, 16 gigatavun class 10 -muistikortista, Raspberry Pi:n kamerasta, USB-virtalähteestä (5,25 V ja 2 A), ethernet-kaapelista RJ45-liittimillä ja kannettavasta tietokoneesta Windows 8.1 -käyttöjärjestelmällä. Sovelluksen kehitysvaiheessa Raspberry Pi:tä käytettiin kannettavan tietokoneen kautta suoralla etäyhteydellä. Raspbianin asennuksessa sekä suoran etäyhteyden luomisvaiheessa Raspberry Pi:tä käytettiin HDMI-liittimellä varustetulla näytöllä, USB-näppäimistöllä ja USB-hiirellä. Edellä mainitussa vaiheessa internetyhteys hoitui Wi-Fi -adapterin avulla ja USB-lisälaitteet oli kytketty Raspberry Pi:hin virtalähteellä varustetun USB-hubin avulla. Sovelluksessa käytetty laitteistokokoonpano on esitetty alla olevassa kuvassa (kuva 7).



KUVA 7. Käytetty laitteistokokoonpano.

7.2 Suora etäyhteys

Suoraa etäyhteyttä käytettiin helpottamaan sovelluksen kehittämistä. Suoran etäyhteyden avulla pystyttiin käyttämään kannettavan tietokoneen näyttöä, näppäimistöä, hiirtä ja verkkoyhteyttä Raspberry Pi:n kanssa sekä hyödyntämään kannettavan tietokoneen ohjelmointityökaluja sovelluksen kehityksessä. Tällöin Raspberry Pi:tä pystyttiin käyttämään lähes missä vain, kun ei tarvinnut kannettavan tietokoneen lisäksi kantaa mukana muita lisälaitteita. Tässä kappaleessa on käyty lävitse kuinka suora etäyhteys saatiin toimimaan. Havainnoiva kuva suorasta etäyhteydestä on esitetty alla olevassa kuvassa (kuva 8).



KUVA 8. Suora etäyhteys. [11]

Aluksi selvitettiin käytetyn kannettavan tietokoneen langallisen verkkoadapterin IP-osoite ja jaettiin Wi-Fi -adapterin verkkoyhteys. Kannettavan tietokoneen verkkoasetuksista mentiin Wi-Fi -adapterin ominaisuuksiin ja sieltä jakamisasetuksiin. Jakamisasetuksista laitettiin ruksi kohtaan, joka sallii muiden verkkokäyttäjien yhdistää tietokoneen internet-yhteyden kautta. Langallisen ethenet-adapterin ominaisuuksista katsottiin sen IPv4 osoite.

Nyt kun tiedettiin kannettavan tietokoneen langallisen ethernet-adapterin IP-osoite ja internet-yhteys oli jaettu muille verkon käyttäjille, niin Raspberry Pi:lle täytyi asettaa IP-osoite. Raspberry Pi:n IP-osoite asetettiin muokkaamalla "cmdline.txt"-tekstitiedostoa, joka sijaitsee Raspbianin käynnistysosiossa. Kyseisen tekstitiedoston loppuun lisättiin "ip=192.168.137.10::192.168.137.1". Näistä kahdesta IP-osoitteesta

ensimmäinen on Raspberry Pi:lle asetettava osoite ja toinen kannettavan tietokoneen IP-osoite. Raspberry Pi:n osoite tuli olla samassa osoiteavaruudessa kannettavan tietokoneen osoitteen kanssa. Eli IP-osoitteen kolme ensimmäistä numerosarjaa tuli täsmätä kannettavan tietokoneen osoitteen kanssa. Tämän jälkeen suora etäyhteys oli käyttövalmis ja kannettavan tietokoneen internet-yhteys oli jaettu Raspberry Pi:n kanssa. Etäyhteyden käyttämiseen käytettiin Windowsin Remote Desktop Connection -työkalua. [11]

Helpottamaan ohjelmointia Raspberry Pi:n tiedostohakemisto jaettiin kannettavan tietokoneen kanssa. Tiedostohakemiston jakamiseen käytettiin Samba nimistä ohjelmistoa, koska se tukee Windows Internet Name Service -palvelua (WINS) ja Windowsin workgroup-työryhmiä. Ennen Samban asentamista päivitettiin Raspbianin pakettilistat komennolla "sudo apt-get update", jolloin ohjelmia asentaessa Raspberry Pi löytää viimeisimmät versiot ohjelmista. Samba asennettiin kirjoittamalla Raspberry Pi:n komentoriville "sudo apt-get install samba samba-common-bin", jolloin Raspberry Pi hakee automaattisesti asennuspaketit palvelimelta ja asentaa ne.

Asennuksen valmistuttua Samban asetuksia säädettiin kirjoittamalla komentoriville "sudo leafpad /etc/samba/smb.conf &", jolloin Samban asetustiedosto aukeaa Leafpad nimisessä tekstinkäsittelyohjelmistossa. Asetustiedostossa kirjoitettiin "workgroup"-kohtaan kannettavan tietokoneen workgroup-työryhmän nimi ja "wins support"-kohtaan "yes". Tekstitiedostoon lisättiin myös "Share Definitions"-kohdan alle seuraavat rivit:

[pihome]
comment= Pi Home
path=/home/pi
browseable=Yes
writeable=Yes
only guest=no
create mask=0777
directory mask=0777
public=no

Edellä mainituilla riveillä Samba loi lähiverkkoon pihome nimisen tiedostohakemiston, joka sisälsi kaikki Raspberry Pi:n osoitteessa "/home/pi" sijaitsevat tiedostot. Lähiverkossa näkyvä hakemisto oli salasanasuojattu ja salasana asetettiin kirjoittamalla komen-

toriville komento "smbpasswd -a pi". Tämän jälkeen jaettu tiedostohakemisto oli käytettävissä kannettavan tietokoneen tiedostoselaimen avulla. [12]

7.3 OpenCV-kirjaston asentaminen ja käyttöönotto

OpenCV-kirjaston C++-versio asennettiin Raspbianille kirjoittamalla komentoriville komento "sudo apt-get install libopencv-dev". Kirjoitushetkellä kyseisellä komennolla ja päivitetyillä pakettilistoilla Raspbian asensi OpenCV:n version numero 2.3.

OpenCV tukee natiivisti USB-kameroita, mutta ei tue suoraan Raspberry Pi:n kameramoduulia. Tästä muodostui työn alkuvaiheessa ongelma, mutta onneksi Pierre Raufaust oli ratkaissut tämän ongelman Think RPI -blogissaan. [3] Pierre oli ottanut Raspberry Pi:n userland-kirjaston lähdekoodin, joka sisältää myös lähdekoodit raspistill sekä raspivid -työkaluille, ja muokannut Raspberry Pi:n kameramoduulin ohjaamisen käytettävän osuuden tuottamaan OpenCV-kuvia. [13] Käytännössä Pierre oli muokannut kameran kuvapuskurin syöttämään OpenCV:n kuvaobjekteja.

Helpottamaan työskentelyä OpenCV:n ja Raspberry Pi:n kameramoduulin kanssa, Emil Valkov oli luonut robidouille-blogissaan Pierren tekemästä työstä erillisen kirjaston. [9] Tätä Emilin luomaa RaspiCamCV-kirjastoa käytettiin tässä työssä. RaspiCamCV-kirjasto sisältää Raspberry Pi:n kameramoduulille toiminnallisuudet, jotka korvaavat OpenCV:n HighGUI-moduulin vastaavat toiminnot kuvavirran lukemiseen kameralta.

Ennen RaspiCamCV-kirjaston asentamista täytyi ensiksi asentaa muutama ohjelma ja kirjasto. Raspbianin komentoriville kirjoitettiin komento "sudo apt-get install cmake git", jolloin Raspbian asentaa CMake-käännösjärjestelmän ja Git-versionhallintaohjelmiston. Ohjelmakoodin kääntämistä varten asennettiin gcc ja g++ -kääntäjät komennolla "sudo apt-get install gcc g++". Tarvittavat kirjastot asennettiin komennolla "sudo apt-get install libx11-dev libxt-dev libxext-dev libgraphicsmagick1-dev libcv-dev libhighgui-dev".

