Markus Vesa

Autonominen RC-auto

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Ajoneuvotekniikka

6.1.2019

Metropolia

Tekijä Otsikko Sivumäärä Aika	Markus Vesa Autonominen RC auto 43 sivua + 8 liitettä 6.1.2019
Tutkinto	Insinööri (AMK)
Tutkinto-ohjelma	Ajoneuvotekniikka
Ammatillinen pääaine	Autosähkötekniikka
Ohjaajat	Lehtori Vesa Linja-aho

Opinnäytetyön aiheena oli muuttaa normaalisti radio-ohjattava auto itsestään ajavaksi laitteeksi ja samalla tutkia operaation vaatimuksia. Projektin autoon asennettiin pieni tietokone ja autoa muokattiin niin, että se voi ottaa tietokoneelta käskyjä vastaan.

Opinnäytetyön aikana tutkittiin neuroneiden, neuroverkon, konenäön ja verkkotekniikan toimintaa ja tuotiin kaikki yhtenäiseksi kokonaisuudeksi auton autonomian luomiseksi.

Aluksi projektissa luotiin mahdollisuus ohjata autoa tietokoneen välityksellä. Tämän jälkeen autoon asennettiin kamera, jolla mahdollistettiin kuvadatan keräys ajon aikana. Auton kameran suoratoisto tallennettiin auton komentojen kanssa. Komennot ja kuvamateriaali paritettiin keskenään ja siitä luotiin yhtenäinen taulukkoformaattiin perustuva tiedosto, joka oli valmis neuroverkkoon syötettäväksi.

Neuroverkko koulutettiin luodulla datalla, minkä jälkeen auton ja tietokoneen välille luotiin uusi verkkokommunikointisovellus. Tietokoneella otettiin autosta vastaan kuvainformaatiota, joka syötettiin koulutettuun neuroverkkomalliin. Tietokone loi arvauksen saamastaan informaatiosta ja muokkasi sen verkon kautta lähettäväksi dataksi autolle. Auto ottaa komennon vastaan ja suorittaa koneen luomia arvauksia niin kauan kuin malli on käynnissä.

Työn tuloksena syntyi itsestään ajava radio-ohjattava auto. Auto tunnistaa ympäristöstä ajoradan ja kykenee tekemään päätöksiä saamansa datan perusteella.

Avainsanat	Autonominen auto, itsestään ajava ajoneuvo, koneoppiminen

Metropolia

Author Title Number of Pages Date	Markus Vesa Autonomous RC Car 43 pages + 8 appendices 6 January 2019
Degree	Bachelor of Engineering
Degree Programme	Automotive Engineering
Professional Major	Automotive Electronics Engineering
Instructors	Vesa Linja-aho, Senior Lecturer

The goal of the thesis was to create autonomous car in a small form factor and to explore what this operation demands. A small computer was installed in to the car in order to make it able to receive commands via desktop computer.

During the thesis neurons, neural network, machine vision and internet communication was research for creating autonomous vehicle.

A web communication application was created for the car to receive direct commands from the desktop computer. The car was fitted with a camera for data collection. Video stream from the car was saved with commands from the desktop computer. Video stream was merged with driving commands to create array file. This array file was ready to be fed to neural network.

A convolutional neural network (CNN) model was created and trained with collected data. New communication program was created so the CNN and the car were able to communicate. Desktop computer receives video information from the car's camera and feeds it to the loaded CNN model. CNN then creates a prediction based on that information, reformats it and sends the order for the car to execute. The car continues to receive predictions until either communication program is killed.

The product of the thesis was autonomous vehicle in RC car form factor. The vehicle is capable recognizing road and making decisions based on the data it receives.

Keywords	Autonomous car, self driving, machine learning

Metropolia

Sisällys

Lyhenteet

1 Johdanto 1

2 Koneoppimisen perusta 3

2.1 Neuroverkon lyhyt historia 3
2.2 Neuroni ja neuroverkko 4

3 Laitteisto 6

3.1 Auton ominaisuuksien esittely 6
3.2 Raspberry Pi ja lisälaitteet 7
3.3 GPIO 9
3.4 Tietokone 10

4 Auton modifikaatio 11

5 Käytössä olevat ohjelmistot 13

5.1 NumPy 13
5.2 OpenCV 13
5.3 Tornado ja socket 14
5.4 RPi Cam Web Interface 14
5.5 Tensorflow 15
5.6 TFlearn 17

6 Datan keräys 18

6.1 Manuaaliajo 18
6.2 Session tallennus 20
6.3 Koulutusdatan luonti 21

7 Neuroverkko 22

7.1 Konvoluutio (convolutional layer) 22
7.2 Yhdistyskerros (max pooling layer) 23
7.3 Kerätyn datan tuominen verkolle 25
7.4 Neuroverkon malli 26

7.5 Koulutuksen tulos 30

8 Itsestään ajava auto 33

8.1 Arvauksista komennoiksi 33
8.2 Verkkokommunikointi 34
8.3 Auton käyttö 35
8.4 Ongelmia 36

9 Yhteenveto 36

Lähteet

Liitteet
Liite 1. Piirisarja
Liite 2. manuaaliajo.py
Liite 3. tallenna_striimi.py
Liite 4. save_all_runs_as_numpy_files.py
Liite 5. tf_CNN.py
Liite 6. Host_drive.py
Liite 7. Prediction_client.py
Liite 8. Dataprep.py

Erityissanasto

Neuroverkko	Verkko neuroneja, joita käytetään koneoppimiseen.
Konvoluutio	Neuroverkoissa käytettävä tapa skannata kuva läpi etsien suurinta arvoa pikseleistä.
Max pool	Konvoluution yhteydessä käytettävä skannaus, joka etsii suurimmat erityispiirteet kuvasta.
Epoch	Koulutusvektorien painojen päivityksien kerrat.
Batch	Neuroverkkomalliin kerralla syötettävän datan määrä.
Ennusteet	Predictor. Koulukseen käytettävä data.
Luokka/nimikkeet	Label. Koulutusdatalle annettu luokka tai nimike.
RELU	Rectified Linear Unit. Aktivointifunktio neuroneille.
Softmax	Aktivointifunktio viimeiselle kerrokselle.
Tensorflow	Koneoppimiseen käytettävä kirjasto.
Tflearn	Tensorflow'lle luotu korkeampi rajapinta, jonka tarkoituksena on nopeuttaa ja yksinkertaistaa neuroverkon luontia.
Numpy	Taulukoiden käsittelyyn luotu kirjasto Pythonille.
Python	Ohjelmointikieli.
Tornado	Web-sovelluksiin luotu kirjasto.

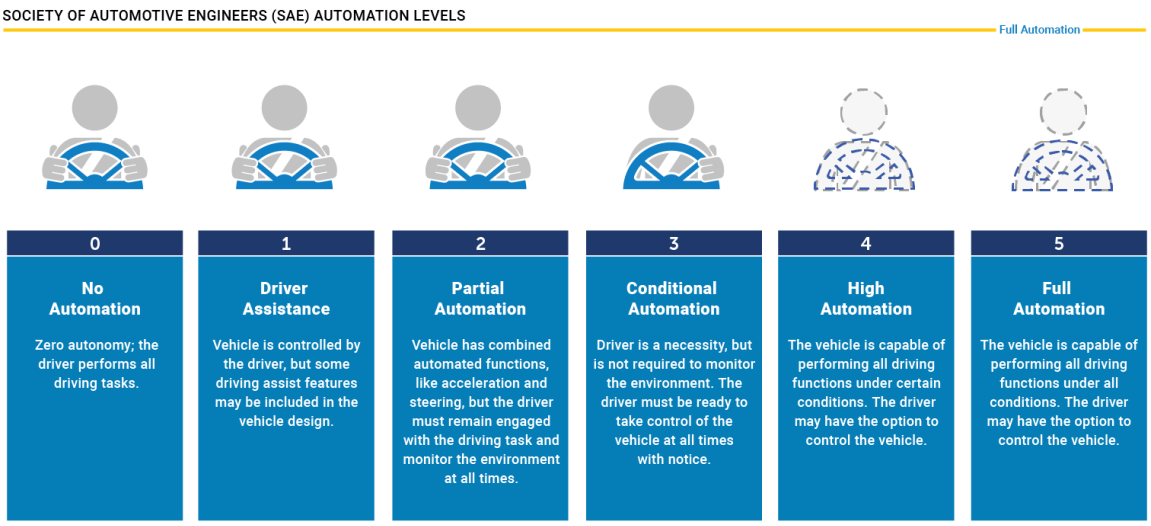
OpenCV	Konenäköön luotu kirjasto.
Paino	Weight. Neuroneille luotava paino, joka vaikuttaa neuroneiden aktivoitumiseen.
Taipumus	Bias. Painoon lisättävä arvo. Nopeuttaa oppimista.
ESC	Electronic Speed Controller. Autossa oleva piiri, joka säätää auton toimintoja ja nopeutta.
GPIO	General Purpose Input Output. Raspberryn ohjelmoitavat pinnit.
Ylisovitus	Tilanne, jossa neuroverkko oppii käytössä olevan datan liian hyvin ja antaa tulokseksi liian suuren tarkkuuden.
Alisovitus	Tilanne, jossa neuroverkko ei kykene luomaan datasta mallia.
CUDA	Nvidian rinnakkainen laskenta-alusta ja ohjelmointimalli, joka hyödyntää näytönohjainta. Käytetään koneoppimisen nopeuttamiseen

1 Johdanto

Tässä opinnäytetyössä luodaan itse itsestään ajava ajoneuvo ja tutkitaan, mitä sen tekeminen vaatii. Täysikokoisen ajoneuvon sijasta käytössä on radio-ohjattava auto, jonka tarkoitus on ajaa sisätilassa luotua rataa ympäri täysin itsenäisesti. Autolle luodaan oma neuroverkkomalli ja se koulutetaan itse kerätyllä datalla. Autolla on käytössä oma tietokone, joka ohjaa auton toimintaa ja välittää dataa. Neuroverkon koulutuksessa käytetään pöytätietokonetta.

Itsestään ajavia ajoneuvoja ja muuta automaatiota voi lähestyä utooppisesta tai dystooppisesta näkökulmasta. Utopistisessa maailmassa autonomiset ajoneuvot vähentävät liikenteessä tapahtuvia tapaturmia ja kuolemia, tarjoavat jokaiselle edullisen tavan liikkua kaupungeissa ja säästävät aikaa eliminoimalla ruuhkan. Dystooppisesta näkökulmasta katsottuna autonomia aiheuttaisi tuhansien henkilöiden työttömyyden. Vaikka onnettomuudet olisivatkin harvinaisempia, ne voivat johtua asioista, jotka ovat ihmiselle vaikea selvittää tai joihin vaikuttaminen on hankalaa. Koneäly voi myös kehittää taipumuksia, jotka eivät vastaa meidän sosiaalisia tai eettisiä arvojamme. Tietoturvan taso nousee myös kysymykseksi autonomiassa. Todellisuus tulee varmaan olemaan utopistisen ja dystooppisen näkemyksen välimaastossa ja täysin itsestään ajavien autojen yleistyminen tulee kestämään vuosia, ellei vuosikymmeniä. (1, s. 2.)

Automaation tasot voidaan jakaa kuuteen tasoon. Tason 0 autosta ei löydy mitään ajoavusteita, ja tasossa 5 auto toimii täysin autonomisesti. Suurin osa tänään kaduilla kulkevista autoista voidaan lokeroida tasoon 1. Monista uusista keskihintaisista autoista löytyy kaistavahteja ja adaptiivisia vakionopeudensäätimiä, jotka tekevät niistä tason 2 autoja. Tasoon 3 yltäviä autoja ovat esimerkiksi Tesla Model S ja X. Tasossa 3 auto pystyy ottamaan ohjauksen itselleen, mutta siirtää hallinnan takaisin kuljettajalle kohdatessaan tilanteen, jota se ei pysty selvittämään itsekseen. Tason 4 auto vaatii vielä ihmisen ohjausta vaativissa tilanteissa, mutta auto kykenee ajamaan huomattavasti enemmän itsekseen verrattuna tasoon 3. Kuvassa 1 kuvataan, mitä eri automaation tasot pitävät sisällään. (1, s. 6; 30.)



Kuva 1. SAE:n automaation tasot (2).

