**IoT Järjestelmän toteutus**

**IoT Face Recognition**

Jaber Askari (M2947)  
Samson Azizyan (M3156)  
Tuukka Bordi (M2296)  
Arttu Häyrynen (M3350)

Kurssityö

Joulukuu 2019

Koulutusalan nimi Tekniikan ala

Tutkinto-ohjelman nimi Insinööri (AMK), Tieto- ja viestintätekniikka

**Sisältö**

[1 Johdanto 1](#_Toc27580253)

[2 Raspberry Pi 3](#_Toc27580254)

[2.1 Main.py 3](#_Toc27580255)

[2.2 Auth.py 4](#_Toc27580256)

[2.3 cl\_facerec.py 5](#_Toc27580257)

[3 ESP32 6](#_Toc27580258)

[4 Grove 7](#_Toc27580259)

[5 Käytetyt kirjastot 8](#_Toc27580260)

[5.1 Face\_recognition -kirjasto 9](#_Toc27580261)

[5.1.1 Komentorivi-ohjelma 9](#_Toc27580262)

[5.1.2 Python moduli 10](#_Toc27580263)

[5.2 Dlib –library 13](#_Toc27580264)

[6 Pohdinta 15](#_Toc27580265)

[Lähteet 16](#_Toc27580266)

# Johdanto

Kurssin alussa saimme tehtävän suunnitella ja rakentaa IoT järjestelmä. Ryhdyimme lähestymään tätä sillä ajatuksella, että haluamme tehdä järjestelmän jonka koemme itsellemme mielekkääksi ja jonka rakentaminen opettaisi meille samalla jotain uutta. Ryhdyimme tutkimaan internetin tee-se-palstoja ja siinä toiveessa, että löytäisimme projektin, joka soveltuisi kurssin vaatimuksiin, olisi mielekäs toteuttaa ja johon olisi olemassa jo valmiiksi hyviä ohjeita ja vinkkejä. Totesimme heti aluksi, että emme halua tehdä sitä perinteistä sääasema-projektia, vaan haluamme tähdätä hieman korkeammalle. Tästä syystä päädyimme tutkimaan projekteja, jotka eivät itsessään ole ns. IoT projekteja, mutta olisivat mukautettavissa sellaisiksi.

Päädyimme suunnittelemaan kasvojentunnistusjärjestelmän, joka rakentuu Raspberry Pi:lle ja sen kameramoduulin. Laajensimme järjestelmän langattomia ominaisuuksia liittämällä siihen Esp32client-moduulin. Raspberry Pi suorittaa varsinaisen kasvojentunnistuksen ja ESP32 mahdollistaa järjestelmän ulkopuolisten laitteiden ohjauksen. Käytimme lähtökohtana Seeed studion älylukko projektia, jossa kasvojentunnistusta käytetään sähkölukon ohjaukseen.

Järjestelmämme tarkoituksena on toimia käyttäjän autentikoinnin välineenä ja tuoda kaksivaiheinen autentikointi eli 2FA reaalimaailmaan. 2FA:n ollessa yleinen menetelmä kirjautua monille internet sivustoille, halusimme luoda järjestelmän, joka mahdollistaa sen vaivattoman käytön myös muissa ohjaus- ja valvontakäytöissä.

Ensimmäiset kunnianhimoiset ajatukset olivat luoda järjestelmä kahvinkeittimen ohjaamiseksi, jossa kasvojentunnistusta käytetään menetelmänä, jolla järjestelmävalvoja voi myöntää tai kieltää käyttäjältä kahvinkeittimen käyttöluvan. Tästä luovuttiin kuitenkin hyvin pian koska totesimme sellaisen järjestelmän olevan tarpeettoman monimutkainen tehtävänantoon nähden. Päädyimme suoraviivaistamaan järjestelmää ja ohjaamaan yksinkertaista LEDiä, kahvinkeittimen sijaan, koska tehtävänantona oli osoittaa järjestelmän toimivuus eikä niinkään luoda lopullinen tuote. Tämä ratkaisi monia ongelmia mitä tulee heikko- ja vahvavirtajärjestelmien sulauttamisen yhteydessä.