RaspiCamCV-kirjastoa varten hankittiin userland-kirjaston lähdekoodi kirjoittamalla Raspbianin komentoriville seuraavat komennot:

mkdir -p ~/git/raspberrypi
cd ~/git/raspberrypi
git clone https://github.com/raspberrypi/userland.git
cd userland
./buildme

Edellä mainituilla komennoilla luotiin Raspbianin käyttäjän tiedostohakemiston juureen "/git/raspberrypi" -tiedostohakemisto sekä kloonattiin palvelimelta userland-kirjaston lähdekoodi kyseiseen tiedostohakemistoon. Lopuksi userland-kirjaston lähdekoodi käännettiin ajamalla "userland"-kansiossa "buildme"-ohjelma. Tämän jälkeen hankittiin RaspiCamCV-kirjaston lähdekoodi kirjoittamalla komentoriville seuraavat komennot:

cd ~/git
git clone https://github.com/robidouille/robidouille.git
cd robidouille/raspicam_cv
mkdir objs
make

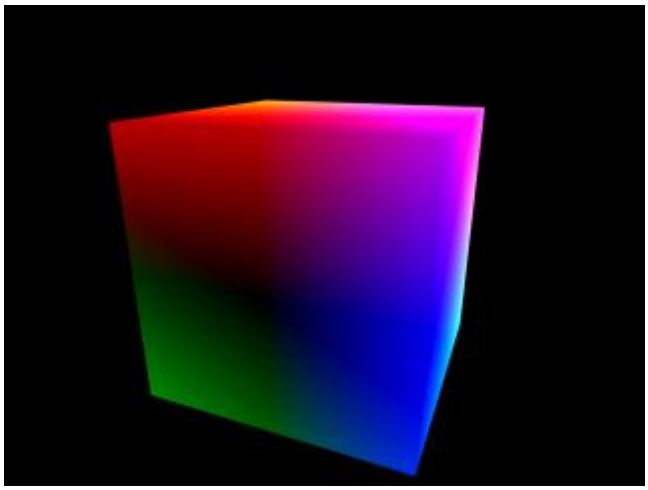
Edellä mainituilla komennoilla kirjaston lähdekoodi kloonattiin "git"-kansioon, lähdekoodin hakemistoon luotiin "objs"-kansio ja lopuksi "make"-komennolla kirjaston ohjelmakoodi käännettiin. Kirjaston ohjelmakoodin kääntäminen loi kirjaston hakemistoon "libraspicamcv.a"-tiedoston, joka sisältää OpenCV:n kaltaisen rajapinnan Raspberry Pi:n kameramoduulille, sekä "raspicamtest"-sovelluksen kirjaston toiminnallisuuden testaamista varten. Tämän jälkeen RaspiCamCV-kirjasto oli käyttövalmis.

7.4 Kuvien tallentaminen kuvavirrasta

Kameralta luettavasta kuvavirrasta kaapattavat kuvat tallennettiin OpenCV:n "Mat"-tyyppisiin muuttujiin. "Mat"-muuttuja on käytännössä luokka, joka sisältää matriisin tunnisteen ja pointterin kuvapikselien arvot sisältävään matriisiin. Tunniste sisältää mm. tiedot matriisin koosta, tallennustavasta ja matriisin osoitteesta. Eli helpommin sanottuna kuvat tallennetaan matriiseihin, joissa yksi solu sisältää yhden pikselin tiedot.

Kun tällainen matriisi syötetään algoritmille, niin algoritmi käy sen solu kerrallaan lävitse, jolloin mitä suurempi resoluutio tallennetulla kuvalla on, niin sitä hitaampaa algoritmin laskeminen on. Raspberry Pi:n prosessoritehot ovat hyvin rajalliset, joten työssä tuli optimoida kuvien tallentaminen kuvavirrasta.

Raaka kuvadata oli kameran puskurissa YUV I420 -formaatissa ja kuvat täytyi sovelluksen algoritmeja varten muuntaa RGB-värimallin muotoon. Tämä muunnosprosessi käyttää prosessoritehoa. Kun halutaan RGB-värimallin mukainen kuva, niin kuvapuskurista täytyy lukea kaikki kolme kuvakomponenttia. Kuvakomponenteista Y on luminanssi eli kirkkauskomponetti ja U sekä V ovat krominansseja eli värikomponentteja. YUV-väriavaruus on esitetty alla olevassa kuvassa (kuva 9). [15]



KUVA 9. YUV-väriavaruus sen pimeältä puolelta (Y=0). [15]

Mustavalkokuvaa varten tarvitaan ainoastaan Y-komponentti, mikä on huomattavasti nopeampaa kuin kaikkien kolmen komponentin käyttäminen. Valitettavasti mustavalkokuvan käyttäminen ei ollut mahdollista tässä työssä, koska kohteita etsittiin kuvasta värien perusteella. Tästä syystä suorituskyvyn optimoimiseksi täytyi säätää tallennettavien kuvien resoluutiota.

RaspiCamCV-kirjasto muuntaa oletuksena kuvat kameran puskurista RGB-muotoon ja tallentaa ne 640x480 resoluutiolla. Työssä näillä arvoilla ikkunassa näytettävälle videokuvalle ilman kuvantunnistusalgoritmeja mitattiin n. 6 ruutua/sekunti. Kun RaspiCamCV-kirjaston lähdekoodista muokattiin tallennusresoluutioksi 320x240, niin videokuvalle ilman kuvantunnistusalgoritmeja mitattiin n. 20 ruutua/sekunti. RaspiCamCV-kirjastoon tehdyt muutokset on esitetty alla olevassa kuvassa (kuva 10).

```
// default status
static void default_status(RASPIVID_STATE *state)
{
   if (!state)
   {
      vcos_assert(0);
      return;
   }

   // Default everything to zero
   memset(state, 0, sizeof(RASPIVID_STATE));

   // Now set anything non-zero
   state->finished          = 0;
   state->width             = 320;      // use a multiple of 320 (640, 1280)
   state->height            = 240;      // use a multiple of 240 (480, 960)
   state->bitrate           = 17000000; // This is a decent default bitrate for 1080p
   state->framerate         = VIDEO_FRAME_RATE_NUM;
   state->immutableInput    = 1;
   state->graymode          = 0;        // Gray (1) much faster than color (0)

   // Set up the camera_parameters to default
   raspicamcontrol_set_defaults(&state->camera_parameters);
}
```

KUVA 10. RaspiCamCV-kirjastoon tehdyt muutokset rajattuna suorakulmiolla.

Tämä oli huomattava suorituskykyparannus, joten kyseistä resoluutiota päätettiin käyttää työssä. Kokeilun vuoksi mitattiin myös käyttäen edellä mainittua resoluutiota ja mustavalkokuvaa. Tällöin suorituskyvyksi mitattiin 30 ruutua/sekunti, joka on Raspberry Pi:n kameramoduulin natiivi videokuvan päivitysnopeus 1080p-resoluutiolla. Kuvanpäivitysnopeuden mittaamiseen käytettiin Pierre Raufastin työtä, joka sisälsi koodin kuvan päivitysnopeuden laskemiseen. [3]

7.5 Toimintaperiaate ja rakenne

Sovelluksen toteuttamiseen käytettiin hyödyksi Kyle Hounslowin tekemiä OpenCV tutoriaaleja. [10] Sovellus rakennettiin suoraan Kyle Hounslowin kirjoittaman "objectTrackingTutorial.cpp"-tiedoston päälle. Toteutetun kohteen seuranta -sovelluksen lähdekoodi löytyy kokonaisuudessaan kommentoituna liitteestä (liite 1). Työssä käytettiin testitilanteena kuvaa, jossa oli kolmea erilaista kohdetta; punaisia palloja, keltaisia palloja ja sinisiä palloja.

Sovelluksen pääohjelmasilmukan alussa luodaan kolme matriisia; cameraFeed, threshold ja HSV. Näistä matriisesta cameraFeed on alkuperäistä kamerakuvaa varten, joka näytetään käyttäjälle, threshold-matriisi HSV-suodatettua kuvaa varten ja HSV matriisi RGB-väriavaruudesta HSV-väriavaruuteen muunnettavaa kuvaa varten. Mat-

riisien luomisen jälkeen luodaan objekti kuvien kaappaamiseksi kameran kuvavirrasta. Näiden tärkeiden alustusten jälkeen ohjelmassa siirrytään while-silmukkaan, joka jatkuu loputtomasti kunnes käyttäjä painaa tietokoneen näppäimistöltä mitä tahansa näppäintä.