Automaation tason noustessa vaaditaan entistä enemmän sensoreita, jotka keräävät dataa, lisää laskentatehoa sekä parempia ohjelmistoja. Itsestään ajavissa autoissa ohjelmisto tehdään kouluttamalla neuroverkko kerätyllä datalla.

Neuroverkko ei itsessään ole uusi keksintö. Ensimmäinen neuroverkkomalli luotiin virtapiirejä käyttämällä jo 1943 Warren McCullochin ja Walter Pittsin toimesta, mutta neuroverkot ovat nousseet suosiossa vasta tällä vuosikymmenellä. Suurin syy suosion nousuun on laskentatehon nousu ja saatavilla oleva datan määrä. Myös sensorit ovat kehittyneet tarkemmiksi, ja tallennustilan kasvu ovat mahdollistaneet suuremman datan määrän keräyksen. (3; 4) Waymon insinööri Sacha Arnoud kuvaili tilannetta, että kun olet 90-prosenttisesti valmis, jäljellä oleva työ vaatii 10 kertaa suuremman työmäärän. (5)

2 Koneoppimisen perusta

Meille ihmisille tulee yleensä itsestään selvyytenä, miten helppoa on erottaa koira kissasta, mutta meiltä unohtuu, että tämä on miljoonien vuosien evoluution tulos. Kun koneelle aletaan opettamaan asioiden ja esineiden tunnistamista, on lähdettävä nollasta liikenteeseen ja mietittävä, miten voimme käyttää hyväksi omaa tietämystämme kuvien hahmottamisesta.

Itsestään ajava auto voidaan toteuttaa neuroverkolla. Ennen neuroverkon luontia pitää kerätä paljon dataa. Mitä enemmän dataa on saatavilla, sen parempi ja tarkempi neuroverkosta tulee. Projektin auto luottaa täysin kamerasta saatuun dataan arvioidessaan seuraavia liikkeitään. Neuroverkko opetetaan manuaalisesta ajosessiosta kerätyllä datalla, ja neuroverkko pyrkii matkimaan tätä ajoa. Neuroverkolle syötetään sessiosta saatuja kuvia ja luokituksia, joiden perusteella se pystyy hahmottamaan datassa olevia yhtenäisyyksiä ja antaa arvauksia. Verkolle annetaan painot (weight) ja taipumukset (bias), joiden perusteella vääriä arvauksia karsitaan pois ja oikeille vastauksille annetaan enemmän painoarvoa. Neuroverkolle annettava data pitää jakaa koulutusdataksi ja testidataksi. Koulutusdataa tulee olla suhteessa testidataan huomattavasti enemmän. Data tulisi mielellään jakaa niin, että koulutukseen käytettävä data on noin 80 prosenttia koko datan määrästä ja testaukseen käytetään loput 20 prosenttia. (6, s. 39.)

2.1 Neuroverkon lyhyt historia

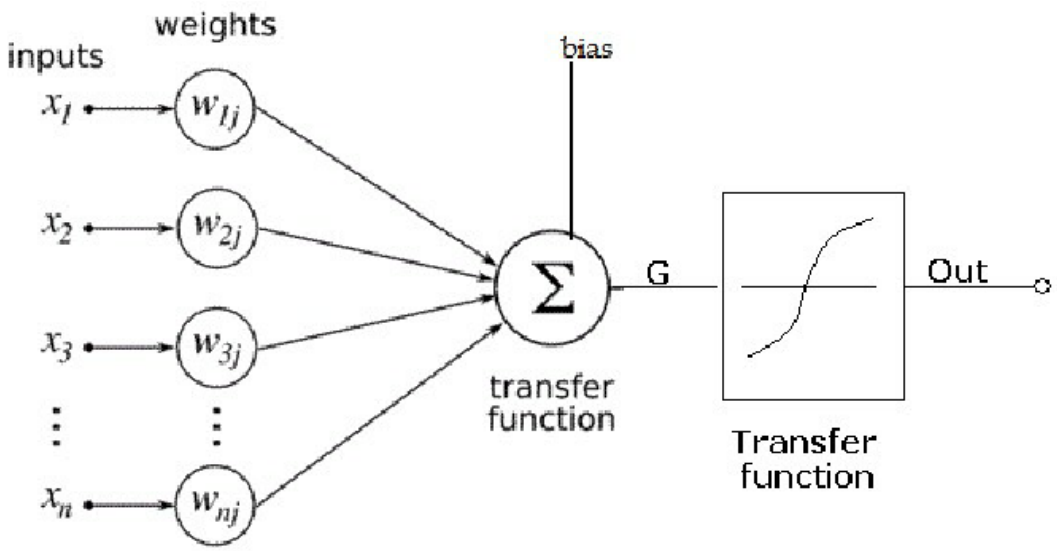
1943 neuropsykologi Warren McCulloch ja matemaatikko Walter Pitts tekivät tutkimuksen, miten aivojen neuronit mahdollisesti toimivat ja mallinsivat neuronien toiminnan yksinkertaisella virtapiirillä. 1949 Donald Hebb jalosti neuroverkkojen ideaa eteenpäin kirjoittamalla ”The Organization of Behaviour” -nimisen kirjan, jossa hän ehdotti, että neuronien välinen yhteys vahvistuu, mitä useammin neuroneja käytetään. 1950-luvulla IBM alkoi tekemään tutkimuksia neuroverkon simulaatiosta.

1959 Marcian Hoff ja Bernard Widrow loivat ADALINE- ja MADALINE-neuroverkot, joista jälkimmäinen on vieläkin käytössä puhelinlinjojen kaiun suodatuksessa. Aikaisemmat

onnistumiset loivat innostuksen neuroverkkoja kohtaan, joka lopulta viileni, koska epärealistia odotuksia ei pystytty täyttämään. 1982 luvulla kiinnostus neuroverkkoihin tuli takaisin, kun John Hopfield esitti tutkimuksen, jossa ihmisen aivojen mallinnuksen sijaan neuroverkkoja pyrittiin käyttämään hyödyllisten laitteiden luontiin. (3; 4)

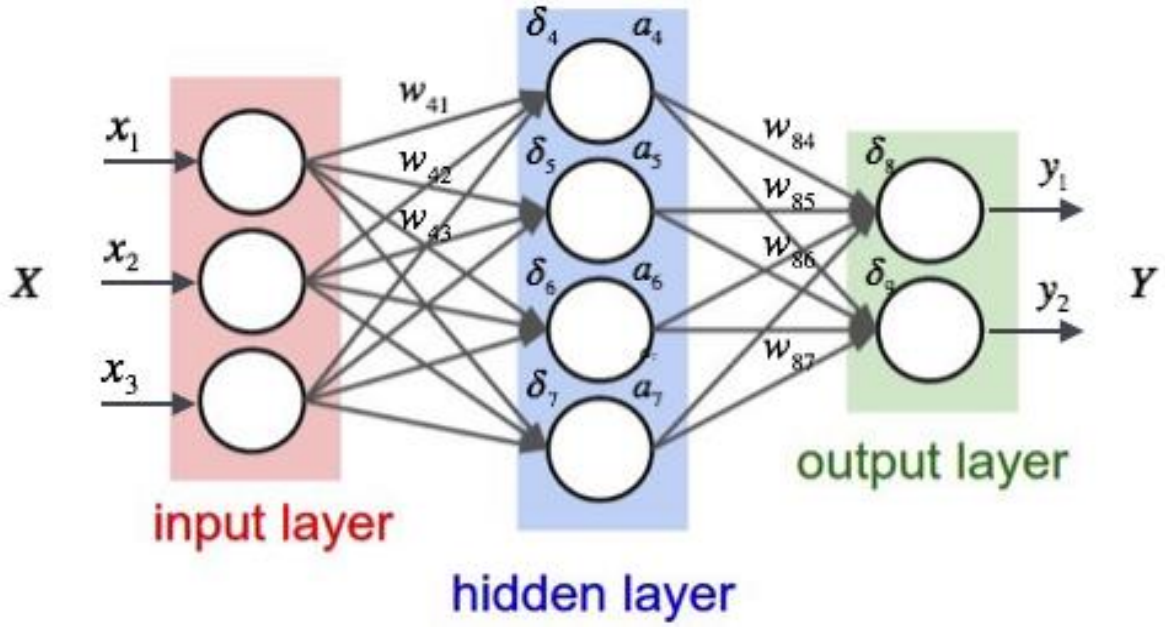
2.2 Neuroni ja neuroverkko

Biologisessa neuronissa dendriitti ottaa vastaan syötteen. Syöte viedään sooman läpi, josta signaali kulkee aksonille, jonka päässä on terminaalit. Aksonin terminaalit ovat kytkettynä toisen neuronin dendriitteihin. Vastaavalla tavalla toimii myös keinotekoinen neuroni. Neuronille voidaan antaa x määrä syötteitä. Jokaiselle syötteelle on oma paino, ja syötteet summataan keskenään. Syötteiden summalle lisätään myös taipumus. Taipumus ei ole neuroniverkossa pakollinen ominaisuus, mutta se nopeuttaa neuroniverkon opetusta. Syötteiden ja taipumuksen summa viedään aktivointifunktiolle. Aktivointifunktiolla on oma kynnys, jonka pitää ylittyä, että neuroni aktivoituu. Mikäli aktivointikynnys ylitetään, antaa neuroni arvon, joka toimii syötteenä seuraaville neuroneille. Jos aktivointikynnystä ei ylitetä, neuroni ei aktivoidu eikä neuronilta saada uutta arvoa eli neuroni antaa tulokseksi arvon nolla. Aktivointikynnys riippuu täysin siitä, mikä aktivointifunktio on käytössä. Kuvassa 2 on havainnollistettu yhden keinotekoisen neuronin toiminta.



Kuva 2. Keinotekoinen neuroni (7).

Neuroverkko on minimissään kolmikerroksinen. Ensimmäinen kerros on aina syötekerros ja viimeinen on tuloskerros. Syöte- ja tuloskerroksen välissä on yksi tai useampi piilotettukerros. Jos piilotettuja kerroksia on useampi kuin yksi, kyseessä on syväneuroverkko. Lisäämällä piilotettuja kerroksia, voidaan verkolle opettaa enemmän ominaisuuksia tai tehdä siitä tarkempi. Kerrosten määrää kasvattamalla verkon koko kasvaa ja samalla koulutuksessa menee pidempään. Kuvassa 3 on mallinnettu yksinkertaista neuroverkkoa, jossa syötekerroksessa on kolme kappaletta neuroneita. Syötekerroksen neuronit ovat kaikki yhdistetty piilotetun kerroksen neljään neuroniin. Piilotetun kerroksen kaikki neljä neuronia ovat vastaavasti yhdistetty tuloskerroksen kahteen neuroniin. Esimerkin neuroverkko ottaa vastaan kolme syötettä ja luokittelee syötteet kahteen eri osaan. Syötteen eri piirteet aktivoivat eri neuroneja.



Kuva 3. Neuroverkko (8).

Neuroverkkomalleja voi olla useita erilaisia, mutta yksinkertaisin malli on eteenpäin syöttävä eli ”feed forward” -neuroverkko, jossa data menee suoraan neuronien läpi ja vertaa verkon luomaa tulosta haluttuun tulokseen. Aluksi verkon neuroneille luodaan satunnaisesti omat painot. Kun syötteestä saatua tulosta verrataan haluttuun lopputulokseen, painoja säädetään niin, että syötteen tulos olisi lähempänä toivottua tulosta. Neuroverkon tarkkuutta voi säätää eri tavoilla, mutta paras keino saada verkkoa tarkemmaksi on

iso määrää dataa. Jos dataa on liian pieni määrä, neuroverkko oppii testidatan liian hyvin, jolloin verkko ilmoittaa olevansa erittäin tarkka. Tällainen verkko ei kuitenkaan osaa luokitella uutta dataa hyvin. Tätä kutsutaan ylisovitukseksi. Vastaavasti alisovituksessa verkko ei kykene luomaan datasta mallia, jota se voisi käyttää datan luokituksessa. (9; 10, s. 24–26.)