Jotta projektimme hallinta ja dokumentointi olisi johdonmukaista ja työnjako olisi paremmin organisoitua, päätimme käyttää Gitlabia. Se helpotti työskentelyä myös kurssin tuntien ja luokan ulkopuolella, tiedostojen jaon helpottuessa. Tämä myös helpotti ja suoraviivaisti myös viikoittaista raportointia opettajalle. Gitlabin käyttö saikin kiitosta heti projektin alusta asti.

Suunnittelemamme järjestelmä on Raspberry Pi:n pohjalle kameramoduulin avulla toteutettu kasvojentunnistusjärjestelmä. Tähän järjestelmään on liitettynä ESP32 -moduuli, jonka avulla saamme lisättyä järjestelmään langattomia toiminnallisuuksia. Projekti voi laajentua ja toiminnallisuudet lisääntyä, jos näyttää siltä, että tähän on aikaa.

# Raspberry Pi

## Main.py

Jaoimme projektin kokonaisuuden osiin helpottaaksemme siihen liittyvän työn ja ajan hallintaa. Ohjelman tekeminen moduuleiksi helpottaa myös ohjelman logiikan ymmärrystä ja muokkausta, joten pyrimme tekemään kaikista projektimme ominaisuuksista moduuleita.

Alla on pääohjelmamme koodi. Pääohjelma sisältää koko projektin toimintalogiikan. Logiikka toimii näissä osissa:

* Kasvon tunnistus (cl\_facerec –pythonmoduuli)
* Autentikointi (auth –pythonmoduuli)
* Ledin etäohjaus (esp32client –pythonmoduuli)

Lisäksi käytämme pythonin json-moduulia tunnettujen käyttäjien tietojen parsimiseksi json-tiedostosta.

Ohjelma toimii seuraavasti näissä käyttötilanteissa:

1. Käyttäjä ottaa kuvan, mutta käyttäjää ei tunnisteta: ohjelma alkaa hälyttää summerilla 2 sekuntia
2. Ohjelma ottaa kuvan, käyttäjä tunnistetaan mutta käyttäjä ei lähetä salasanaa tunnistusohjelman hallinnoimaan sähköpostiin: ajan loputtua (n. 2 min) laite laskee epäonnistuneen yrityksen, ei sytytä lediä (joka olisi onnistuneen autentikoinnin merkki), tallentaa kuvan tuntemattomasta käyttäjästä aikaleimoineen ja palaa kasvojentunnistukseen
   * Sama käyttäjä yrittää tunnistautua 3 kertaa mutta epäonnistuu joka kerralla: laite alkaa hälyttää summerilla 2 sekuntia
   * Laite ottaa kuvan, käyttäjä tunnistetaan mutta käyttäjä lähettää väärän salasanan tunnistusohjelman hallinnoimaan sähköpostiin: ohjelma ei tee mitään ennen kuin autentikoinnin aika kuluu umpeen.
3. Laite ottaa kuvan, käyttäjä tunnistetaan, käyttäjä lähettää oikean salasanan autentikointiajan sisällä: laite lähettää verkon kautta komennon ESP32-laitteelle joka sytyttää ledin.

Autentikointiaika on määritetty auth-moduulissa, kuvan ottaminen ja sen vertaaminen nykyisiin käyttäjiin tapahtuu cl\_facerec –moduulissa niin kuin myös kuvan tallentaminen epäonnistuneiden yritysten jälkeen. Jos käyttäjän todentaminen onnistuu, aktivoidaan LED-valo ESP32-laitteessa tekemämme esp32client-moduulin kautta.