While-silmukan alussa kaapataan kuva kuvavirrasta IplImage-tyyppiseen muuttujaan, joka sisältää itse kuvan ja lukusia tietoja kuvan ominaisuuksista. Kaapattu kuva välitetään cameraFeed-matriisin kautta OpenCV:n värimuunnos-funktiolle, joka muuntaa cameraFeed-matriisin sisältämän RGB-värimallisen kuvan HSV-värimalliin ja tallentaa sen HSV-matriisiin. Alla olevissa kuvissa on esitetty edellä mainittu ohjelmakoodi (kuva 11) ja cameraFeed sekä HSV -matriisien sisällöt tässä vaiheessa ohjelmakoodia (kuva 12).

```
// Luodaan matriisit kameralta kaapatulle ja suodatetuille kuville
Mat cameraFeed;
Mat threshold;
Mat HSV;
// Luodaan kamerakuvan kaappausobjekti, jolla hankitaan kameralta videokuvaa
RaspiCamCvCapture * capture = raspiCamCvCreateCameraCapture(0);
// Suoritetaan while-silmukkaa niin kauan kunnes näppäimistöltä painetaan jotakin näppäintä
while(cvWaitKey(10)<0)
{
    // Kaapataan kameran kuvavirrasta kuva
    IplImage* image = raspiCamCvQueryFrame(capture);
    // Tallennetaan kaapattu kuva cameraFeed matriisiin
    cameraFeed = image;
    // Muunnetaan tallenettu kuva RGB väriavaruudesta HSV väriavaruuteen
    // ja tallenetaan muunnettu kuva HSV matriisiin
    // HUOM! OpenCV:ssä oletuskuvaformaatti on kyllä RGB, mutta bitit ovat käänteisessä
    // järjestyksessä, jolloin ensimmäinen tavu on sininen, toinen vihreä ja kolmas punainen
    cvtColor(cameraFeed,HSV,COLOR_BGR2HSV);
```

KUVA 11. Kuvan kaappaaminen kameralta ja muunnos HSV-väriavaruuteen.

KUVA 12. CameraFeed (vasen) ja HSV (oikea) -matriisien sisällöt.

Usean kohteen samanaikaista tunnistusta varten ohjelmaan rakennettiin oliorakenne tunnistettaville kohteille, jotta kohteiden määrittely olisi ohjelmakoodissa nopeampaa ja helpompaa. Värimuunnoksen jälkeen silmukassa luodaan tunnistettaville kohteille oliot

ja asetetaan niille nimet. Koodi tunnistettavien kohteiden olioiden luontiin on esitetty alla olevassa kuvassa (kuva 13).

```
// Luodaan tunnistettaville kohteille oliot
Object Object0("Calibration"),Object1("Kel."),Object2("Sin."),Object3("Pun.");
```

KUVA 13. Määritellään tunnistettaville kohteille oliot.

Kuten yllä olevasta kuvasta nähdään, niin sovelluksessa luodaan neljä oliota, joista object0 on kalibrointitilaa varten, object1 keltaisia, object2 sinisiä ja object3 punaisia kohteita varten. Object-olion toteutus sisältää HSV-suodatusarvojen asettamisen määritetyille kohteille nimen perusteella sekä lukuisia funktioita olion muuttujien välittämiseen ja asettamiseen. Alla olevassa kuvassa (kuva 14) on esitetty olion toteutuksessa olevan HSV-suodatusarvojen asettavan funktion ohjelmakoodi.

```
// Määritellään kullekin kohteelle HSV arvot, joiden mukaan kohteet etsitään kuvasta
// Näitä kohteita voidaan helposti luoda lisää tai poistaa oliorakenteen ansiosta
Object::Object(string name)
{
    setType(name);

    if(name=="Calibration")
    {
        setHSVmin(Scalar(0,0,0));
        setHSVmax(Scalar(256,256,256));
        setColour(Scalar(0,0,0));
    }
    if(name=="Kel.")
    {
        setHSVmin(Scalar(16,165,68));
        setHSVmax(Scalar(256,228,256));
        setColour(Scalar(0,255,255));
    }
    else if(name=="Sin.")
    {
        setHSVmin(Scalar(0,0,0));
        setHSVmax(Scalar(256,256,106));
        setColour(Scalar(255,0,0));
    }
    else if(name=="Pun.")
    {
        setHSVmin(Scalar(0,150,73));
        setHSVmax(Scalar(15,256,256));
        setColour(Scalar(0,0,255));
    }
}
```

KUVA 14. Asetetaan tunnistettaville kohteille HSV-suodatusarvot olion toteutuksessa.

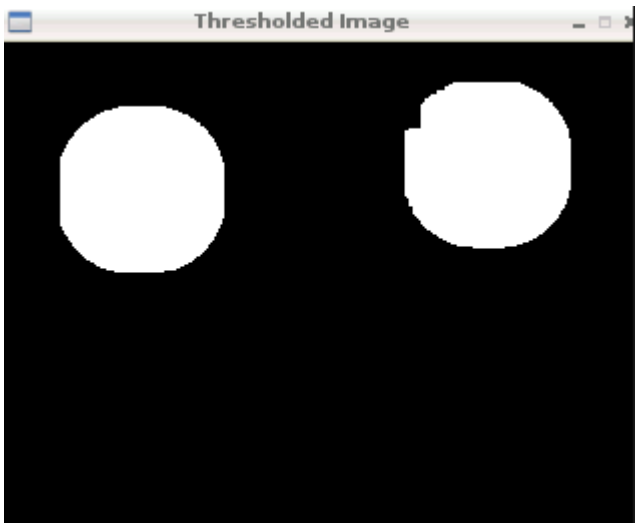
Olioiden luonnin jälkeen kutsutaan openCV:n inRange-funktiota. Kyseiselle funktiolle syötetään HSV-matriisi, HSV-suodatusarvot ja ulostuloa varten threshold-matriisi.

Funktio suodattaa HSV-matriisin sisältämästä kuvasta suodatusarvojen ulkopuolelle jäävät bitit pois, jolloin tuloksena saadaan kuva, josta on suodatettu pois kaikki sellaiset värit, jotka jäävät suodatusarvojen ulkopuolelle.

Tämä suodatettu threshold-matriisiin tallennettu kuva välitetään ohjelmakoodissa morphOps-aliohjelmalle, joka sisältää morphologisia operaatioita threshold-matriisin kuvalle. Kyseisessä aliohjelmassa luodaan kaksi rakentavaa elementtiä. Näistä elementeistä ensimmäinen on nimeltään erodeElement, jota käytetään poistamaan kuvasta kaikki 3x3-pikselin kokoiset objektit ja näin päästää eroon kuvassa olevasta kohinasta. Toinen elementti on nimeltään dilateElement, jota käytetään laajentamaan kaikki kuvaan jäljelle jääneet objektit 8x8-pikselin kokoisiksi. Tällöin kuvasta etsittävä kohde on selkeämmin erotettavissa. Nämä morphologiset toiminnot ovat OpenCV:n funktioita. Seuraavissa kuvissa on esitetty morphOps-aliohjelman ohjelmakoodi (kuva 15) ja morphologisilla operaatioilla käsitelty HSV-suodatettu kuva threshold-matriisissa (kuva 16).

```
// Funktio morphologisille operaatioille
void morphOps(Mat &thresh)
{
    // Luodaan kaksi rakentavaa elementtiä
    Mat erodeElement = getStructuringElement( MORPH_RECT,Size(3,3));
    Mat dilateElement = getStructuringElement( MORPH_RECT,Size(8,8));

    // Rappeuttamalla kuvaa siitä karsitaan kaikki objektit
    // jotka ovat kooltaan alle 3x3 pikseliä
    // Näin päästään eroon kohinasta
    erode(thresh,thresh,erodeElement);
    erode(thresh,thresh,erodeElement);

    // Laajennetaan jäljelle jääneet objektit
    // jotta kuvasta haettava kohde on selkeämmin erotettavissa
    dilate(thresh,thresh,dilateElement);
    dilate(thresh,thresh,dilateElement);
}
```

KUVA 15. MorphOps-aliohjelman ohjelmakoodi.



KUVA 16. Morphologisilla operaatioilla käsitelty HSV-suodatettu kuva.

Huomataan kuinka yllä olevassa kuvassa (kuva 16) näkyy ainoastaan punaiset pallot valkoisella ja muut värit mustalla. Tämä johtuu siitä että tässä vaiheessa ohjelmakoodia HSV-kuva on suodatettu ainoastaan yhden kohteen suodatusarvoilla. HSV-kuva voidaan suodattaa vain yhden kohteen suodatusarvoilla kerrallaan. Nämä suodatusarvot toimivat raja-arvoina, jolloin raja-arvojen sisäpuolelle jäävät värit näkyvät kuvassa valkoisena ja ulkopuolelle jäävät mustina. Tällöin kuva on binäärinen, jolloin pikseleillä on arvoina ykkönen tai nolla.