3 Laitteisto

3.1 Auton ominaisuuksien esittely

Itsestään ajavan auton pohjana toimii radio-ohjattava auto Maverick Strada MT. Auto on tyypiltään nelivetoinen, ja ohjaavat renkaat ovat etuakselilla normaaliin tapaan. Auto on tarkoitettu ulkona ajettavaksi, mikä takia jokainen rengas on erillisjousitettu kahdella kierrejousella ja kahdella pneumaattisella iskunvaimentimella. Autolla on valmistajan mukaan huippunopeus 40 kilometriä tunnissa. Koska autolla on tarkoitus ajaa sisätiloissa koulutus- ja harjoitteluajo, on maksiminopeus rajoitettu ohjelmistollisesti. Auton voimanlähteenä toimii tasavirtamoottori, ja ohjaus on toteutettu servomoottorilla. Auto saa tehonsa 7,2-volttiselta nikkelimetallihydridiakulta (NiMH). Akun kapasiteetti on 1800 mAh, ja täydellä latauksella akku kestää noin 20 - 30 minuuttia. Käyttäjä antaa radio-ohjaimelta ohjauskomennon, joka menee auton vastaanottimelle. Elektroninen nopeudensäädin eli ESC ottaa vastaan komennot vastaanottimelta ja välittää moottorille ja servolle ohjaussignaalit.

Auto on 400 mm pitkä ja 310 mm leveä, ja sen akseliväli on 275 millimetriä. Renkaan halkaisija on 118 mm. Voimanvälityksen ratassuhde on 13,36:1. Sähkömoottorin malli on MM-22 540, servon malli on Maverick MS-22, vastaanottimen malli on MRX-22 27 mHz AM ja nopeudensäätimen malli on MSC-22. ESC:n käyttöjännite on 6 - 8,4 volttia. ESC:ssa on yli- ja alijännitesuoja, ylivirtasuojaus sekä suojaus ylikuumenemista varten. ESC:n PWM taajuus on 1,5 kHz. ESC sisältää 200 A FET:n ajamista varten ja 100 A

FET:n peruutusta varten. ESC:n BEC (battery eliminator circuit) jännite on 6 volttia ja tätä jännitettä käytetään vastaanottimen ja servon käyttöön. (11, s. 4.)

3.2 Raspberry Pi ja lisälaitteet

Raspberry Pi on monipuolinen ja yleisesti harrastajien ja hakkerien suosiossa ollut laite, jolla on mahdollista tehdä useita eri projekteja. Raspberryn halpa hinta on pitänyt sen suosittuna, ja sillä on aktiivinen yhteisö, jonka jäsenet jakavat omia projektejaan. Mallista riippuen Raspberryt sisältävät eri määrän GPIO-pinnejä, jotka ovat käyttäjän itse ohjelmoitavissa, USB-portteja, Ethernet-liitännän sekä video- ja ääniliitäntöjä. Projektissa on käytössä Raspberry Pi 3 model B. Model B:n tekniset tiedot ovat kuvassa 4.

Raspberry Pi 3 Model B

Specifications	
Processor	Broadcom BCM2387 chipset. 1.2GHz Quad-Core ARM Cortex-A53 802.11 b/g/n Wireless LAN and Bluetooth 4.1 (Bluetooth Classic and LE)
GPU	Dual Core VideoCore IV® Multimedia Co-Processor. Provides Open GL ES 2.0, hardware-accelerated OpenVG, and 1080p30 H.264 high-profile decode. Capable of 1Gpixel/s, 1.5Gtexel/s or 24GFLOPs with texture filtering and DMA infrastructure
Memory	1GB LPDDR2
Operating System	Boots from Micro SD card, running a version of the Linux operating system or Windows 10 IoT
Dimensions	85 x 56 x 17mm
Power	Micro USB socket 5V1, 2.5A
Connectors:	
Ethernet	10/100 BaseT Ethernet socket
Video Output	HDMI (rev 1.3 & 1.4 Composite RCA (PAL and NTSC)
Audio Output	Audio Output 3.5mm jack, HDMI USB 4 x USB 2.0 Connector
GPIO Connector	40-pin 2.54 mm (100 mil) expansion header: 2x20 strip Providing 27 GPIO pins as well as +3.3 V, +5 V and GND supply lines
Camera Connector	15-pin MIPI Camera Serial Interface (CSI-2)
Display Connector	Display Serial Interface (DSI) 15 way flat flex cable connector with two data lanes and a clock lane
Memory Card Slot	Push/pull Micro SDIO

Kuva 4. Raspberry Pi 3 Model B:n tekniset tiedot (12, s. 2).

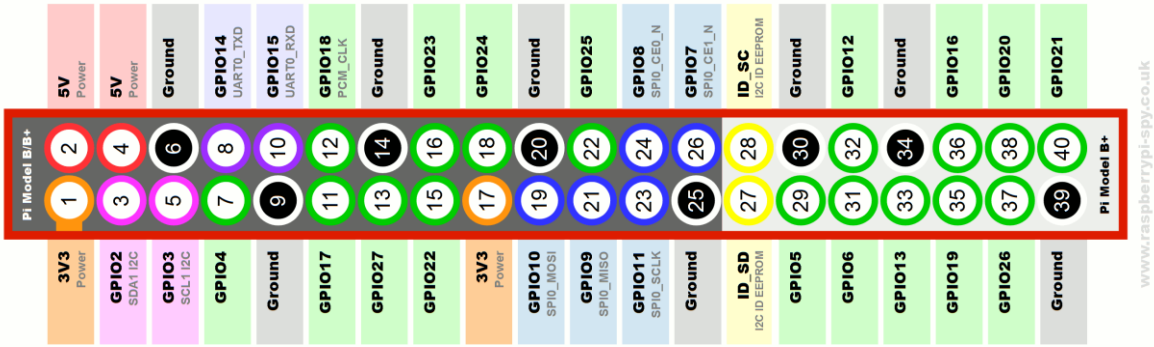
Raspberryyn on asennettu 64-gigatavuinen micro SDXC -tyyppinen muistikortti, joka on formatoitu FAT32-tiedostomuotoon. Yli 32 Gt:n muistikortit on oletuksena formatoitu exFAT-tiedostomuotoon, minkä takia projektissa käytettävä muistikortti piti erikseen formatoida FAT32-muotoon. Raspberrylle erikseen luotu Linux Debian variantin asennusmedia ei tue exFAT-tiedostomuotoa. Raspberrylle luodusta Debian Linux -versiosta käytetään yleisesti nimeä Raspbian. (13; 14; 15)

Koska projekti käyttää hyödykseen visuaalista dataa, Raspberryyn on liitetty kameramoduuli. Kameramoduuli on Raspberryn virallinen kamera, joka menee sille erikseen tehtyyn CSI-2-liittimeen kiinni. Kameran virallinen mallinimi on Camera Module V2, joka sisältää Sonyn valmistaman IMX219-sensorin. Sensori on 8-megapikselinen, ja se tukee 1080 p:n tarkkuutta 30 kuvaa sekunnissa ja 720 p:n tarkkuutta 60 kuvaa sekunnissa. Raspberry tukee myös USB-käyttöisiä web-kameroita.

Raspberryn virtalähteenä on käytössä 3350 mAh:n akku. Akku on tarkoitettu kännykän varavirtalähteeksi. Akun jännite on 5 volttia, ja se antaa 1 ampeerin virtaa. Virranmäärä jää teknisten tietojen suosituksen alle selkeästi, mutta on kuitenkin riittävä, kun otetaan huomioon, että Raspberryyn on kytketty vain muutama lisälaite ja autolla on oma akku. Auton akkua voi myös hyödyntää Raspberryn virtalähteenä, mutta se vaatii erillisen jänniteregulaattorin, koska ESC: n BEC jännite on liian suuri suoraan Raspberryn käytettäväksi. Riskinä on myös, että servon ja moottorin hetkittäinen tehon tarve johtaa jännitteen putoamiseen, mikä voi sammuttaa Raspberryn. Erillisen akun käyttö on tässä tapauksessa yksinkertaisin ja kustannustehokkain vaihtoehto.

3.3 GPIO

Raspberry Pi 3 Model B:ssä on yhteensä 40 kappaletta GPIO-pinnejä. Kuvassa 5 on eritelty pinnien ominaisuudet.



Kuva 5. RPi model 3B -pinnit (16).

GPIO-pinneistä saa tarvittaessa 5 V:n tai 3,3 V:n tasajännitteen, ja pinnirivistössä on myös maapisteet. GPIO-pinnit voivat olla HIGH- tai LOW-asennossa, jolloin HIGH-asento antaa 3,3 V:n jännitteen LOW-asento antaa 0 V:n jännitteen. Pinnit voivat olla IN- tai OUT-tilassa, jossa Raspberry joko tuottaa tai ottaa vastaan signaalin. Pinnejä voi kutsua niiden fyysisen sijainnin mukaan tai piirisarjan numeroinnin mukaan. GPIO 12- ja GPIO 13 -pinnejä voi käyttää myös PWM-tilassa. Yksittäinen pinni antaa turvallisesti maksimissaan 16 mA:n virtaa ja GPIO-pinneistä voi yhteensä ottaa turvallisesti 50 mA (17; 18).

3.4 Tietokone

Projektissa käsitellään suurta määrää dataa. Käsittely vaatii paljon laskenta tehoa ja muistia, eikä Raspberryssä ole riittävästi kumpaakaan, minkä takia laskenta suoritetaan pöytätietokoneella. Taulukossa 1 on eritelty koneen tekniset tiedot.

Taulukko 1. Tietokoneen tekniset tiedot.

Prosessori	Intel Core i7-6700K
Näytönohjain	Nvidia GTX 1080, 8 Gt GDDR5X
Muisti	16 Gt DDR4 2666 MHz
Massamuisti	Samsung 950 Pro m.2 SSD 256 Gt
Käyttöjärjes-telmä	Windows 10 pro 64-bit

Laskennassa hyödynnetään sekä prosessoria että näytönohjainta. Prosessorin kellotaajuus on nostettu Intelin 4,0 GHz:n turbotaajuudesta 4,5 GHz:n taajuuteen kaikilla ytimillä. Näytönohjain on MSi:n versio, joka toimii rasituksessa noin 1900 MHz:n taajuudella.

Näytönohjaimen laskentateho on Nvidian peruskellotaajuudella 8,9 Tflopsia. Näytönohjaimessa on 2560 kappaletta CUDA-ytimiä, joita hyödynnetään neuroverkon mallin koulutuksessa. Tuhansien rinnakkaisten ytimien ansiosta neuroverkon koulutus voi olla jopa 10 kertaa nopeampaa kuin prosessorilla. Nopeuseroihin vaikuttavat verrattavat komponentit. (19)

4 Auton modifikaatio

Autossa oleva radiovastaanotin korvattiin Raspbery Pillä. Ennen muutosta vastaanottimelta ESC:lle tulevat ohjaussignaalit mitattiin yleismittarilla. Vastaanotin antaa ESC:lle ohjaussignaalin renkaiden kääntösäteestä ja moottorin nopeudesta sekä pyörimissuunnasta. Taulukosta 2 nähdään mitatut servon ohjausjännitteet.

Taulukko 2. Servon jännitearvot.

Servon asento	Ohjausjännite (V)
Täysin oikealle	0,20
Keskiasento	0,25
Täysin vasemmalle	0,30

Servon käyttöjännitteeksi mitattiin 5,11 volttia. Renkaat ovat keskiasennossa, kun ohjausjännite on 0,25 volttia, täysin oikealle käännettynä 0,2 V:ssa ja täysin vasemmalle käännetty ohjausjännitteen ollessa 0,3 V. DC-moottorille menevän käyttöjännitteen arvoksi mitattiin 7,26 - 7,5 volttia, kun moottori oli täysin kuormitettu. Taulukossa 3 on listattu mitatut moottorin ohjausjännitteet.

Taulukko 3. Moottorin ohjausjännitteet.

Moottorin tila	Ohjausjännite (V)
Levossa	0,28

Täysin eteenpäin	0,39
Täysin taaksepäin	0,22

ESC:n kytkentäkaaviota ei valitettavasti ole saatavilla, mutta se todennäköisesti toimii kuin h-silta, jossa transistorit avautuvat kynnysjännitteen saavutettua. Muuttamalla jännitettä matalammaksi, matalanpuolen transistorit avautuvat ja aiemmin avoinna olevat korkean jännitteen puolen transistorit sulkeutuvat, jolloin virran kulkusuunta vaihtuu ja moottori pyörii vastakkaiseen suuntaan.