## Auth.py

Osana käyttäjän tunnistamista ohjelmoimme ohjelmallemme moduulin, joka lähettää ja lukee tiedon tunnistuksesta sähköpostilla. Tämä luo erillisen turvallisuuskerroksen järjestelmäämme. Moduuli käyttää tähän sisään koodattua googlen gmail-sähköpostiosoitetta, jonka loimme tätä projektia varten. Tiedostamme käyttäjätunnuksen ja salasanan itse ohjelmaan koodaamiseen liittyvän tietoturvariskin, mutta totesimme että meidän on siedettävä se tässä projektissa. Moduulissa on kolme funktiota: gen\_key, joka luo avaimen, jota käytämme tunnistuksessa, sendEmail, joka lähettää avaimen sisältävän sähköpostin erikseen määritellylle käyttäjälle ja readEmail, joka lukee ja etsii sähköpostilaatikosta viestiä, jossa on avaimen sisältävä otsikko.

Moduuliin toteutettiin rajoitinominaisuus, joka estää ohjelman jumiutumista sähköpostin lukuun readEmail –funktiossa. Ongelmamme oli, että projektin alussa sähköpostin luku oli toimenpide, joka vaati käyttäjän toimia (napin painallusta), jonka jälkeen ohjelma tarkisti löytyikö sähköpostista oikea avain.

## cl\_facerec.py

Varsinainen kasvojentunnistus tapahtuu cl\_facerec python moduulissa. Moduuli sisältää kuvien ottamisen, niiden koodaamisen ja koodausten vertailun. Käytimme tässä lähtökohtana SEEED projektikeskuksen julkaisemaa Face Recognition Smart Lock with LTE Pi HAT -projektia josta kuva alla, ja sen koodia jota muokkasimme omaan tarpeeseemme.



kuva 1 SEEED projektin laitteisto. (Seeed R&D Team. 2019).

Kohtasimme ongelman kasvojen tunnistuksessa suorasta videokuvasta joten päätimme lisätä laitteistoon Grove kytkimen jolla käyttäjä ottaa still-kuvan jota moduuli käyttää itse kasvojen tunnistuksessa. Tämä puolestaan toi tarpeen tuoda käyttäjälle ikkuna nähdä kameran syöttämä video jotta hän voi itse ottaa kuvan kun hänen kasvonsa ovat kuvassa.

Moduuli ottaa kuvan käyttäjän painaessa nappia, aikaleimaa sen ja tallentaa sen käsittelyä varten.

Kuvan saatuaan moduuli muuttaa sen face\_recognition.face\_encodings funktiolla 128 ulotteiseksi koodiksi.

Tämän jälkeen moduuli käyttää face.recognition.comare\_faces funktiota verratakseen tuntemattomia kasvojen koodauksia, tunnettujen kasvojen vastaaviin.

Mikäli tuntemattomat kasvot tunnistetaan, antaa moduuli tuloksena nimen joksi se on kasvot tulkinnut.

Mikäli moduuli ei tunnista kasvoja se antaa tuloksena ” Is the unknown face a new person that we've never seen before?”