Morphologisesti käsitelty HSV-suodatettu kuva välitetään trackFilteredObject-aliohjelmalle, joka sisältää toiminnot kohteiden paikantamiseksi kuvasta. Aliohjelman alussa luodaan vector-tyyppinen muuttuja, johon löydettyjen kohteiden parametrit työnnetään. Tämän lisäksi luodaan findContours-funktiota varten tarpeelliset vectorit sekä väliaikainen matriisi. FindContours-funktio on OpenCV:n funktio, joka hakee sille syötetystä kuvasta ääriviivoja, eli tämän sovelluksen tapauksessa valkoisten alueiden ääriviivat kuvasta. Funktio palauttaa ääriviivat tallennettuina vectoriin sekä topologista tietoa löydetyistä ääriviivoista. Vectorien sekä findContours-funktion käyttö on esitetty seuraavalla sivulla olevassa kuvassa (kuva 17).

```
// Funktio etsimään määriteltyjä kohteita kaapatusta kamerakuvasta
void trackFilteredObject(Object theObject,Mat threshold,Mat HSV, Mat &cameraFeed)
{
    // Luodaan Object olion tyyppinen vektori
    vector<Object>objects1;
    // Luodaan väliaikainen matriisi
    Mat temp;
    // Kopioidaan suodatettu kuva temp matriisiin
    threshold.copyTo(temp);
    // Luodaan tarvittavat vektorit findContours funktiota varten
    vector<vector<Point> > contours;
    vector<Vec4i>hierarchy;
    // Etsitään kuvasta objektien ääriviivoja OpenCV:n findContours funktiota käyttäen
    findContours(temp,contours,hierarchy,CV_RETR_CCOMP,CV_CHAIN_APPROX_SIMPLE );
```

KUVA 17. Vectorien sekä findContours-funktion käyttö trackFilteredObject-aliohjelmassa.

Jos findContours-funktio on löytänyt ääriviivoja, niin aliohjelmassa siirrytään seuraavassa kuvassa (kuva 18) esitettyyn if-lauseeseen.

```
double refArea = 0;
bool objectFound = false;
if (hierarchy.size() > 0)
{
    int numObjects = hierarchy.size();
    // Jos löydettyjen objektien lukumäärä on suurempi kuin MAX_NUM_OBJECTS muuttujan arvo
    // niin suodatettu kuva sisältää liikaa kohinaa kohteiden paikantamiseksi
    if(numObjects<MAX_NUM_OBJECTS)
    {
        for (int index = 0; index >= 0; index = hierarchy[index][0])
        {
            // Lasketaan objektien ääriviivojen sisällä olevien pikselien määrä
            // eli toisin sanoen katsotaan minkä kokoinen ääriviivojen sisällä oleva alue on
            Moments moment = moments((Mat)contours[index]);
            double area = moment.m00;
            // Jos alueen koko on pienempi kuin MIN_OBJECT_AREA-muuttujan koko, niin objekti on kohinaa
            // Jos alue on saman kokoinen kuin 3/2 koko kuvan koosta
            // niin silloin se on todennäköisesti huono suodatus
            if(area>MIN_OBJECT_AREA)
            {
                // Luodaan Object tyyppinen olio Object1
                Object Object1;

                // Asetetaan Object1 oliolle koordinaatit, tyyppi ja väri
                Object1.setXPos(moment.m10/area);
                Object1.setYPos(moment.m01/area);
                Object1.setType(theObject.getType());
                Object1.setColour(theObject.getColour());
                // Työnnetään olion parametrit objects1 vektoriin
                objects1.push_back(Object1);
                // Merkitään että kohde on löytynyt
                objectFound = true;
            }
            else
                // Muuten merkitään ettei kohdetta löytynyt
                objectFound = false;
        }
```

KUVA 18. Kohteiden koordinaattien etsiminen moments-funktion avulla.

Aluksi yllä esitetyssä if-lauseessa (kuva 18) katsotaan kuinka paljon löydettyjä objekteja on. Jos objekteja on enemmän kuin ohjelmakoodissa määritelty maksimimäärä, niin

kuvassa on liikaa kohinaa kohteiden paikantamiseksi. Jos objektien lukumäärä on pienempi kuin määritelty maksimimäärä, niin lasketaan ääriviivojen sisällä olevien pikselien lukumäärä käyttäen OpenCV:n moments-funktiota. Jos pikselialueen koko on suurempi kuin ohjelmakoodissa määritelty minimiarvo, niin luodaan Object-tyyppinen olio, jonka parametreiksi asetetaan moments-funktiolla löydetyt koordinaatit ja työnnetään olio vectoriin sekä merkitään kohde löytyneeksi. Lopuksi trackFilteredObject-aliohjelmassa kutsutaan drawObject-aliohjelmaa, jos kuvasta on löydetty kohde. Kyseinen aliohjelmakutsu on esitetty alla olevassa kuvassa (kuva 19).

```
// Jos kohde löydettiin kuvasta niin kutsutaan drawObject funktiota
// merkitsemään tunnistettu kohde kuvaan
if(objectFound==true)
{
    drawObject(objects1,cameraFeed);
}
```

KUVA 19. Kutsutaan drawObject-aliohjelmaa, jos kuvasta on löytynyt kohde.

DrawObject-aliohjelmaa käytetään merkitsemään löydetty kohde alkuperäiseen kuvaan. Kyseisessä aliohjelmassa hyödynnetään OpenCV:n piirtofunktioita piirtämään löydetyn kohteen koordinaatteihin ympyrä, ympyrän yläpuolelle kohteelle määritelty nimi ja alapuolelle koordinaatit. DrawObject-aliohjelman ohjelmakoodi on esitetty alla olevassa kuvassa (kuva 20).

```
// Funktio tunnistettujen kohteiden merkitsemiseksi lopulliseen kuvaan OpenCV:n piirtofunktioiden avulla
void drawObject(vector<Object>theObject,Mat &frame)
{
    for(int i=0;i<theObject.size();i++)
    {
        // Piirretään ympyrä tunnistetun kohteen koordinaatteihin
        circle(frame,cv::Point(theObject.at(i).getXPos(),theObject.at(i).getYPos()),
                10,cv::Scalar(0,0,255));
        // Piirretään kohteen koordinaatit kohteen alapuolelle
        putText(frame,intToString(theObject.at(i).getXPos())+ ", "
                + intToString(theObject.at(i).getYPos()),cv::Point(theObject.at(i).getXPos(),
                theObject.at(i).getYPos()+22),1,1,Scalar(0,0,0));
        // Piirretään kohteen nimi tunnistetun kohteen yläpuolelle
        putText(frame,theObject.at(i).getType(),cv::Point(theObject.at(i).getXPos(),
                theObject.at(i).getYPos()-15),1,1,theObject.at(i).getColour());
    }
}
```

KUVA 20. Merkitään löydetty kohde kuvaan drawObject-aliohjelmalla.

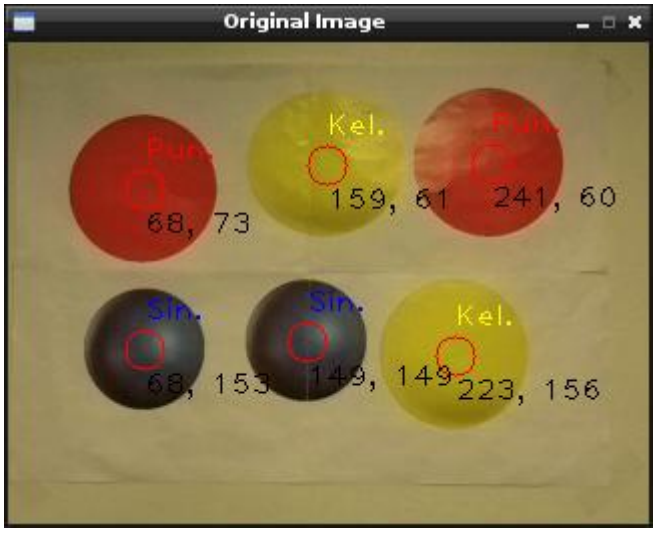
Tässä vaiheessa sovelluksen ohjelmakoodia alkuperäiseen kuvaan on merkitty vain yksi määritetyistä kohteista. Jotta loputkin kohteista tunnistetaan, niin ohjelmakoodissa suoritetaan inRange, morphOps ja trackFilteredObject -funktiot muiden määritetyiden kohteiden suodatusarvoilla. Lopuksi kun kaikki kohteet on ohjelmakoodissa käyty lävitse,