Auton vastaanottimen ohjaussignaalien johdot yhdistettiin Raspberry Pin GPIO 12- ja GPIO 13 -pinniin, jotka molemmat kykenevät tuottamaan PWM-signaalia. PWM-signaalilla voidaan dynaamisesti vaikuttaa auton ohjauskulmiin ja moottorin pyörintänopeuteen. Liitteessä 1 nähdään auton kytkentäkaavio muutosten jälkeen. ESC:n ohjaussignaalin johto on yhdistetty GPIO 12 -pinniin ja servon ohjaussignaali vastaavasti GPIO 13 -pinniin. Servo saa edelleen käyttöjännitteensä auton akulta ESC:n kautta. ESC ja Raspberry on yhdistetty yhteiseen maapisteeseen. ESC:llä on oma pääkatkaisin.

Autoon on lisätty ultraäänisensori, joka pystyy lukemaan etäisyyden esteisiin. Ultraäänisensori saa käyttöjännitteensä Raspberryn akulta. Echo-pinni on yhdistetty Raspberryn GPIO 23 -pinniin, joka on OUPUT-tilassa. Trigger-pinni on yhdistetty GPIO 24-pinniin, joka on INPUT-tilassa. Koska ultraäänisensorin trigger-signaali tuottaa 5 voltin jännitteen, täytyi jännitettä alentaa asentamalla 1 kΩ resistori ennen Raspberryn liitäntää ja asentaa vielä 2 kΩ alasvetoresistori maahan. Resistorin R_2 arvo lasketaan käyttämällä kaavaa $V_{\text{out}} = V_{\text{in}} \cdot \frac{R_2}{R_1+R_2}$. Kun tiedetään, että $V_{\text{out}} = 5\ \text{V}, V_{\text{in}} = 3{,}3\ \text{V}$, saadaan R_1:n arvoksi 1 kΩ ja R_2:n arvoksi 2 kΩ, koska $R_1 \approx \frac{R_2}{2}$.

5 Käytössä olevat ohjelmistot

Projektissa on käytössä useita eri ohjelmistoja ja kirjastoja. Projekti ohjelmoitiin Python 3 -kielellä. Python on kielenä helppo, ja se sopii hyvin aloitteleville ohjelmoijille. Pythonia käytettiin niin Raspberryllä kuin pöytätietokoneella. Raspberrylle asennetussa Raspbian-käyttöjärjestelmässä Python oli esiasennettuna, mutta pöytätietokoneelle se piti asentaa erikseen. Raspbian-käyttöjärjestelmä on ladattavissa verkosta. (13)
Python 3 ladattiin https://www.python.org/downloads/-sivulta.Käytössä oleva versio on Python 3.6.5, ja ohjelmasta asennettiin 64-bittinen versio.

5.1 NumPy

NumPy on Pythonille asennettava paketti, ja se on luotu tehokkaaseen moniulotteisien taulukoiden prosessointiin ja manipulointiin. NumPylla voidaan myös säilyttää dataa. NumPy asennetaan Windowsin komentorivillä komennolla

”pip install numpy”.

Numpy saadaan Pythonissa käyttöön komennolla

“import numpy as np”.

NumPya käytetään projektissa kerätyn datan säilytykseen ja käsittelyyn. (20)

5.2 OpenCV

OpenCV on avoimenlähdekoodin kirjasto, joka mahdollistaa konenäön. OpenCV mahdollistaa muun muassa reunojen havaitsemisen ja videon muokkauksen. OpenCV:tä voidaan käyttää kameralla tai suoraan tietokoneen työpöydällä.
OpenCV asennetaan Windowsin komentorivillä komennolla

”pip install opencv-python”.

OpenCV saadaan käyttöön Pythonissa komennolla

”import cv2”.

Projektissa OpenCV:llä luetaan ja tallennetaan Raspberrylta tulevaa videon suoratoistoa. (21)

5.3 Tornado ja socket

Tornado ja Socket ovat verkkokommunikointiin tarkoitettuja kirjastoja Pythonille. Socket on Pythonin mukava tuleva moduuli, mutta Tornado pitää asentaa erikseen.

Tornado asennetaan Raspberryn ja Windowsin komentorivillä komennolla:

”pip install tornado” -Windows versio

“pip3 install tornado” -Raspberry versio.

Tornado otetaan käyttöön esimerkiksi komennoilla

“import tornado.ioloop”

”import tornado.web”.

Socket saadaan käyttöön komennolla

“import socket”.

Tornadoa ja Socketia käytetään Raspberryn ja pöytätietokoneen välisessä kommunikoinnissa langattoman verkon välityksellä. (22; 23)

5.4 RPi Cam Web Interface

RPi Cam Web Interface on ohjelma, joka suoratoistaa videota Raspberrysta. Ohjelma toimii ainoastaan Raspberryn kameramoduulilla eikä tue USB-kameroita. Suoratoisto on katsottavissa verkkoselaimen kautta. Ohjelma tukee lisäksi videon nauhoitusta, yksittäisten kuvien tallennusta ja liikkeentunnistusta.

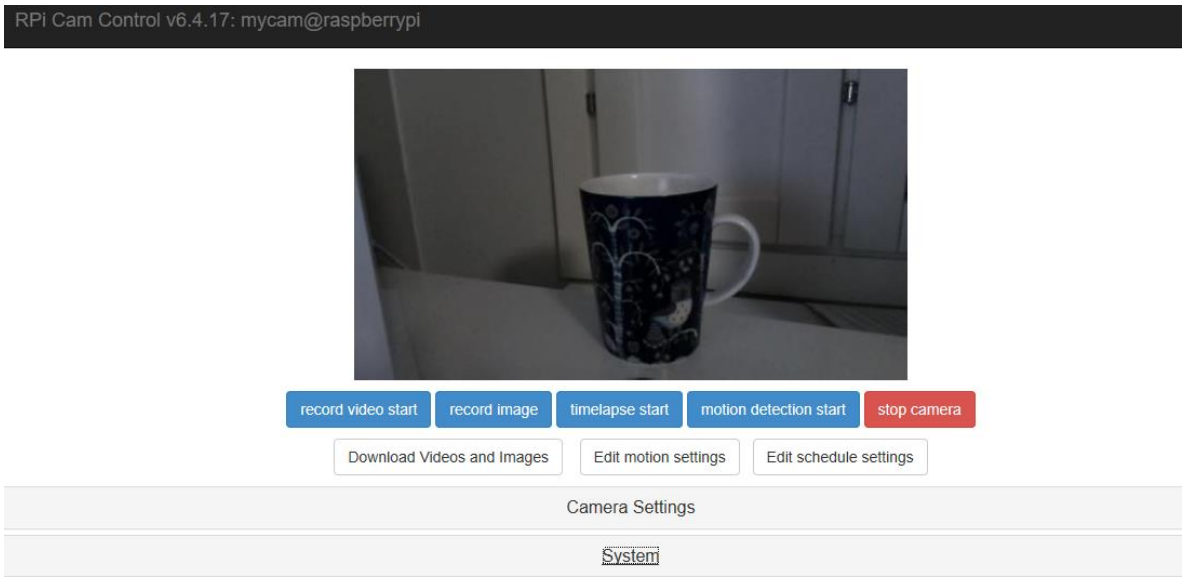
Ohjelma asennetaan Raspberryn komentorivillä komennoilla

“git clone https://github.com/silvanmelchior/RPi_Cam_Web_Interface.git”

“cd RPi_Cam_Web_Interface”

“./install.sh”.

Kuvassa 6 nähdään ohjelman käyttöliittymä.



Kuva 6. RPi Cam Web Interface.

Ohjelma käynnistyy automaattisesti Raspberryn käynnistyksen yhteydessä, ja sitä käytetään verkkoselaimen kautta. Ohjelmaan pääsee käsiksi kirjoittamalla selaimen osoitekenttään http://192.168.1.105/html/. Suoratoiston resoluutio on 512 x 288 pikseliä ja kuvasuhde on 16:9. (24)

5.5 Tensorflow

Tensorflow on alun perin Googlen luoma nykyään avoimen lähdekoodin kirjasto, jolla voidaan suorittaa numeerista laskentaa. Tensorflow tukee laskennan tekemistä prosessorilla ja näytönohjaimella. Tensorflow'lla luodaan ja koulutetaan projektissa käytössä oleva neuroverkko. Tensorflow vaatii 64-bittisen version Pythonista toimiakseen. Tensorflow asennetaan Windowsin komentorivillä komennolla

"pip install tensorflow-gpu".

Tensorflow'n saa Pythonissa käyttöön komennolla

"import tensorflow as tf".

Tensorflow'sta asennettiin versio, joka tukee näytönohjainta. Kuvassa 7 on listattu laitevaatimukset ja ohjelmistovaatimukset GPU-tuelle.

Hardware requirements

The following GPU-enabled devices are supported:

- NVIDIA® GPU card with CUDA® Compute Capability 3.5 or higher. See the list of CUDA-enabled GPU cards.

Software requirements

The following NVIDIA® software must be installed on your system:

- NVIDIA® GPU drivers —CUDA 9.0 requires 384.x or higher.
- CUDA® Toolkit —TensorFlow supports CUDA 9.0.
- CUPTI ships with the CUDA Toolkit.
- cuDNN SDK (>= 7.2)
- *(Optional)* NCCL 2.2 for multiple GPU support.
- *(Optional)* TensorRT 4.0 to improve latency and throughput for inference on some models.

Kuva 7. Tensorflow GPU:n laitevaatimukset (25)

Käytössä oleva GTX 1080 tukee CUDA Compute Capability versiota 6.1. GPU-ajurit ovat ladattavissa Nvidian sivulla https://www.nvidia.com/Download/index.aspx?lang=en-us ja CUDA Toolkit on ladattavissa sivulta https://developer.nvidia.com/cuda-zone. Viimeinen pakollinen vaatimus cuDNN SDK vaatii Nvidian sivulle rekisteröitymistä. Rekisteröitymisen jälkeen sivulta voi ladata cuDNN:n osoitteesta https://developer.nvidia.com/cudnn.

Sivulta ladattavat cuDNN:n pitää olla yhteensopiva Tensorflow'n CUDA-version kanssa. Zip-tiedosto pitää purkaa CUDA-asennuskansioon. Paketissa pitää olla *cuDNN64_7.dll*-niminen tiedosto, koska Tensorflow ei lataudu ilman sitä. CUDA, CUPTI ja cuDNN pitää asennuksen jälkeen vielä lisätä Windowsin ympäristömuuttujan alle (engl. environmental variable). Liittäminen tehdään Windowsin komentorivillä komennoilla (25)

"SET PATH=C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v9.0\bin;%PATH%"
"SET PATH=C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v9.0\extras\CUPTI\libx64;%PATH%"
"SET PATH=C:\tools\cuda\bin;%PATH%".

Kuvassa 8 on selvitetty Tensorflow'n versionumero ja kokeiltu, tunnistaako Tensorflow näytönohjaimen. Tensorflow antaa versionumeroksi 1.7.0. GPU:n saatavuuteen saadaan arvoksi True, ja CUDA:n testaus palauttaa myös arvon True. Laitelistauksessa näytönohjain tunnistautuu oikein.

```
>>> import tensorflow as tf
>>> print(tf.__version__)
1.7.0
>>> tf.test.is_gpu_available(
    cuda_only=False,
    min_cuda_compute_capability=None)
True
>>> tf.test.gpu_device_name()
'/device:GPU:0'
>>> tf.test.is_built_with_cuda()
True
>>> from tensorflow.python.client import device_lib
>>> print(device_lib.list_local_devices())
[name: "/device:CPU:0"
device_type: "CPU"
memory_limit: 268435456
locality {
}
incarnation: 13251306083693750108
, name: "/device:GPU:0"
device_type: "GPU"
memory_limit: 6594656337
locality {
  bus_id: 1
  links {
  }
}
incarnation: 753900380912961450
physical_device_desc: "device: 0, name: GeForce GTX 1080, pci bus id: 0000:01:00.0,
compute capability: 6.1"
]
>>>
```

Kuva 8. Tensorflow-asennuksen testaus.

5.6 TFlearn

TFlearn on Tensorflow'n päälle rakennettu korkean tason API. TFlearnin ideana on olla helppokäyttöinen ja suoraviivaistaa neuroverkon luontia sekä pysyä yhteensopivana Tensorflow'n kanssa. TFlearn asennetaan Windowsin komentorivillä komennolla

"pip install tflearn".