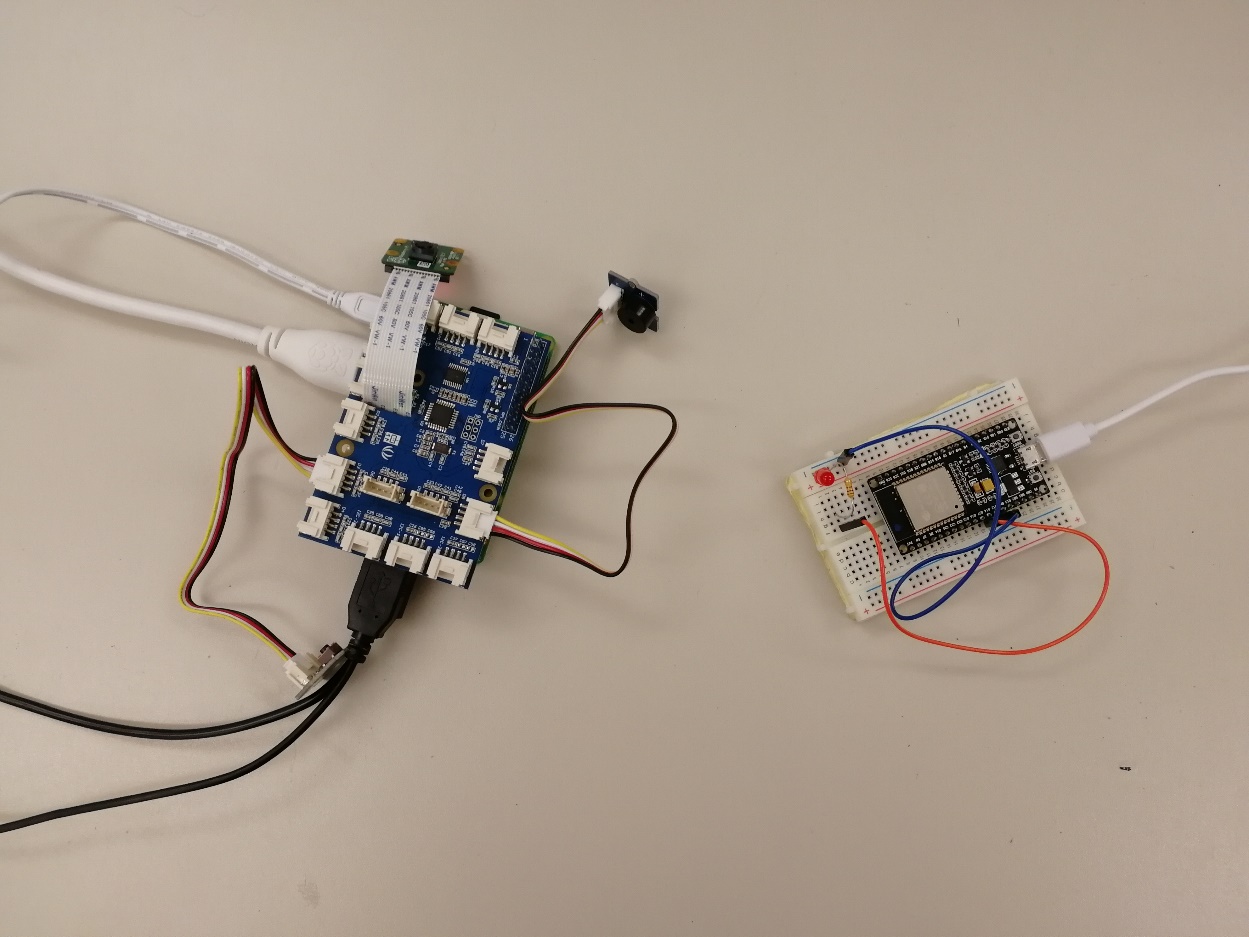
# ESP32

Cl\_facerec- ja Auth-moduuleja käytetään siis käyttäjän tunnistamiseen ja niiden jälkeen täytyy suorittaa itse toiminnon ohjaus. Tähän käytämme ESP32 moduulia jolla muodostamme wifi 802.11 yhteyden itse ESP32:een ja ja annamme sille käskyn suorittaa toiminto. Koska projektimme tarkoituksena on vain havainnollistaa konseptia, käytimme suoritettavana toimintona LEDin sytyttämistä, mutta LEDin vaihtaminen releeseen mahdollistaisi muiden järjestelmien ohjauksen.

Itse ESP32n ohjaaminen tapahtuu socketien avulla. Ensin lähetetään viesti ’1’ joka sytyttää LEDin, tämän jälkeen ohjelma pitää viiden sekunnin tauon jonka ajan LED palaa ja sen jälkeen ESP:lle lähetetään viesti ’0’ joka sammuttaa LEDin.

Koska projekti lähti liikkeelle ajatuksesta käyttää kasvojentunnistusta sähkölukon ohjaukseen, on syytä huomauttaa että Raspberry Pin ja ESP32 välinen wifi-yhteys muodostaa tietoturva-aukon jonka tunnistimme mutta päätimme hyväksyä, johtuen projektin demonstratiivisesta luonteesta. Yhteyden salaaminen ja varmistaminen olisi mahdollista, mutta ei tarkoituksen mukaista, projektin tässä vaiheessa.