näytetään kuva ikkunassa, johon kaikki tunnistetut kohteet on merkitty. Ohjelmakoodi kaikkien kohteiden ja lopullisen kuvan näyttämiseen ikkunassa on esitetty alla olevassa kuvassa (kuva 21).

```
    // Muulloin tunnistetaan määritetyt kohteet kuvasta
    else
    {

        inRange(HSV,Object1.getHSVmin(),Object1.getHSVmax(),threshold);
        morphOps(threshold);
        trackFilteredObject(Object1,threshold,HSV,cameraFeed);

        inRange(HSV,Object2.getHSVmin(),Object2.getHSVmax(),threshold);
        morphOps(threshold);
        trackFilteredObject(Object2,threshold,HSV,cameraFeed);

        inRange(HSV,Object3.getHSVmin(),Object3.getHSVmax(),threshold);
        morphOps(threshold);
        trackFilteredObject(Object3,threshold,HSV,cameraFeed);

    }
    // Näytetään loppullinen kamerakuva ikkunassa
    // johon on merkitty mahdolliset tunnistetut kohteet
    imshow(windowName,cameraFeed);
}
// Vapautetaan kamerakuvan kaappausobjekti
raspiCamCvReleaseCapture(&capture);
// Poistutaan ohjelmasta
return 0;
```

KUVA 21. Tunnistetaan kaikki määritetyt kohteet kuvasta ja näytetään kuva ikkunassa.

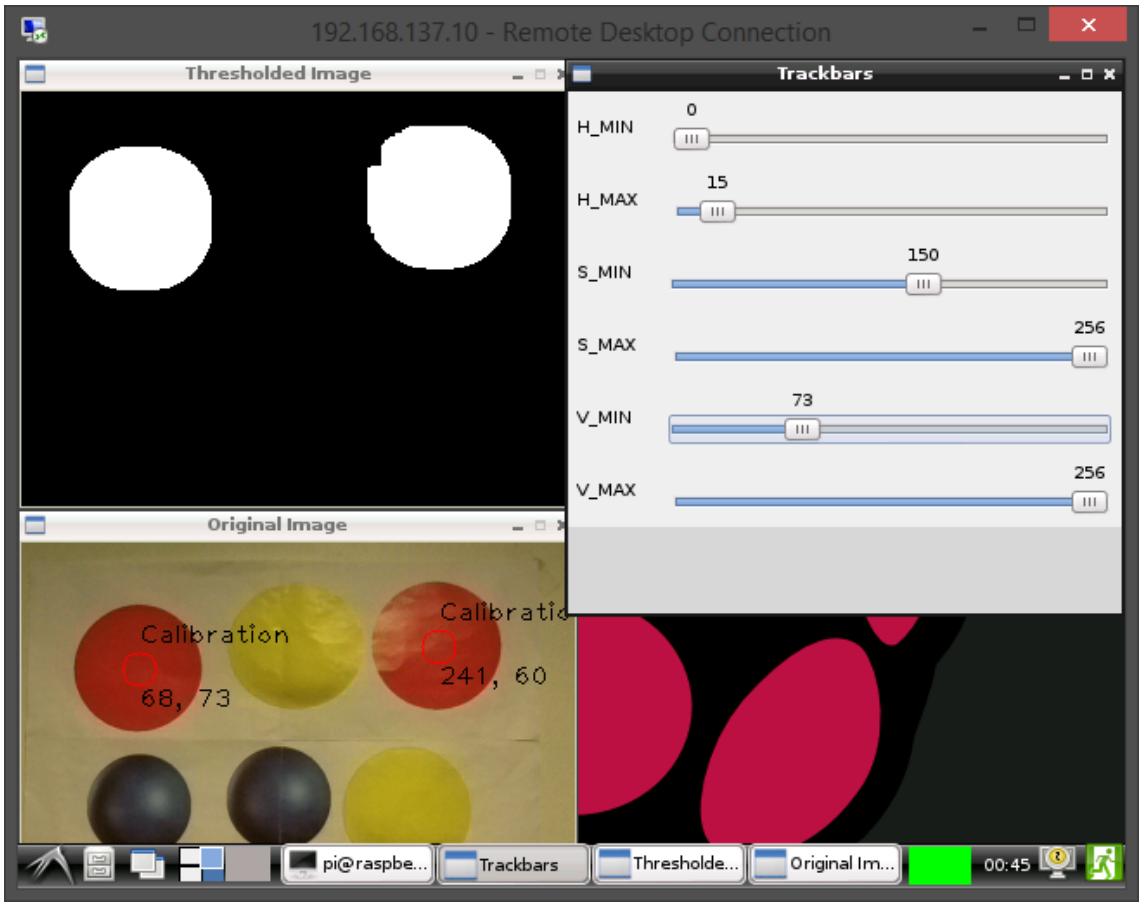
CameraFeed-matriisin sisältämä kuva, kun kaikki kohteet on haettu kuvasta, on esitetty alla olevassa kuvassa (kuva 22).



KUVA 22. Kaikki kohteet haettu kuvasta ja merkitty alkuperäiseen kuvaan.

7.6 Kalibrointitila

Kohteen seurannan ohjelmakoodiin rakennettiin kalibrointitila HSV-suodatusarvojen määrittämiseen kohteille. Jos kalibrointitila on ohjelmakoodissa kytketty päälle, niin luodaan OpenCV:n funktioita käyttäen liukusäätimet, joilla haetaan haluttua kohdetta varten HSV-suodatusarvot. Kalibrointitilassa näytetään myös threshold-matriisin sisältö, jotta parhaiden suodatusarvojen löytäminen olisi helpompaa. Kalibrointitilan ollessa päällä ei suoriteta määritettyjen kohteiden etsimistä kuvasta. Kalibrointitilan käyttöliittymä on esitetty alla olevassa kuvassa (kuva 23).



KUVA 23. Kalibrointitila.

Yllä olevassa kuvassa (kuva 23) on tilanne, jossa on haettu punaisten pallojen suodatusarvoja. Eli toisin sanoen on haettu punaisten pallojen minimi- ja maksimiarvot HSV-kuvan sävylle, kylläisyydelle sekä kirkkaudelle. Kalibrointitilan (kuva 24) sekä liukusäätimien luontiin käytetyn aliohjelman (kuva 25) ohjelmakoodit on esitetty seuraavalla sivulla olevissa kuvissa.

```
// Luodaan muuttuja kalibrointimoodille
bool calibrationMode = true;
// Jos kalibrointitila on päällä niin luodaan liukusäätimet
if(calibrationMode){createTrackbars();}
// Jos kalibrointitilan muuttuja on true niin suoritetaan kalibrointitila
if(calibrationMode==true)
{
    // Suodatetaan HSV kuva määriteltyjen arvojen puitteissa
    // ja tallennetaan suodatettu kuva threshold matriisiin
    inRange(HSV,Scalar(H_MIN,S_MIN,V_MIN),Scalar(H_MAX,S_MAX,V_MAX),threshold);
    // suoritetaan morphologiset operaatiot threshold matriisissa olevalle kuvalle
    // jotta päästään eroon kohinasta
    morphOps(threshold);
    // Näytetään threshold matriisissa sijaitseva kuva ikkunassa
    imshow(windowName2,threshold);
    //imshow(windowName1,HSV);
    // Tunnistetaan kalibrointikohde, joka määritellään kalibrointitilassa
    // liukusäätimien arvojen mukaan
    trackFilteredObject(Object0,threshold,HSV,cameraFeed);
}
```

KUVA 24. Kalibrointitilan ohjelmakoodi.

```
// Funktio liukusäätimien luomista varten
void createTrackbars()
{
    // Luodaan ikkuna liukusäätimille
    namedWindow(trackbarWindowName,0);

    // Luodaan merkkitaulukko johon tallennetaan liukusäätimien nimet
    char TrackbarName[50];
    // Asetetaan liukusäätimille nimet
    sprintf( TrackbarName, "H_MIN", H_MIN);
    sprintf( TrackbarName, "H_MAX", H_MAX);
    sprintf( TrackbarName, "S_MIN", S_MIN);
    sprintf( TrackbarName, "S_MAX", S_MAX);
    sprintf( TrackbarName, "V_MIN", V_MIN);
    sprintf( TrackbarName, "V_MAX", V_MAX);

    // Luodaan liukusäätimet ja asetetaan ne ikkunaan
    // Aliohjelmakutsussa olevat kolme viimeistä parametria vasemmalta oikealle ovat:
    // Liukusäätimellä muutettavan parametrin osoite (esim. H_MIN)
    // Suurin mahdollinen määrä mitä liukusäädin kykenee liikkumaan (esim. H_MAX)
    // Funktio joka kutsutaan aina kun liukusäädintä liikutetaan (on_trackbar)
    createTrackbar( "H_MIN", trackbarWindowName, &H_MIN, H_MAX, on_trackbar );
    createTrackbar( "H_MAX", trackbarWindowName, &H_MAX, H_MAX, on_trackbar );
    createTrackbar( "S_MIN", trackbarWindowName, &S_MIN, S_MAX, on_trackbar );
    createTrackbar( "S_MAX", trackbarWindowName, &S_MAX, S_MAX, on_trackbar );
    createTrackbar( "V_MIN", trackbarWindowName, &V_MIN, V_MAX, on_trackbar );
    createTrackbar( "V_MAX", trackbarWindowName, &V_MAX, V_MAX, on_trackbar );
}
```

KUVA 25. Liukusäätimien luontiin käytetyn createTrackbars-aliohjelman koodi.