TFlearn otetaan käyttöön komennolla

"import tflearn". (26)

6 Datan keräys

Koulutusdatan keräykseen käytetään autosta tulevaa videokuvaa. Autolla ajetaan manuaalisesti internetin välityksellä. Ajon aikana nauhoitetaan videokuvaa ja tallennetaan komennot tekstitiedostoon. Auton ja tietokoneen kellot täytyy olla samassa ajassa, jotta komentojen ja videokuvan paritus onnistuu oikein. Datan keräyksessä ja muokkauksessa hyödynnetään Ryan Zottin luomaa itsestään ajavan auton mallia, mitä muokataan omaa käyttötarkoitusta varten sopivaksi. Ryan Zottin koodi on nähtävissä hänen Github-sivultaan. (27)

Käytössä oleva data vaikuttaa eniten neuroverkon laatuun. Kerätylle datalle annetaan kolme luokkaa auton komentojen mukaan: vasen, suoraan ja oikea. Luokat paritetaan kuvaan aikaleiman perusteella, jonka jälkeen kaikki data yhdistetään yhdeksi taulukkotiedostoksi, minkä jälkeen data on oikeassa muodossa neuroverkkoon syötettäväksi.

6.1 Manuaaliajo

Manuaalisessa ajossa käytetään hyödyksi Tornado-kirjastoa Pythonilla. Tornadolla luodaan web-sovellus, jota käytetään verkkoselaimella. Sovellus ottaa vastaan pöytätietokoneelta tulevat komennot ja lisäävät ne tekstitiedostoon. Sovelluksella ohjataan auton servoa ja moottoria PWM-signaalilla Raspberryn GPIO-pinnien välityksellä. Liitteessä 2 on koko koodi manuaalista ajamista varten. Sovellus käynnistetään Raspberrylla komennolla

”sudo python3 manuaali_ajo.py --speed_percent 100”.

Sovelluksella pääsee ajamaan autoa menemällä selaimella osoitteeseen 192.168.1.105:5555/drive. Sovellus ottaa tietokoneelta neljä komentoa vastaan, jotka ovat näppäimistössä näppäimet ←, ↑, → ja ↓. Komennoilla auto menee vasemmalle, menee suoraan, menee oikealla ja peruuttaa. Sovellus pystyy ottamaan vastaan yhden komennon kerrallaan, minkä takia ”vasen” ja ”oikea” komennot liikuttavat autoa myös eteenpäin kääntäen renkaita samalla. (27)

Raspberryn GPIO laitetaan BCM-tilaan ja aktivoidaan pinnit 12 ja 13. Pinniä 12 käytetään moottorin ohjaukseen ja pinni 13 on servon ohjausta varten. Pinnit laitetaan ulostulotilaan komennoilla

"GPIO.setup(self.pinMotor, GPIO.OUT)"

"GPIO.setup(self.pinServo, GPIO.OUT)".

Pinnit laitetaan PWM-tilaan komennoilla, joissa sulkeiden sisällä oleva ensimmäinen argumentti valitsee pinnin numeron ja toinen argumentti valitsee PWM:n taajuuden

"self.pwm_motor = GPIO.PWM(self.pinMotor, 50)"

"self.pwm_servo = GPIO.PWM(self.pinServo, 50)".

PWM:n taajuudeksi valittiin 50 Hz iteroimalla eri taajuuksia, ja 50 Hz tuotti parhaan tuloksen. Lopulta PWM-signaali aloitetaan komennoilla

"self.pwm_motor.start(6.6)"

"self.pwm_servo.start(7.5)".

PWM-signaalia säädetään muuttamalla käyttöjaksoa. Kun käyttöjakso on 100 %, PWM-signaali on jatkuvasti 3,3 V ja vastaavasti käyttöjakson ollessa 0 % signaali on jatkuvasti 0 V. Huippunopeuden saavuttamiseksi ESC vaatii vain 0,39 V. Tästä voidaan laskea autolle maksimikäyttöjakso:

$$\frac{0.39}{3.3} \times 100\ \% = 11{,}8\ \%.$$

Koska 40 km/h on aivan liian suuri nopeus sisätilassa ajettavaksi, nopeutta rajoitetaan käyttämällä käyttöjaksoa 8,4 %, joka
vastaa jännitettä 0,28 V. Mikäli nopeus kasvaa lineaarisesti jännitteen noustessa, voidaan nopeuden kasvu laskea karkeasti seuraavasti
$\frac{0.39\ \mathrm{V} - 0.28\ \mathrm{V}}{40\ \mathrm{km/h}} = 2{,}7\,\frac{\mathrm{mV}}{\mathrm{km/h}}$. Mittausten mukaan 8,4 %:n käyttöjaksolla auton ei pitäisi lähteä liikkeelle, mutta kokeilemalla eri käyttöjaksoja, auto liikkuu jo 7,7 %:n käyttöjaksolla. Ero mittausten ja testauksen kohdalla voi johtua joko heikkolaatuisesta yleismittarista, tai alkuperäisen vastaanottimen ohjaussignaali voi olla muodoltaan erilainen, ja ESC reagoi eri tavalla saadessaan PWM-signaalia.
Vastaavalla tavalla saadaan laskettua servon asennot:

Vasemmalle: $\frac{0.30}{3.3} \times 100\ \% = 9\ \%.$

Keskellä: $\frac{0.25}{3.3} \times 100\ \% = 7{,}5\ \%.$

Oikealle: $\frac{0.20}{3.3} \times 100\ \% = 6\ \%$.

Auton akun varauksella on vaikutus renkaiden keskikohtaan ja liikkeellelähtöön.

6.2 Session tallennus

Web-sovelluksen ollessa käynnissä Raspberrylla, pöytätietokoneessa käynnistettään sovellus tallenna_striimi.py. Liitteessä 3 on sovelluksen koko koodi. Suoratoiston tallennuksessa luetaan Raspberrylta tuleva videokuva ja tallennetaan se .mov-tiedostomuotoon. Jokaisen yksittäisen luetun kuvan yhteydessä kuvalle luodaan aikaleima tekstitiedostoon. Video on resoluutiolta 512 x 288, ja videon nopeus on 20 kuvaa sekunnissa. Suoratoisto luetaan ja tallennetaan käyttämällä OpenCV-kirjastoa Pythonilla. Sovelluksen IP antaa yhden kuvan, jonka takia kuva pitää silmukassa lukea joka kerta uudestaan.

Rivillä *” fourcc = cv2.VideoWriter_fourcc(*'jpeg')”* määritellään käytössä oleva muoto. Seuraavalla rivillä *” out = cv2.VideoWriter('output.mov',fourcc, 20.0, (512,288))”* määritellään videon nimi, muoto, kuvataajuus ja resoluutio.
Silmukassa netistä luetaan kuva riveillä

“cap = cv2.VideoCapture(ip)”

“ret, frame = cap.read()”.

Kuva tallennetaan rivillä

”out.write(frame)”.

Kun sessio lopetetaan, siirrytään selaimessa osoitteeseen 192.168.1.105/StoreLogEntries. (27; 28) Tämän jälkeen manuaali_ajo.py-sovelluksen voi sulkea ja tallenna_striimi.py-sovellus suljetaan painalla näppäintä ”q” Pythonin videoikkunan ollessa aktiivinen. Manuaali_ajo.py luo tiedostot session.txt ja clean_session.txt ja tallenna_striimi.py-sovellus luo output.mov-videotiedoston sekä video_timestamps.txt-tekstitiedoston.

6.3 Koulutusdatan luonti

Sessiosta saatu data pitää muuttaa muotoon, joka voidaan syöttää neuroverkkoon. Kaikki 4 sessiosta saatua tiedostoa pitää siirtää samaan kansioon. Muodonmuutos tapahtuu sovelluksella save_all_runs_as_numpy_files.py. Sovellus ottaa videotiedoston ja teksti tiedostot ja yhdistää ne isoksi NumPy-tiedostoksi. Session tallennuksesta saadut tiedostot ovat yhteensä noin 113 MB. Tulokseksi tuleva NumPy-tiedosto on kooltaan 5,7 Gt, mikä on yli 50 kertaa suurempi tiedosto.

Ensimmäiseksi värikuvaa varten luodaan gammataulukko, joka tapahtuu funktiossa *”make_gamma_tables”*. Datan prosessointi, joka tehdään funktioissa process_session, aloitetaan luomalla videon aikaleimoista lista, josta poistetaan rivinvaihto. Clean_session.txt:stä luodaan komentolista. Komentolistasta jätetään pois kaikki väärät komennot, luodaan riville indeksi ja lisätään komentolistaan komento ja aikaleima. Lopuksi komennoista vielä poistetaan tyhjät komennot ja peruuttaminen. Listan alkuun ensimmäinen komento ja aikaleima määritellään aloituskohdaksi ja viimeisen aikaleiman jälkeen lisätään lopetusajankohta.

Lopullista tiedostoa varten luodaan kaksi listaa: ”predictors” ja ”targets”. Silmukassa avataan video luettavaksi ja luodaan laskuri kuville ja komennoille. Luetut komennot jaetaan kolmeen osaan, joka muotoillaan seuraavasti

target = [0, 0, 0].

Listassa ensimmäinen kohta edustaa vasenta, toinen kohta edustaa eteenpäin ja viimeinen kohta on oikea. Luettu komento saa numeroksi 1 ja muut numerot pysyvät nollana. Jokainen luettu komento näyttää seuraavalta:

Vasen:	target = [1, 0, 0]
Suoraan:	target = [0, 1, 0]
Oikea:	target = [0, 0, 1].

Kun videon kuva on saatu ja säädetty gammataulukon mukaan, lisätään kuva predictions-listaan ja komento lisätään targets-listaan. Kun kaikki kuvat ja komennot on käyty läpi ja lisätty listoihin, tallennetaan listat yhdeksi tiedostoksi funktiossa ”video_to_rgb_npz”, jolloin tulokseksi saadaan predictors_and_targets.npz-NumPy-tiedosto. Liitteessä 4 on koodi koulutusdatan luontia varten. (27)

7 Neuroverkko

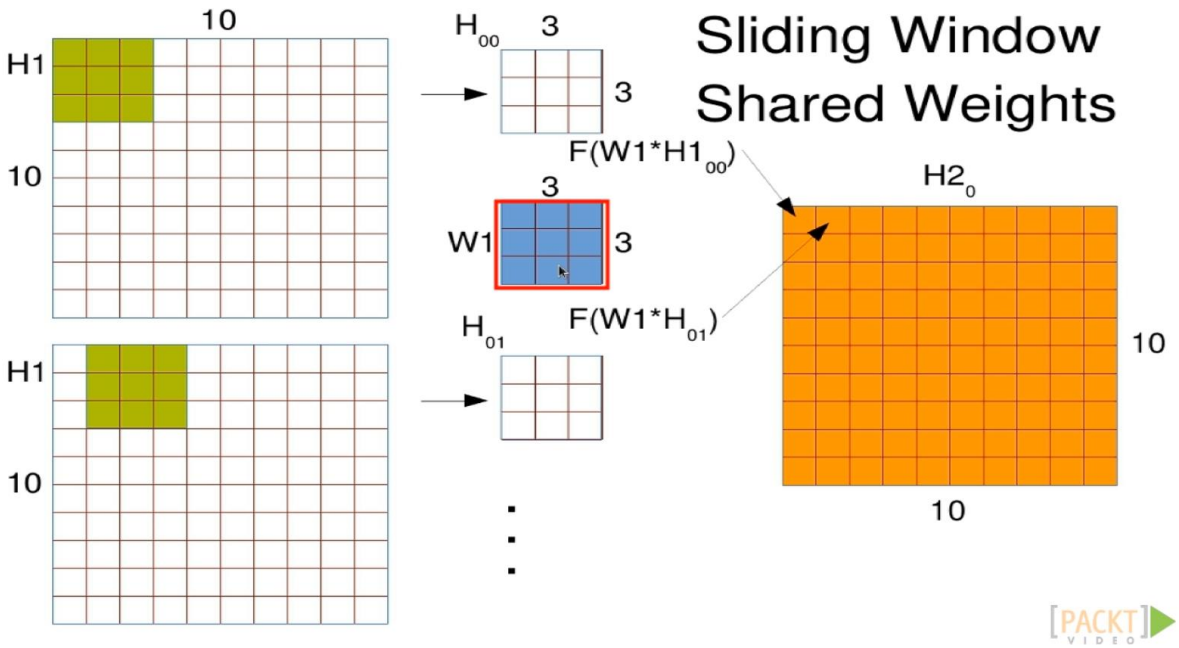
Neuroverkon luontiin käytetään Tflernia. Luodussa neuroverkon mallissa on syötekerros, kaksi kappaletta konvoluutiokerroksia, kaksi kappaletta max pool -kerroksia ja lopulta yksi tuloskerros. Syötteenä neuroverkkoon käytetään Raspberrystä tallennettua kuvaa ja nimikettä. Tulokseksi neuroverkko antaa kolme luokittelua kuvalle. Näitä luokitteluja hyödynnetään itsestään ajavan auton päätöksenteossa. Tflearnilla voidaan tehdä vastaava neuroverkko huomattavasti pienemmällä määrällä koodia, eikä eri ominaisuuksia tarvitse itse määritellä, vaan ne ovat suoraan kutsuttavissa. Haittapuolena voidaan pitää pienempää kustomointi mahdollisuutta.