# Grove

Päätimme käyttää erillistä kytkintä kasvojentunnistuksen kameran ohjaukseen ja tätä varten hankimme erillisen Groven kytkimen ja totesimme että samalla voisimme liitää järjestelmään erillisen summerin jolla saisimme äänimerkin epäonnistuneesta tunnistuksesta. Valitsimme Groven kytkimen ja summerin koska meidän oli helppo löytää niille tarvittavat kirjastot ja tiesimme niiden käyttöönoton olevan suhteellisen vaivatonta. Päädyimme lopulta kuvan 1 mukaiseen kytkentään sillä totesimme, että voimme käyttää Raspberry Pin hallintaan koulun näyttöä, hiirtä ja näppäimistöä. 

Kuva 2 järjestelmä kytkettynä demonstraatiota varten. (Häyrynen, A. 2019).

# Käytetyt kirjastot

Projekti vaati seuraavat Linux-kirjastot:

* build-essential
* cmake
* gfortran
* git
* wget
* curl
* graphicsmagick
* libgraphicsmagick1-dev
* libatlas-dev
* libavcodec-dev
* libavformat-dev
* libboost-all-dev
* libgtk2.0-dev
* libjpeg-dev
* liblapack-dev
* libswscale-dev
* pkg-config
* python3-dev
* python3-numpy

Python-komponenteista tarvitsemme nämä:

* https://github.com/ageitgey/face\_recognition.git - python-kirjasto, jota käytämme kasvojen tunnistamiseen
* GPIO -kirjasto, jota käytetään GPIO-pinnien ohjaukseen
* picamera -kirjasto kameran ohjaukseen
* PIL / Image -kirjastoa kuvan esittämiseen näytöllä tunnistautumisen yhteydessä

Lisäksi käytämme tunnettuja Linux / Python-kirjastoja ESP32:n ohjaukseen.

Erityistä huomiota vaativia kirjastoja olivat face\_recognition.git ja Dlib koska varsinainen kasvojentunnistus ja siten koko projekti rakentuu niiden varaan, Dlib-kirjaston ollessa se yksittäinen kirjasto minkä varassa kaikki toimii.

## Face\_recognition -kirjasto

Käytimme projektissamme Python 3.6:n face\_recognition -kirjaston versiota 1.2.3 joka on julkaistu elokuussa 2018 ja joka rakentuu Dlib kirjastolle. Face\_recognition -kirjasto on tarkoitettu komentorivi ohjelmaksi ja käytettäväksi Mac ja Linux järjestelmissä, eikä Windowsille ole virallista tukea vaikka käyttäjät ovatkin luoneet ja julkaisseet ohjeita siihen. Sen sijaan kirjaston ylläpitäjä tarjoaa VMwaren virtuaalikoneen kuvan helpottamaan käyttöä. (Geitgey, A. 2017b.)

### Komentorivi-ohjelma

Kirjaston mukana tulee yksinkertainen face\_recognition komentorivi-ohjelma joka vertaa unknown\_pictures -kansiossa olevia kuvia known\_people -kansion kuviin. Tunnetut kuvat tulee nimetä henkilön nimen mukaan. Komentorivi -ohjelma antaa rivin jokaista tunnistamaansa kasvoa kohden. Mikäli kasvoja ei tunnisteta, se nimeää kasvot unknown\_person -nimikkeellä.(Mt.)

* Käsky: $ face\_recognition ./pictures\_of\_people\_i\_know/ ./unknown\_pictures/
* Tunnistetu kuva: /unkown\_pictures/unknown.jpg,Matti Meikäläinen
* Tunnistamaton kuva: /facerecognition\_test/unknown\_pictures/unknown.jpg,unknown\_person

Mikäli yhdet kasvot tunnistetaan useammaksi henkilöksi, saattaa siihen olla syynä että henkilöillä on hyvin samankaltaiset kasvonpiirteet ja tolerance -arvoa tarvitsee laskea jotta tunnistamisen kynnys kasvaisi. Tämä tapahtuu laskemalla --tolerance parametria ja sen vakioasetuksena on 0.6. Mikäli halutaan nähdä kuinka lähellä tunnettu ja tuntemattomat kasvot ovat toisiaan ja täten helpottaa tolerancen säätöä, tulee käyttää --show-distance true komentoa(Mt.)