7.7 Ohjelmakoodin kääntäminen

Kohteen seurannan ohjelmakoodin kirjoittamiseen käytettiin Visual Studio 2012 -ohjelmankehitysympäristöä suoran etäyhteyden avulla, mutta koodin kääntäminen suoritettiin Raspberry Pi:llä käyttäen g++-kääntäjää. Ohjelmakoodin kääntämisessä muodostui ongelmaksi kaikkien tarvittavien kirjastojen sisällyttäminen käännöskomentoon. Sisällytettäviä kirjastoja oli niin paljon etteivät ne mahtuneet komentoriville.

Tästä syystä työssä käytettiin CMake-käännösjärjestelmää ja make-työkalua. CMake-käännösjärjestelmä kykenee luomaan makefile-tiedoston, joka kertoo make-työkalulle mitä kääntäjää, mitä kirjastoja ja mistä osoitteista otetaan mukaan ohjelmakoodin kääntämiseen. CMake:lle kirjoitettiin CMakeLists-tekstitiedosto, joka sisältää makefile-tiedoston luomista varten kaikki tarvittavat tiedot. Toteutettu CMakeLists-tekstitiedosto on esitetty liitteessä (liite 2).

CMakeLists-tekstitiedosto sijoitettiin käännettävän ohjelman tiedostohakemiston juureen ja komentoriville kirjoitettiin ohjelman juuressa komento "cmake .", jolloin CMake luo makefile-tiedoston samalla näyttäen komentorivillä statistiikkaa makefile-tiedoston luonnista. Kun makefile-tiedosto oli luotu onnistuneesti, niin komentoriville kirjoitettiin komento "make", jolloin make-työkalu kääntää ohjelmakoodin makefile-tiedoston sisältämien käännöskomentojen avulla. Kääntäjä näyttää käännösprosessin aikana statistiikkaa ohjelmakoodin kääntämisestä ja ilmoittaa mahdollisista virheistä ohjelmakoodissa.

8 POHDINTA

Työn tavoitteena oli tutustua Raspberry Pi -tietokoneeseen, siinä käytettävään Linux-käyttöjärjestelmään, kuvantunnistukseen ja toteuttaa jokin toimiva reaaliaikainen kuvantunnistussovellus Raspberry Pi:llä. Nämä opinnäytetyölle asetetut tavoitteet saavutettiin. Työ oli itsessään sopivan haastava, sillä työssä käsitellyistä asioista C++-ohjelmointikieltä lukuun ottamatta ei ollut kokemusta laisinkaan. Lopputuloksena saavutettiin hyvä käsitys Raspberry Pi -tietokoneen kyvyistä, ohjelmoinnista Linux-käyttöjärjestelmässä, kuvantunnistuksesta ja siitä kuinka Raspberry Pi kykenee reaaliaikaiseen kuvantunnistukseen.

Toteutetussa kohteen seuranta -sovelluksessa huomattiin Raspberry Pi:n tehojen rajallisuus ja kuvamuunnosten sekä algoritmien laskennallinen vaativuus. Etenkin kun ohjelmakoodiin määriteltiin kolme erilaista kohdetta, niin sovelluksen kuvanpäivitys alkoi olemaan sellaisissa lukemissa, ettei sellaiselle sovellukselle olisi järkevää käyttöä. Kolmella kohteella ja kameramoduulin minimiresoluutiolla sovelluksen ruudunpäivitysnopeus oli n. 10 ruutua/sekunti. Yhdellä määritetyllä kohteella sovelluksen suorituskyky oli jo sen verran hyvä (n. 20 ruutua/sekunti), että sovellusta olisi mahdollista käyttää vaikka robotissa hakemaan ympäristöstänsä tietyn värisiä objekteja.

Kohteiden paikantaminen värien perusteella on yksinkertainen muttei kovin luotettava tapa etsiä kohteita. Sovelluksessa käytetyllä kuvantunnistusmetodilla tarvitaan hyvä ja tasalaatuinen valaistus, sillä valonlähteen kirkkaus sekä valon sävy vaikuttavat merkittävästi HSV-kuvan arvoihin. Esimerkiksi jos sovellus kalibroitaisiin kellertävälle valaistukselle hämärässä tilassa ja vietäisiin sovellus paikkaan jossa valaistus olisi lähempänä luonnon valoa, niin sovellus toimisi huonosti tai ei laisinkaan. Suodatusarvoja optimoimalla on mahdollista saavuttaa jonkin verran joustavuutta valaistuksen suhteen.

Tämän työn perusteella Raspberry Pi soveltuisi paremmin sovellukseen, jossa ei käytetä värejä vaan mustavalkokuvaa, jolloin käsiteltävänä olisi vain yksi värikanava. Tällöin sovelluksen suorituskyky olisi huomattavasti parempi, kun ei tarvitse käsitellä kaikkia kolmea värikanavaa, eikä sovellus olisi niin herkkä valaistukselle. Tällainen mustavalkosovellus voisi olla vaikka liikkeentunnistus, missä sovelluksessa verrataan kahta kuvaa keskenään ja tunnistetaan kuvasta kohteita liikkeen perusteella.

LÄHTEET

1 Raspberry Pi -säätiön kotisivut. www.raspberrypi.org

2 Element14 kotisivut. www.element14.com

3 Pierre Raufastin Think Rpi niminen Raspberry Pi:tä käsittelevä blogi. http://thinkrpi.wordpress.com/

4 GNU-projektin kotisivut. www.gnu.org

5 Raspbian Linux -jakelupaketin kotisivut. www.raspbian.org

6 SD Association -yhdistyksen kotisivut. www.sdcard.org

7 How To Geek -sivuston tarjoama ohje NOOBS:n asentamiseen. www.howtogeek.com/173101/how-to-enjoy-dead-simple-raspberry-pi-setup-with-noobs/

8 OpenCV-kuvantunnistuskirjaston kotisivut. www.opencv.org

9 Emil Valkovin Pierre Raufastin työn pohjalta tekemä Raspberry Pi:n kameramoduulin OpenCV-kirjasto. https://robidouille.wordpress.com/2013/10/19/raspberry-pi-camera-with-opencv/

10 Kyle Hounslowin OpenCV tutoriaalit. https://www.youtube.com/user/khounslow

11 Meltware's Raspberry Pi Hardware -blogi. http://pihw.wordpress.com/

12 Raspberry Pi:n tiedostojen jakaminen. http://raspberrywebserver.com/serveradmin/share-your-raspberry-pis-files-and-folders-across-a-network.html

13 Raspberry Pi:n userland-kirjaston lähdekoodi. https://github.com/raspberrypi/userland

14 Bradski, G. & Kaehler, A. 2008. Learning OpenCV, 1st Edition. Luettu 19.4.2014. http://books.google.fi/books?id=seAgiOfu2EIC