Kaikista erilaisista neuroverkon muodoista projektiin valittiin käyttäväksi konvoluutioneuroverkko, koska sen on todettu olevan paras kuvantunnistuksessa. Projektin konvoluutioneuroverkossa on vain muutama kerros, koska projektiin riittää ajoalueen reunojen havaitseminen eikä tarkkojen piirteiden havaitseminen ole tärkeää. Pienellä kerrosmäärällä voidaan pitää malli myös kooltaan pienenä. (10, s. 14.)

7.1 Konvoluutio (convolutional layer)

Konvoluutio on tällä hetkellä kaikkein tehokkain tapa oppia neuroverkolla, mitä kuvassa on. Konvoluutiokerroksella on leveyden ja pituuden lisäksi myös syvyys. Konvoluution ideana on, että kuva analysoidaan alue kerrallaan. Esimerkiksi voidaan ottaa 40 x 40 -kuva, jota analysoidaan 7 x 7 -alueella. Aluetta liikutetaan kuvan yli vasemmalta oikealle, ylhäältä alas. Aluetta liikutetaan yksi pikseli kerrallaan, kunnes koko kuva on käyty läpi. Yhden pikselin liikutuksesta käytetään nimeä ”stride”. 7 x 7 -alueella olevat pikselit summataan ja kerrotaan painojen kanssa, minkä jälkeen tulo viedään aktivointifunktion läpi ja tulos säilytetään uuteen konvoluutio matriisiin. Kun osa ikkunasta menee kuvan yli, käytetään yli menevien pikselien kohdalla tilapäisiä arvoja. Tilapäisten arvojen käytöstä ylimenevällä osalla käytetään nimeä ”same padding”. Samasta kuvasta eri ominaisuuksia haettaessa käytetään joka ominaisuudella omia painomatriiseja. Lisätyt painomatriisit

vastaavat lisättyjä neuroneja tiheästi yhdistetyssä kerroksessa. Jokainen painomatriisi alustetaan erikseen. Kuvassa 9 esitetään konvoluutiokerroksen toimintaa.



Kuva 9. Konvoluutiokerroksen muodostus (29).

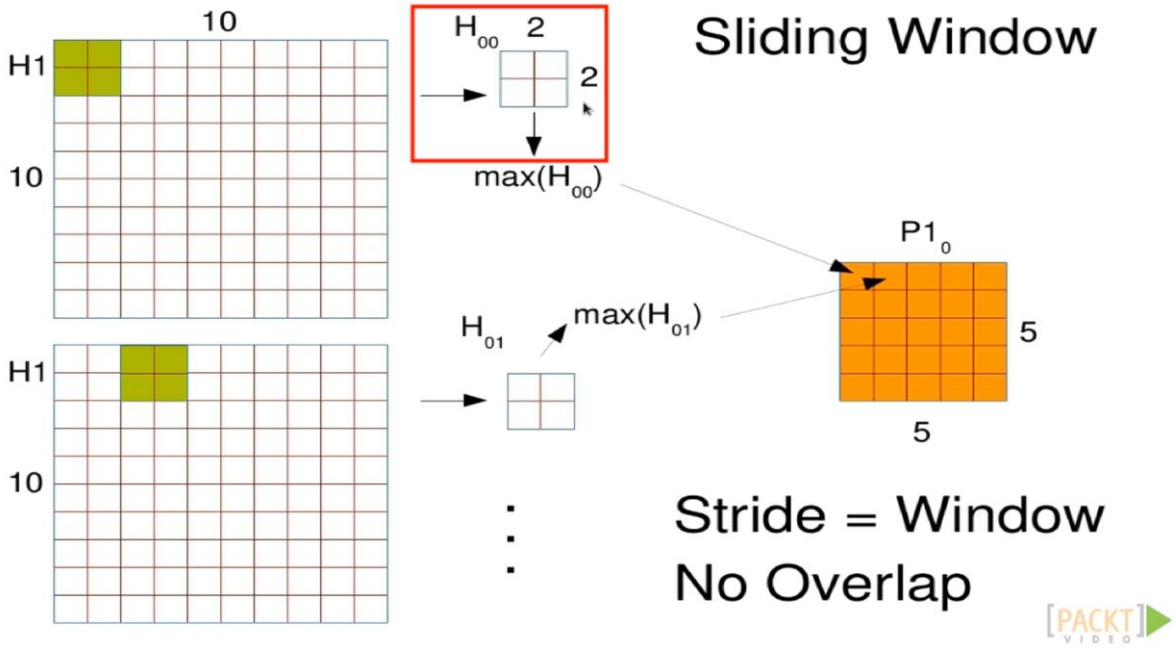
Konvoluutiosta voi ottaa useita ominaisuuksia ulos, mutta konvoluutioon voi myös syöttää useita ominaisuuksia. Esimerkiksi värikuvaa käytettäessä konvoluutioverkkoon syötetään sinisiä arvoja sisältävä matriisi, vihreitä arvoja sisältävä matriisi ja punaisia arvoja sisältävä matriisi. (29; 6, s. 67-73; 30, s. 24-39.)

7.2 Yhdistyskerros (max pooling layer)

Pooling-kerrosta käytetään yleensä konvoluutiokerroksen yhteydessä. Toisin kuin konvoluutio alueessa pooling-alue ei käytä samoja pikseleitä useaan otteeseen vaan siirtyy koko alueensa verran oikealta vasemmalle ja ylhäältä alas. Pooling-alue ei myöskään ylitä kuvan reunoja. Jos kuva on esimerkiksi 10 x 10 pikseliä ja pooling-alue on 2 x 2 pikseliä, saadaan pooling-matriisiksi 5 x 5 -alue. Pooling-kerrokseen ei yleensä käytetä painoja. 9 x 9 -kuva antaa samalla alueella 4 x 4 -lopputuloksen, koska reunojen ylitystä ei sallita. Pooling-alueesta valitaan alueen suurin arvo ja heitetään loput pois. Suurin

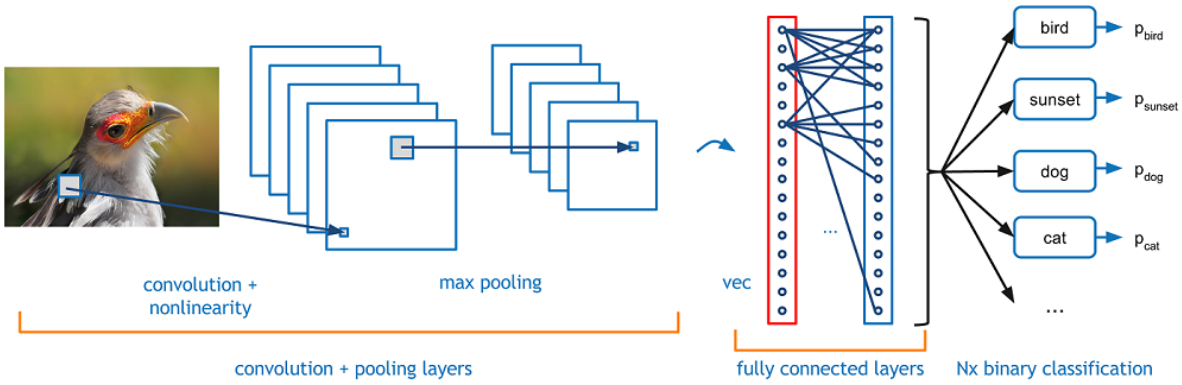
arvo sijoitetaan uuteen pooling-matriisiin. Pooling toimii parhaiten yhdessä konvoluution kanssa. Aluksi konvoluutio etsii kuvasta ominaisuuksia, jonka perään laitetaan max pooling -kerros. Max pooling -kerros ottaa konvoluutiomatriisista alueen, jossa ominaisuus on kaikkein voimakkain ja supistaa alueen neljäsosaan. Siinä missä konvoluutiosta saadaan tulokseksi väri voimakkuuksia, pooling-alueesta saadaan tulokseksi ominaisuus voimakkuuksia. Poolingin tulosta voidaan ajatella uutena kuvana, jolle voidaan tehdä uusi konvoluutiokerros ja uusi pooling-kerros. Jokaisen kerroksen yhteydessä lopullinen kuva supistuu neljänneksen, kun käytössä on 2 x 2 -pooling-alue.

Verkon alussa olevat konvoluutiopainot yleensä kouluttautuvat havaitsemaan yksinkertaisia piirteitä kuten reunoja. Useamman kerroksen jälkeen verkko oppii yhä monimutkaisempia piirteitä ja voi lopulta oppia tunnistamaan kissan tai muita esineitä. Kuvasta 10 nähdään, miten pooling-kerros toimii.



Kuva 10. Max pooling -kerros (31).

Lisäämällä konvoluutio- ja max pooling -kerrosten perään vielä täysin yhdistetyn kerroksen ja tulo kerroksen, saadaan kuvan 11 kaltainen lopputulos (31).



Kuva 11. Konvoluutioneuroverkko (10).

Kuvassa 11 kuva linnusta syötetään konvoluutiokerrokseen, josta tehdään max pooling -kerros. Pooling-kerros on yhdistetty täysin yhdistettyyn kerrokseen, josta viimein tulokerros luokittelee kuvan. (10; 30, s. 25; 32.)

7.3 Kerätyn datan tuominen verkolle

Aiemmin tehty koulutusdata ladataan ohjelmassa. Data on NumPy -taulukkomuodossa. ”Predictors” ja ”targets” -taulukot jaotellaan kuvat ja nimikkeet muuttujien alle. Seuraavaksi tuodut kuvat ja nimikkeet sekoitetaan. Sekoituksella estetään ei-toivottujen taipumusten muodostumista, mikä voi johtua useasta samanlaisesta kuvasta peräkkäin datassa. Sekoituksessa käytettään indeksointia, jonka avulla oikeat nimikkeet pysyvät kuvien yhteydessä. Sekoituksen jälkeen kuvat jaetaan koulutusdataksi ja testidataksi. Yksittäisiä kuvia on käytössä 13 236. Kuvista otetaan noin 20 % testikäyttöön ja loput käytetään kouluttamiseen. Testikuvia on yhteensä 2 600 ja koulutuskuvia on 10 636.

Neuroverkolle koulutus- ja testidata syötetään muuttujien X, Y, test_x ja test_y alla. Liitteessä 5 on koko neuroverkon koulutus ja funktiot DATA ja shuffle_dataset tekevät datan sekoituksen ja jaottelun. (27)

7.4 Neuroverkon malli

Neuroverkkoa varten pitää ladata Tflearn kirjastosta moduulit

”import tflearn”

“from tflearn.layers.conv import conv_2d, max_pool_2d”

“from tflearn.layers.core import input_data, dropout, fully_connected”

“from tflearn.layers.estimator import regression”.

Mallin alussa muotoillaan X rivillä

”X = X.reshape([-1, 288, 512, 3])”.

Rivin ensimmäinen numero on -1, jolloin X pitää muotonsa, toinen numero on videon pystyresoluutio, kolmas numero on videon horisontaalinen resoluutio ja viimeinen numero on käytössä olevat värikanavat. Sama tehdään myös test_x:lle. Ensimmäinen kerros mallissa on syötekerros. Syötekerros on kooltaan sama kuin aiemmin muotoiltu X. Ensimmäisen kerroksen rivi on

”convnet = input_data(shape=[None, 288, 512, 3], name='input')”.