* $ face\_recognition --show-distance true ./pictures\_of\_people\_i\_know/ ./unknown\_pictures/
* /unknown\_pictures/unknown.jpg,Matt Meikäläinen,0.378542298956785
* /face\_recognition\_test/unknown\_pictures/unknown.jpg,unknown\_person,None

### Python moduli

Käytimme projektissa Face\_recognitionin python moduulia koska halusimme sisällyttää sen osaksi ohjelmaamme emmekä pitää sitä irrallisena osana. Moduli jakautuu osiin seuraavasti:

face\_recognition.api.batch\_face\_locations(images,number\_of\_times\_to\_upsample=1,batch\_size=128)

Funktio antaa joukon 2D laatikoita, jotka rajaavat kuvasta kasvot, tähän käytetään cnn kasvo-tunnistinta. Mikäli käytetään GPUta niin tuloksia saadaan nopeammin koska GPU pystyy käsittelemään kuvat yhtäaikaisesti erissä. Mikäli GPUta ei käytetä niin tätä funktiota ei tarvita. Funktion käyttämät parametrit ovat:

* images, lista kuvista joissa jokainen kuva on numpy joukkona.
* number\_of\_times\_to\_upsample, Kuinka monta näytettä kuvasta kuvasta otetaan kasvojen löytämiseksi. Mitä korkeampi luku on, sitä pienemmät kasvot funktio löytää kuvasta.
* batch\_size, Kuinka monta kuvaa yksittäinen GPUn erä sisältää.

Funktio antaa listan css järjestyksen mukaisen tuplen löydettyjen kasvojen sijainneista. ( Geitgey, A. 2017a)

face\_recognition.api.compare\_faces(known\_face\_encodings,face\_encodings\_to\_check,tolerance=0.6)

Funktio vertaa tunnettujen kasvojen koodausluetteloa ehdokkaiden luetteloon, etsien vastaavuuksia.Funktion käyttämät parametrit ovat:

* known\_face\_encodings, lista tunnetujen kasvojen koodauksista
* face\_encoding\_to\_check, lista kasvokoodauksia, joita verrataan tunnetujen kasvojen koodausten listaan.
* tolerance, Kuinka lähellä toisiaan kasvojen täytyy olla, jotta ne tulkitaan tunnistetuiksi.

Funktio antaa True/False sen mukaan mikä known\_face\_encodings arvo vastaa tarkistattavaa.(Mt.)

face\_recognition.api.face\_distance(face\_encodings,face\_to\_compare)

Funktio vertaa annettua luetteloa kasvojen koodauksista, tunnettujen kasvojen koodauksiin, antaen euklidisen arvon etäisyydestä verrattavista kasvoista. Etäisyys kertoo kuinka samankaltaisia kasvot ovat.Funktion käyttämät parametrit ovat:

* faces. lista kasvojen koodauksista, joita verrataan
* faces\_to\_compare, lista kasvojen koodauksista, joihin verrataan

Funktio antaa numpy n-joukon etäisyyksineen, ”faces” joukon mukaisessa järjestyksessä. (Mt.)

face\_recognition.api.face\_encodings(face\_image,known\_face\_locations=None,num\_jitters=1)

Funktio antaa 128-ulotteisen koodauksen jokaisesta kasvosta mitä siihen syötetty kuva sisältää. Funktion käyttämät parametrit ovat:

* face\_image, kuva joka sisältää yhden tai useammat kasvot
* known\_face\_locations, valinnainen, kasvot rajaava laatikko jokaiselle havaitulle kasvolle.
* num\_jitters, kuinka monta kertaa kasvoista otetaan näytteet kun niille lasketaan koodausta. Mitä enemmän näytteitä otetaan, sitä tarkempi koodaus saadaan mutta se on myös hitaampaa. Arvo 100 on 100 kertaa hitaampi kuin 1.