15 Christopher Wrightin kotisivut. YUV-väriavaruus. http://softpixel.com/~cwright/programming/colorspace/yuv/

LIITTEET

Liite 1. Kohteen seurannan lähdekoodi

multipleObjectTracking.cpp

```
//objectTrackingTutorial.cpp

//Written by  Kyle Hounslow 2013

//Permission is hereby granted, free of charge, to any person obtaining a copy of this software
and associated documentation files (the "Software")
//, to deal in the Software without restriction, including without limitation the rights to use,
copy, modify, merge, publish, distribute, sublicense,
//and/or sell copies of the Software, and to permit persons to whom the Software is furnished to
do so, subject to the following conditions:

//The above copyright notice and this permission notice shall be included in all copies or sub-
stantial portions of the Software.

//THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING
BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
//FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPY-
RIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
//LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN
CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS
//IN THE SOFTWARE.

//Modified by Verneri Keskinen 2014
//Modified to work with Raspberry Pi and Raspberry Pi camera
//Optimized parts of the original code
//Added functions to track multiple objects
//Added finnish comments to explain how the software works

#include <sstream>
#include "/home/pi/git/robidouille/raspicam_cv/RaspiCamCV.h"
#include "Object.h"
#include <vector>

// Alustusarvot HSV suodattimelle
int H_MIN = 0;
int H_MAX = 256;
int S_MIN = 0;
int S_MAX = 256;
int V_MIN = 0;
int V_MAX = 256;
// Oletuskoko kamerakuvan kaappaukselle
const int FRAME_WIDTH = 320;
const int FRAME_HEIGHT = 240;
// Suurin mahdollinen määrä tunnistettavia kohteita yhdestä ruudusta
const int MAX_NUM_OBJECTS=50;
// Kohteelle pienin mahdollinen määrä tunnistettavia pikseleitä
const int MIN_OBJECT_AREA = 30*30;
// Kohteelle suurin mahdollinen määrä tunnistettavia pikseleitä
const int MAX_OBJECT_AREA = FRAME_HEIGHT*FRAME_WIDTH/1.5;
// Nimet eri kuvien ikkunoille
const string windowName = "Original Image";
const string windowName1 = "HSV Image";
const string windowName2 = "Thresholded Image";
const string windowName3 = "After Morphological Operations";
const string trackbarWindowName = "Trackbars";

// Tämä funktio kutsutaan aina kun liukusäätimien sijaintia muutetaan
void on_trackbar( int, void* )
{
   // Funktio on tyhjä mutta silti tarpeellinen liukusäätimien luontifunktion toiminnan kannalta
}

// Funktio jolla muunnetaan int tyyppinen muuttuja string tyyppikseksi muuttujaksi
string intToString(int number)
{
   std::stringstream ss;
   ss << number;
```

```
    return ss.str();
}

// Funktio liukusäätimien luomista varten
void createTrackbars()
{
    // Luodaan ikkuna liukusäätimille
    namedWindow(trackbarWindowName,0);

    // Luodaan merkkitaulukko johon tallennetaan liukusäätimien nimet
    char TrackbarName[50];
    // Asetetaan liukusäätimille nimet
    sprintf( TrackbarName, "H_MIN", H_MIN);
    sprintf( TrackbarName, "H_MAX", H_MAX);
    sprintf( TrackbarName, "S_MIN", S_MIN);
    sprintf( TrackbarName, "S_MAX", S_MAX);
    sprintf( TrackbarName, "V_MIN", V_MIN);
    sprintf( TrackbarName, "V_MAX", V_MAX);

    // Luodaan liukusäätimet ja asetetaan ne ikkunaan
    // Aliohjelmakutsussa olevat kolme viimeistä parametria vasemmalta oikealle ovat:
    // Liukusäätimellä muutettavan parametrin osoite (esim. H_MIN)
    // Suurin mahdollinen määrä mitä liukusäädin kykenee liikkumaan (esim. H_MAX)
    // Funktio joka kutsutaan aina kun liukusäädintä liikutetaan (on_trackbar)
    createTrackbar( "H_MIN", trackbarWindowName, &H_MIN, H_MAX, on_trackbar );
    createTrackbar( "H_MAX", trackbarWindowName, &H_MAX, H_MAX, on_trackbar );
    createTrackbar( "S_MIN", trackbarWindowName, &S_MIN, S_MAX, on_trackbar );
    createTrackbar( "S_MAX", trackbarWindowName, &S_MAX, S_MAX, on_trackbar );
    createTrackbar( "V_MIN", trackbarWindowName, &V_MIN, V_MAX, on_trackbar );
    createTrackbar( "V_MAX", trackbarWindowName, &V_MAX, V_MAX, on_trackbar );
}

// Funktio tunnistettujen kohteiden merkitsemiseksi lopulliseen kuvaan OpenCV:n piirtofunktioi-
    den avulla
void drawObject(vector<Object>theObject,Mat &frame)
{
    for(int i=0;i<theObject.size();i++)
    {
        // Piirretään ympyrä tunnistetun kohteen koordinaatteihin
        circle(frame,cv::Point(theObject.at(i).getXPos(),theObject.at(i).getYPos()),
               10,cv::Scalar(0,0,255));
        // Piirretään kohteen koordinaatit kohteen alapuolelle
        putText(frame,intToString(theObject.at(i).getXPos())+ ", "
               + intToString(theObject.at(i).getYPos()),cv::Point(theObject.at(i).getXPos(),
               theObject.at(i).getYPos()+22),1,1,Scalar(0,0,0));
        // Piirretään kohteen nimi tunnistetun kohteen yläpuolelle
        putText(frame,theObject.at(i).getType(),cv::Point(theObject.at(i).getXPos(),
               theObject.at(i).getYPos()-15),1,1,theObject.at(i).getColour());
    }
}

// Funktio morphologisille operaatioille
void morphOps(Mat &thresh)
{
    // Luodaan kaksi rakentavaa elementtiä
    Mat erodeElement = getStructuringElement( MORPH_RECT,Size(3,3));
    Mat dilateElement = getStructuringElement( MORPH_RECT,Size(8,8));

    // Rappeuttamalla kuvaa siitä karsitaan kaikki objektit
    // jotka ovat kooltaan alle 3x3 pikseliä
    // Näin päästään eroon kohinasta
    erode(thresh,thresh,erodeElement);
    erode(thresh,thresh,erodeElement);

    // Laajennetaan jäljelle jääneet objektit
    // jotta kuvasta haettava kohde on selkeämmin erotettavissa
    dilate(thresh,thresh,dilateElement);
    dilate(thresh,thresh,dilateElement);
}

// Funktio etsimään määriteltyjä kohteita kaapatusta kamerakuvasta
void trackFilteredObject(Object theObject,Mat threshold,Mat HSV, Mat &cameraFeed)
{
    // Luodaan Object olion tyyppinen vektori
    vector<Object>objects1;
    // Luodaan väliaikainen matriisi
    Mat temp;
```

```
    // Kopioidaan suodatettu kuva temp matriisiin
    threshold.copyTo(temp);
    // Luodaan tarvittavat vektorit findContours funktiota varten
    vector<vector<Point> > contours;
    vector<Vec4i>hierarchy;
    // Etsitään kuvasta objektien ääriviivoja OpenCV:n findContours funktiota käyttäen
    findContours(temp,contours,hierarchy,CV_RETR_CCOMP,CV_CHAIN_APPROX_SIMPLE );
    double refArea = 0;
    bool objectFound = false;
    if (hierarchy.size() > 0)
    {
        int numObjects = hierarchy.size();
        // Jos löydettyjen objektien lukumäärä on suurempi kuin MAX_NUM_OBJECTS muuttujan arvo
        // niin suodatettu kuva sisältää liikaa kohinaa kohteiden paikantamiseksi
        if(numObjects<MAX_NUM_OBJECTS)
        {
            for (int index = 0; index >= 0; index = hierarchy[index][0])
            {
                // Lasketaan objektien ääriviivojen sisällä olevien pikselien määrä
                // eli toisin sanoen katsotaan minkä kokoinen ääriviivojen sisällä oleva alue on
                Moments moment = moments((Mat)contours[index]);
                double area = moment.m00;
                // Jos alueen koko on pienempi kuin MIN_OBJECT_AREA-muuttujan koko, niin objekti on
                    kohinaa
                // Jos alue on saman kokoinen kuin 3/2 koko kuvan koosta
                // niin silloin se on todennäköisesti huono suodatus
                if(area>MIN_OBJECT_AREA)
                {
                    // Luodaan Object tyyppinen olio Object1
                    Object Object1;

                    // Asetetaan Object1 oliolle koordinaatit, tyyppi ja väri
                    Object1.setXPos(moment.m10/area);
                    Object1.setYPos(moment.m01/area);
                    Object1.setType(theObject.getType());
                    Object1.setColour(theObject.getColour());
                    // Työnnetään olion parametrit objects1 vektoriin
                    objects1.push_back(Object1);
                    // Merkitään että kohde on löytynyt
                    objectFound = true;
                }
                else
                    // Muuten merkitään ettei kohdetta löytynyt
                    objectFound = false;
            }
            // Jos kohde löydettiin kuvasta niin kutsutaan drawObject funktiota
            // merkitsemään tunnistettu kohde kuvaan
            if(objectFound==true)
            {
                drawObject(objects1,cameraFeed);
            }

        }
        else
            // Muuten kirjoitetaan kuvaan että kuvassa on liikaa kohinaa
            putText(cameraFeed,"TOO MUCH NOISE! ADJUST FILTER",Point(0,50),1,2,Scalar(0,0,255),2);
    }
}

// Pääohjelmasilmukka
int main(int argc, char* argv[])
{
    // Luodaan matriisit kameralta kaapatulle ja suodatetuille kuville
    Mat cameraFeed;
    Mat threshold;
    Mat HSV;
    // Luodaan kamerakuvan kaappausobjekti, jolla hankitaan kameralta videokuvaa
    RaspiCamCvCapture * capture = raspiCamCvCreateCameraCapture(0);
    // Suoritetaan while-silmukkaa niin kauan kunnes näppäimistöltä painetaan jotakin näppäintä
    while(cvWaitKey(10)<0)
    {
        // Kaapataan kameran kuvavirrasta kuva
        IplImage* image = raspiCamCvQueryFrame(capture);
        // Tallennetaan kaapattu kuva cameraFeed matriisiin
        cameraFeed = image;
        // Muunnetaan tallenettu kuva RGB väriavaruudesta HSV väriavaruuteen
        // ja tallenetaan muunnettu kuva HSV matriisiin
```

```
        // HUOM! OpenCV:ssä oletuskuvaformaatti on kyllä RGB, mutta bitit ovat käänteisessä
        // järjestyksessä, jolloin ensimmäinen tavu on sininen, toinen vihreä ja kolmas punainen
        cvtColor(cameraFeed,HSV,COLOR_BGR2HSV);
        // Luodaan tunnistettaville kohteille oliot
        Object Object0("Calibration"),Object1("Kel."),Object2("Sin."),Object3("Pun.");
        // Luodaan muuttuja kalibrointimoodille
        bool calibrationMode = false;
        // Jos kalibrointitila on päällä niin luodaan liukusäätimet
        if(calibrationMode){createTrackbars();}
        // Jos kalibrointitilan muuttuja on true niin suoritetaan kalibrointitila
        if(calibrationMode==true)
        {
           // Suodatetaan HSV kuva määriteltyjen arvojen puitteissa
           // ja tallennetaan suodatettu kuva threshold matriisiin
           inRange(HSV,Scalar(H_MIN,S_MIN,V_MIN),Scalar(H_MAX,S_MAX,V_MAX),threshold);
           // suoritetaan morphologiset operaatiot threshold matriisissa olevalle kuvalle
           // jotta päästään eroon kohinasta
           morphOps(threshold);
           // Näytetään threshold matriisissa sijaitseva kuva ikkunassa
           imshow(windowName2,threshold);
           //imshow(windowName1,HSV);
           // Tunnistetaan kalibrointikohde, joka määritellään kalibrointitilassa
           // liukusäätimien arvojen mukaan
           trackFilteredObject(Object0,threshold,HSV,cameraFeed);
        }
        // Muulloin tunnistetaan määritetyt kohteet kuvasta
        else
        {

           inRange(HSV,Object1.getHSVmin(),Object1.getHSVmax(),threshold);
           morphOps(threshold);
           trackFilteredObject(Object1,threshold,HSV,cameraFeed);

           inRange(HSV,Object2.getHSVmin(),Object2.getHSVmax(),threshold);
           morphOps(threshold);
           trackFilteredObject(Object2,threshold,HSV,cameraFeed);

           inRange(HSV,Object3.getHSVmin(),Object3.getHSVmax(),threshold);
           morphOps(threshold);
           trackFilteredObject(Object3,threshold,HSV,cameraFeed);

        }
        // Näytetään loppullinen kamerakuva ikkunassa
        // johon on merkitty mahdolliset tunnistetut kohteet
        imshow(windowName,cameraFeed);
     }
     // Vapautetaan kamerakuvan kaappausobjekti
     raspiCamCvReleaseCapture(&capture);
     // Poistutaan ohjelmasta
     return 0;
}
```