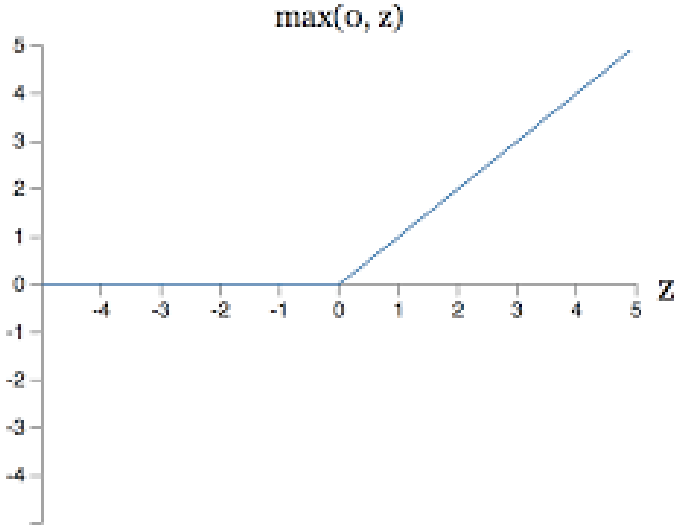
Syötekerrosta seuraa konvoluutiokerros, johon lisätään max pool -kerros.

”*convnet = conv_2d(convnet, 32, 2, activation='relu')”*

“convnet = max_pool_2d(convnet, 2)”.

Konvoluutiokerrokselle syötetään aikaisempi syötekerros, konvoluutiosuodattimia määritellään 32 ja ikkunan koko on 2. Aktivointifunktioina käytetään ReLU:a, joka tulee sanoista Rectified Linear Unit. ReLU:ssa tulo saa arvon nolla, mikäli syötteen arvo on nolla. Jos syöte on positiivinen, tulo saa saman positiivisen arvon. ReLU:n funktio on

$f(x) = \max(0, x)$ ja kuvaaja on kuvassa 12.



Kuva 12. Rectified Linear Unit (33).

Ensimmäisen konvoluutiokerroksen ja max pool -kerroksen perään lisätään toinen konvoluutio ja max pool -kerros. Konvoluutiokerroksen konvoluutiosuodattimien määrä tuplataan ja loput tiedot pysyvät samanlaisena. Toinen max pool -kerros on identtinen aikaisemman kanssa.

Konvoluutiokerroksien jälkeen mallille luodaan täysin yhdistetty kerros. Yhdistetylle kerrokselle syötetään aiempi kerros, määritellään 256 neuronia ja aktivointifunktiona toimii myös ReLU. Yhdistetyn kerroksen perään määritellään ”dropout”, millä pienennetään mallin ylisovitteisuuden mahdollisuutta. Dropoutille syötetään aikaisempi kerros ja määritellään, miten suuri määrä neuroverkon yhteyksistä pidetään prosentteina. Täysin yhdistetty kerros ja dropout kerros tapahtuvat riveillä

”convnet = fully_connected(convnet, 256, activation='relu')”

”convnet = dropout(convnet, 0.8)”.

Viimeinen kerros on tuloskerros. Aiempaan tapaan edellinen kerros syötetään tuloskerrokseen, mutta neuroneja on vain kolme kappaletta, mikä vastaa haluttuja luokituksen määriä. Aiemmasta poiketen viimeisessä kerroksessa aktivointifunktiona käytetään ”softmaxia”. Tuloskerros on myös täysin yhdistetty. Tuloskerroksen rivi näyttää seuraavalta

”convnet = fully_connected(convnet, 3, activation='softmax')”.

Softmax-funktio puristaa jokaisen yksikön tuloksen nollan ja ykkösen väliin. Softmax ottaa huomioon kaikki saadut tulokset ja yhdistää ne niin, että kaikkien tulosten yhteen laskettu summa on 1. Tällä tavalla saadaan neuroverkkoa kokeillessa jokaisen tuloksen todennäköisyys ja voidaan valita tuloksista suurin. Kuvassa 13 nähdään softmaxin perusidea.

Kuva 13. Softmax. (34)

Softmaxn matetaattinen funktion on

$$\sigma(z)_j = \frac{e^{z_j}}{\sum_{k=7}^{k} e^{z_k}},$$

jossa z on syötteiden ja tulon vektori, ja j on tulosyksiköiden indeksi.

Konvoluutiosyväneuroverkon mallin määrityksen viimeisessä rivissä määritellään taantuma (eng. regression). Taantumaan rivi näyttää seuraavalta

“convnet = regression(convnet, optimizer='adam', learning_rate=0.000001, loss='categorical_crossentropy', name='targets')”.

Taantumaan syötetään argumentit: edellinen kerros, optimoija, opintonopeus, loss ja viimeiseksi kohde. Optimoija on algoritmi, jonka perusteella neuroverkon painojen päivitys tapahtuu. Adam optimoija on TFlearnin oletus optimoija. Opintonopeus vaikuttaa verkon painojen säätöön loss-funktion perusteella. Liian suuri opintonopeus voi tehdä neuroverkosta alisovitteisen ja liian pienellä opintonopeudella ylisovitteisuuden riski kasvaa. Loss-funktiolla verkko luokittelee syötteestä saadut tulokset. Lopussa määritellään nimi, jolla liitetään nimikkeet osaksi neuroverkkoa.

Kaikista aiemmin määritetyistä kerroksista muodostetaan lopulta syväneuroverkkomalli rivillä

”model = tflearn.DNN(convnet)”.

Koulutusta varten säädetään asetukset rivillä

“model.fit({'input':X},{'targets':Y}, n_epoch=50, validation_set=({'input':test_x},{'targets':test_y}), batch_size=22, validation_batch_size=22, snapshot_step=500, show_metric=True, run_id='Raidillon')”.

Taulukossa 4 on selitykset asetuksille.

Taulukko 4. Model.fit selitykset.

{'input':X}, {'targets':Y}	Koulutukseen käytettävät kuvat ja nimikkeet
n_epoch	Koulutussyklien määrä
validation_set=({'input':test_x},{'targets':test_y}	Testaukseen käytettävät kuvat ja nimikkeet
batch_size	Kerralla koulutettavan datan määrä
validation_batch_size	Kerralla testattavan datan määrä
snapshot_step	Raportoi edistyksen tällä välillä
show_metric	Näyttää edistyksen
run_id	Verkon tunnus

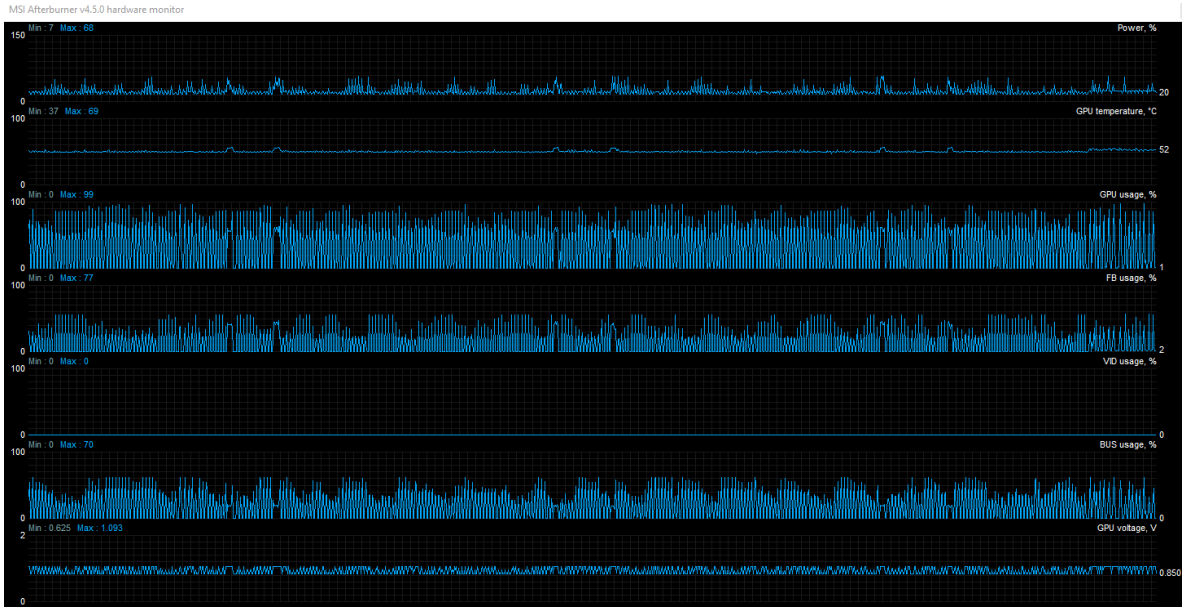
Viimeiseksi malli tallennetaan rivillä

”model.save("tflearnCNN.model")”.

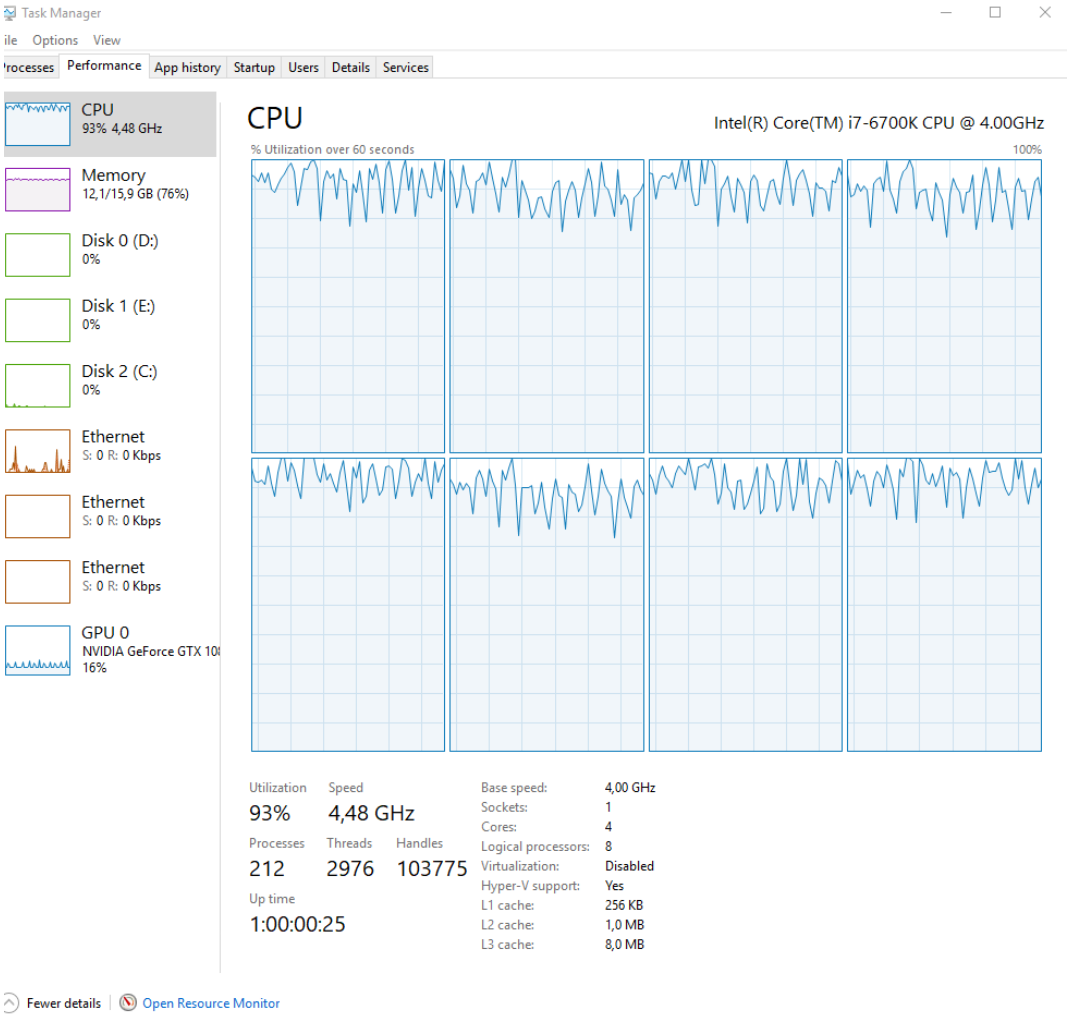
(26; 35; 36)

7.5 Koulutuksen tulos

Neuroverkon koulutuksessa kesti noin 5,5 tuntia. Mallin koulutus varasi käyttöjärjestelmästä muistia tausta prosessien ollessa käynnissä 12,1 Gt, ja näytönohjaimen muistista oli käytössä 7,4/8,0 Gt. Neuroverkon suuruutta tai koulutuserän suuruutta ei voinut kasvattaa, koska TFlearn antoi GPU:n muistilopussa virheilmoituksen. Kuvissa 14 ja 15 nähdään koulutuksen aikana tietokoneen resurssien käyttö.