Funktio antaa listan 128-ulotteisia koodauksia kasvoista. Yhden jokaisesta kasvosta, jota kuvasta havaitaan. (Mt.)

face\_recognition.api.face\_landmarks(face\_image,face\_locations=None,model=’large’)

Funktio palauttaa sanakirjan annetun kuvan jokaisen kasvon kasvojen piirteiden sijainneista. Yhden sanakirjan jokaiselle kasvolle.  
Funktion käyttämät parametrit ovat:

* face\_image, kuva josta kasvoja etsitään
* face\_locations, valinnainen tapa antaa lista kasvojen sijainneista niiden löytämistä varten
* model, valinnainen, käytetäänkö kasvojen piirteiden tunnistamiseen ”large” eli isoa (joka on vakioasetus) mallia vai ”small” eli pientä mallia, joka antaa vain 5 pistetta mutta on paljon nopeampi.

Funktio antaa sanakirjan kasvojen piirteiden kuten silmät, nenä, suu ja korvat sijainneista.(Mt.)

face\_recognition.api.face\_locations(img,numner\_of\_times\_to\_upsample=1,model=’hog’)

Palauttaa jouko laatikoita jotka rajaavat sisälleen kuvasta havaitut ihmisten kasvot.  
Funktio käyttämät parametrit ovat:

* img, kuva numpy joukkona
* number\_of\_times\_to\_upsample, Kuinka monta näytettä kuvasta kuvasta otetaan kasvojen löytämiseksi. Mitä korkeampi luku on, sitä pienemmät kasvot funktio löytää kuvasta.
* model, Käyttetävä kasvojentunnistusmalli. ”hog” on epätarkin mutta nopein käytettäessä CPUta ja sitä käytetään vakiona. ”cnn” on tarkempi deep-learning malli, joka käyttää GPU/CUDA kiihdytystä, mikäli mahdollista.

Funktio antaa css järjestyksen mukaisen tuplen havaittujen kasvojen sijainneista.(Mt.)

face\_recognition.api.load\_image\_file(file, mode=’RGB’)

Funktio muuttaa annetun kuvatiedoston numpy joukoksi.  
Funktion käyttämät parametrit ovat:

* file, Kuvatiedoston nimi tai tiedosto-objekti, muunnosta varten.
* mode, Formaatti johon tiedosto muunnetaan. Funktio tukee vain ”RGB” (8-bittinen, 3-kanavainen RGB) ja ”L” (musta-valkoinen) formaatteja.

Funktio palauttaa numpy joukon kuvan sisällöstä. (Mt.)

## Dlib –library

Dlib on vuonna 2002 jukaistu ja C++ ohjelmointikielellä kirjoitettu, avoimen lähdekoodin ohjelmistokirjasto, joka sisältää laajan skaalan eri koneoppimisen itsenäisiä algoritmeja, jotka ovat suunniteltu nopeiksi ja hyvin modulaarisiksi. Ne on yksinkertaisia ja nopeita käyttää modederneilla C++ API:lla. Dlibin sisältämillä komponenteilla käsittellään ainakin graafisia käyttöliittymiä, data rakenteita, lineaarista algebraa, koneoppimista, kuvan prosessointia, tiedon louhintaa ja Bayesin verkkoja. (Dlib C++ Library. 2019a.)

Dlibin ydin ajatuksena on olla helppokäyttöinen, siksi sen koodi on suunniteltu mahdollisimman kevyeksi eikä sen asennus vaadi käyttäjältä mitään erityisiä konfiguraatiotoimia tai muiden lisäosien asennusta. Tämän helpottamiseksi kaikki järjestelmä kohtainen koodi on rajattu API wrapperiin. Kaikki muu on joko kirjoitettu puhtaalla ISO standardin mukaisella C++:lla tai kirjoitettu wrapperin päälle-Tällä hetkellä Dlib toimii OS X, MS Windows, Linux, Solaris, BSD ja HP-UX käyttöjärjestelmillä. (Mt.)