Object.h

```
#pragma once
#include <string>
#include <cv.h>
#include <highgui.h>
using namespace std;
using namespace cv;

class Object
{
public:
    Object(void);
    ~Object(void);
    Object(string name);
    int getXPos();
    void setXPos(int x);
    int getYPos();
    void setYPos(int y);
    Scalar getHSVmin();
    Scalar getHSVmax();
    void setHSVmin(Scalar min);
    void setHSVmax(Scalar max);
    string getType();
    void setType(string t);
    Scalar getColour();
    void setColour(Scalar c);

private:
    int xPos, yPos;
    string type;
    Scalar HSVmin, HSVmax;
    Scalar Colour;
};
```

Object.cpp

```
#include "Object.h"

Object::Object(void){}

// Määritellään kullekin kohteelle HSV arvot, joiden mukaan kohteet etsitään kuvasta
// Näitä kohteita voidaan helposti luoda lisää tai poistaa oliorakenteen ansiosta
Object::Object(string name)
{
    setType(name);
    if(name=="Calibration")
    {
        setHSVmin(Scalar(0,0,0));
        setHSVmax(Scalar(256,256,256));
        setColour(Scalar(0,0,0));
    }
    if(name=="Kel.")
    {
        setHSVmin(Scalar(16,165,68));
        setHSVmax(Scalar(256,228,256));
        setColour(Scalar(0,255,255));
    }
    else if(name=="Sin.")
    {
        setHSVmin(Scalar(0,0,0));
        setHSVmax(Scalar(256,256,106));
        setColour(Scalar(255,0,0));
    }
    else if(name=="Pun.")
    {
        setHSVmin(Scalar(0,150,73));
        setHSVmax(Scalar(15,256,256));
        setColour(Scalar(0,0,255));
    }
}

Object::~Object(void){}

//Funktiot olion muuttujien arvojen välittämiseen ja asettamiseen
int Object::getXPos(){return Object::xPos;}

void Object::setXPos(int x){Object::xPos=x;}

int Object::getYPos(){return Object::yPos;}

void Object::setYPos(int y){Object::yPos=y;}

Scalar Object::getHSVmin(){return Object::HSVmin;}

Scalar Object::getHSVmax(){return Object::HSVmax;}

void Object::setHSVmin(Scalar min){Object::HSVmin=min;}

void Object::setHSVmax(Scalar max){Object::HSVmax=max;}

string Object::getType(){return type;}

void Object::setType(string t){type=t;}

Scalar Object::getColour(){return Colour;}

void Object::setColour(Scalar c){Colour=c;}
```

Liite 2. CMakeLists.txt

```
cmake_minimum_required(VERSION 2.8)
project(multipleObjectTracking)

find_package( OpenCV REQUIRED )

include_directories(/usr/include/opencv)
include_directories(/home/pi/git/raspberrypi/userland)
include_directories(/home/pi/git/raspberrypi/userland/host_applications/linux/libs/bcm_host/
include)
include_directories(/home/pi/git/raspberrypi/userland/host_applications/linux/apps/raspica)
include_directories(/home/pi/git/raspberrypi/userland/interface/vcos/pthreads)
include_directories(/home/pi/git/raspberrypi/userland/interface/vmcs_host/linux)
include_directories(/home/pi/git/raspberrypi/userland/interface/mmal)
include_directories(/home/pi/git/raspberrypi/userland/build/lib)
include_directories(/home/pi/git/raspberrypi/userland/host_applications/linux/apps/raspicam)
include_directories(/home/pi/git/robidouille/raspicam_cv)
add_executable(multipleObjectTracking multipleObjectTracking.cpp Object.h Object.cpp)
target_link_libraries(multipleObjectTracking
/home/pi/git/raspberrypi/userland/build/lib/libmmal_core.so
/home/pi/git/raspberrypi/userland/build/lib/libmmal.so
/home/pi/git/raspberrypi/userland/build/lib/libmmal_util.so
/home/pi/git/raspberrypi/userland/build/lib/libvcos.so
/home/pi/git/raspberrypi/userland/build/lib/libbcm_host.so
/home/pi/git/robidouille/raspicam_cv/libraspicamcv.a ${OpenCV_LIBS})
```