Kun Dlibiä käytetään kuvan prosessointiin tai manipulointiin, on syytä muistaa, että Dlib ei varsinaisesti käytä kuvaobjekteja vaan käsittelee kaiken array2d objekteina eli kaksi ulotteisina, käyttäjän määritteleminä pikseli joukkoina. (Dlib C++ Library. 2019b.)

Pixel tyyppeistä

Useimmat kuvankäsittelyrutiinit Dlibissä hyväksyvät kaikkia piksel tyyppejä sisältäviä kuvat. Tämän mahdollistaa pixel\_traits luokka joka on määritelty jokaiselle mahdolliselle pixel tyypille. Nämä traits luokat mahdollistavat kuvankäsittely rutiinien päätellä miten jokaista pixel tyyppiä käsitellään, ja siksi vain luokiteltuja pixel tyyppejä voidaan käsitellä. (Mt.)

* RGB, Dlibissä on kahdenlaisia RGB pixel tyyppejä, rgb\_pixel ja bgr\_pixel. Molemmat määrittelevät 24-bittisen RGB pixel tyypin. Bgr\_pixel on identtinen rgb\_pixelin kanssa muuten, mutta se laittaa värikanavat muistiin BGR järjestyksessä RGB järjestyksen sijaan. Tämä mahdollistaa muiden BGR järjestystä käyttävien työkalujen, kuten OpenCV, käytön Dlibin kanssa.
* RGB Alpha, eli rgb\_alpha\_pixel on 8 bittinen alpha komposiittion kanava joka käyttää 8 bittin kanavia RGB järjestyksessä.
* HSI, eli his\_pixel on 24 bittinen pikseli, joka kuvaa pistettä Hue Saturation Intensity (HSI) eli sävyn saturaation voimakkuutta, nk. väriavaruudessa.
* LAB, eli lab\_pixel on 24 bittine pikseli, joka kuvaa pistettä CIELab väriavaruudessa.
* Grayscale eli harmaan skaala, mitä tahansa sisäänrakennettua skaalatyyppiä voidaan käyttää harmaaskaalan tyyppinä. Tällaisia on esimerkiksi douple, int ja unsigned char. (Mt.)

# Pohdinta

Projektina kasvojentunnistus ja sen käyttö laitteen ohjaukseen internetin yli on mielenkiintoinen ja ajankohtainen. Saimme myös hyvää harjoitusta todellisen projektityöskentelyyn ja sen haasteisiin. Hyvänä esimerkkinä haasteista on toiminut työn jaon järkeistäminen, kun varsinainen projektityö tehdään hajautetusti eri paikoissa, eri aikoina. Tämä loikin meille haasteita, joita emme osanneet ennakoida työhön ryhtyessämme, mutta jotka saimme ratkaistua projektin edetessä. Käytimme jokainen tähän projektiin noin 150 työtuntia tahoillamme.

Lähteet

Dlib C++ Library. 2019a. Kirjaston kuvaus verkkosivuilla. Viitattu 12.12.2019. <http://dlib.net/>

Dlib C++ Library. 2019b. Kuvan käsittelyn kuvaus verkkosivuilla. Viitattu 13.12.2019. <http://dlib.net/imaging.html>

Geitgey, A. 2017a. face\_recognition package. Viitattu 12.12.2019. [https://face-recognition.readthedocs.io/en/latest/face\_recognition.html](https://face-recognition.readthedocs.io/en/latest/face_recognition.html%20)

Geitgey, A. 2017b. usage. Viitattu 13.12.2019. <https://face-recognition.readthedocs.io/en/latest/readme.html#usage>

Häyrynen, A. 2019. Projektikuva. Viitattu 18.12.2019

Seeed R&D Team. 2019. Hardware Connection. Viitattu 13.12.2019. c<https://project.seeedstudio.com/SeeedStudio/face-recognization-smart-lock-with-lte-pi-hat-abcec9>