Kuva 14. GPU-rasitus.



Kuva 15. Systeemin rasitus.

Näytönohjainta käytettiin aktiivisesti jatkuvasti, mutta käyttöaste vaihteli hyvin paljon. GPU-ytimien lisäksi kuvapuskuria ja PCIe-väylää käytettiin aktiivisesti koulutuksen aikana. Prosessorin keskimääräinen käyttö oli 93 %, ja kuvaaja käytöstä oli huomattavasti GPU:ta tasaisempi.

Koulutuksen alussa 7. epochin kohdalla mallin tarkkuus oli noussut jo 60 prosenttiin häviön ollessa keskimäärin noin 8. Kuvassa 16 näkyy koulutuksen edistyminen alussa.

```
Command Prompt - tf_CNN.py
| Adam | epoch: 007 | loss: 7.88387 - acc: 0.6386 -- iter: 06424/10636
[A[ATraining Step: 3197  | total loss: [1m[32m8.36643[0m[0m | time: 241.294s
| Adam | epoch: 007 | loss: 8.36643 - acc: 0.6156 -- iter: 06446/10636
[A[ATraining Step: 3198  | total loss: [1m[32m8.74715[0m[0m | time: 242.097s
| Adam | epoch: 007 | loss: 8.74715 - acc: 0.5950 -- iter: 06468/10636
[A[ATraining Step: 3199  | total loss: [1m[32m9.11069[0m[0m | time: 242.907s
| Adam | epoch: 007 | loss: 9.11069 - acc: 0.5764 -- iter: 06490/10636
[A[ATraining Step: 3200  | total loss: [1m[32m9.04056[0m[0m | time: 243.709s
| Adam | epoch: 007 | loss: 9.04056 - acc: 0.5778 -- iter: 06512/10636
[A[ATraining Step: 3201  | total loss: [1m[32m8.85893[0m[0m | time: 244.517s
| Adam | epoch: 007 | loss: 8.85893 - acc: 0.5882 -- iter: 06534/10636
[A[ATraining Step: 3202  | total loss: [1m[32m8.51796[0m[0m | time: 245.310s
| Adam | epoch: 007 | loss: 8.51796 - acc: 0.5976 -- iter: 06556/10636
[A[ATraining Step: 3203  | total loss: [1m[32m8.48832[0m[0m | time: 246.124s
| Adam | epoch: 007 | loss: 8.48832 - acc: 0.6015 -- iter: 06578/10636
[A[ATraining Step: 3204  | total loss: [1m[32m8.05824[0m[0m | time: 246.929s
| Adam | epoch: 007 | loss: 8.05824 - acc: 0.6231 -- iter: 06600/10636
[A[ATraining Step: 3205  | total loss: [1m[32m8.36713[0m[0m | time: 247.743s
| Adam | epoch: 007 | loss: 8.36713 - acc: 0.6108 -- iter: 06622/10636
[A[ATraining Step: 3206  | total loss: [1m[32m8.63998[0m[0m | time: 249.514s
| Adam | epoch: 007 | loss: 8.63998 - acc: 0.5997 -- iter: 06644/10636
[A[ATraining Step: 3207  | total loss: [1m[32m8.50418[0m[0m | time: 250.428s
| Adam | epoch: 007 | loss: 8.50418 - acc: 0.6079 -- iter: 06666/10636
```

Kuva 16. Koulutuksen edistyminen epoch 7.

Viimeisen epochin kohdalla mallin tarkkuus oli noussut 80,1 prosenttiin ja häviö oli laskenut arvoon 0,43. Kuvassa 17 on koulutuksen lopullinen tulos.

```
[A[ATraining Step: 24197  | total loss: [1m[32m0.46395[0m[0m | time: 389.591s
| Adam | epoch: 050 | loss: 0.46395 - acc: 0.7955 -- iter: 10582/10636
[A[ATraining Step: 24198  | total loss: [1m[32m0.45861[0m[0m | time: 390.382s
| Adam | epoch: 050 | loss: 0.45861 - acc: 0.8023 -- iter: 10604/10636
[A[ATraining Step: 24199  | total loss: [1m[32m0.44722[0m[0m | time: 391.172s
| Adam | epoch: 050 | loss: 0.44722 - acc: 0.8039 -- iter: 10626/10636
[A[ATraining Step: 24200  | total loss: [1m[32m0.43417[0m[0m | time: 398.317s
| Adam | epoch: 050 | loss: 0.43417 - acc: 0.8099 | val_loss: 0.55375 - val_acc: 0.7685 -- iter: 10636/10636
--
```

Kuva 17. Valmis koulutus.

Koulutuksen valmistuttua kohdekansioon on muodostunut 4 tiedostoa. Kansiossa on nyt mallin tiedosto, checkpoint, index ja metadata tiedostot. Mallin tiedosto on kooltaan 1,77 Gt. Malli tallentaa tiedostoon koulutetut painot, minkä takia tiedostoa käytettäessä malli pitää määritellä painoja varten uudestaan. Muut tiedostot ovat kooltaan 144 kt yhteensä. Koulutuksen tarkkuus vaikuttaa suurelta ottaen huomioon käytössä olevan datan määrä. On mahdollista, että käytetyillä asetuksilla malli on ylisovitteinen.

Koulutusta on mahdollista säätää lisäämällä mallille dataa, lisäämällä epochien määrää tai säätämällä opintonopeutta. Neuroverkon tarkkuus perinteisesti kasvaa erittäin nopeasti koulutuksen alussa, minkä jälkeen tarkkuuden kasvu tasaantuu ja hidastuu huomattavasti. Jokainen lisäprosentti tarkkuudessa vaatii huomattavasti enemmän dataa ja laskenta tehoa, mikäli koulutus halutaan tehdä järkevässä ajassa. Suuremmalla datan määrällä vältetään samalla koulutusdatan ylisovitusta.

8 Itsestään ajava auto

Autonomiselle kokonaisuudelle on tähän mennessä luotu neuroverkko, videon suoratoisto ja Raspberry ohjaus. Koska malli on 1,7 Gt, se on liian suuri Raspberryn käytettäväksi. Lisäksi Raspberryn pienempi laskentateho vaikuttaa negatiivisesti ennusteiden tekemiseen reaaliajassa. Mallia on siis parempi käyttää pöytätietokoneessa ja lähettää käskyt Raspberrylle suoritettavaksi. Huonona puolena on, että autonomisella autolla voi ainoastaan ajaa paikoissa, joissa Raspberry on mahdollista yhdistää laskentaa suorittavaan tietokoneeseen langattomasti netin välityksellä. Langaton yhdistäminen voi olla mahdollista kaikissa paikoissa missä Raspberryn voi yhdistää puhelimen 4G-verkkoon, mutta aikarajoituksista johtuen projektin autoa käytetään vain sisätiloissa yhteisessä langattomassa verkossa.

8.1 Arvauksista komennoiksi

Arvausta varten määritellään aluksi neuroverkko, jota käytetiin koulutuksessa. Määrityksen jälkeen ladataan koulutetut painot muistiin. Painojen lataus tehdään rivillä

”model.load("tflearnCNN.model")”.

Arvausten tekeminen tapahtuu silmukassa, jossa aluksi luetaan kuva Raspberrylta ja muokataan se verkolle sopivaksi. Seuraavaksi tehdään neuroverkolla arvaus rivillä

”predict = model.predict(frame)”.

Softmax-funktiosta johtuen arvaus tulee esimerkiksi muodossa

[[0.1555, 0.6833, 0.1711]].

Esimerkin voimakkain arvaus on mennä suoraan. Arvaukset muotoillaan uudestaan rivillä

“move = list(np.around(predict)[0])”.

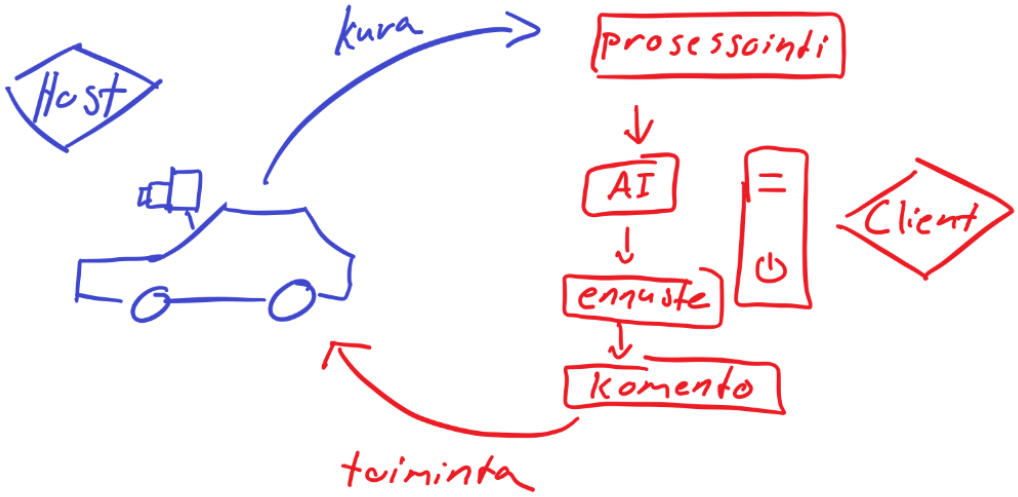
Tulokseksi saadaan lista, joka näyttää seuraavalta:

”move == [0,1,0]”.

Yksinkertaisella if-rakenteella märitetään, että komento on ”suoraan”. ja komento on nyt valmis lähettäväksi Raspberrylle. Komentojen tekeminen löytyy liitteestä 7. (36, 37)

8.2 Verkkokommunikointi

Toisin kuin koulutusdataa kerätessä käskyjen lähetyksessä ei ole tarvetta käyttää verkkoselainta ja komennot lähetetään suoraan Python-ohjelmasta koneelta Raspberrylle hyödyntäen Pythonin Socket-verkko-ohjelmointia. Socket-moduuli löytyy Pythonista suoraan, eikä sitä ole tarve asentaa erikseen. Raspberry toimii verkkokommunikoinnin isäntänä, joka ottaa asiakkaalta komennot vastaan. Asiakas on vastaavasti pöytätietokone. Kuvassa 18 on kuvattu yksinkertaisesti Raspberryn ja koneen välistä kommunikointia ja koneessa tapahtuvaa syötteen läpivientiä.



Kuva 18. Kommunikointikaavio.

Liitteessä 6 on koodi Host_drive.py-sovellukselle ja liitteessä 7 on koodi prediction_client.py-sovellukselle. Host_drive.py-sovellusta käytetään Raspberrylla. Sovellus määritellään serveriksi. Serverin IP-osoite on Raspberryn lähiverkon lokaali osoite ja käytössä olevaksi portiksi valitaan 5050. Sovellus odottaa yhteydenluontia, kunnes se alkaa ottamaan komentoja vastaan asiakkaalta. Komennot tulevat tekstimuodossa, jotka dekoodataan. Komennot lajitellaan yksinkertaisella if-rakenteella. Komennon suorittamiseen käytetään aikaisemmin tehtyä manuaali_ajo.py-sovelluksen ”Move”-luokkaa, joka tuodaan mukaan sovellukseen. Sovelluksessa määriteellään käytettäväksi Raspberryn GPIO-pinnit 12 ja 13. Serveri ottaa käskyjä vastaan niin kauan kuin asiakas lähettää niitä.

Predicton_client.py-asiakasohjelmassa otetaan yhteys serveriin, minkä jälkeen Raspberryn suoratoistosta napataan kuva. Kuva syötetään koulutettuun neuroverkkoon, joka antaa tulokseksi kolme arvausta. Arvauksista voimakkain muutetaan komennoksi, joka enkoodataan sekä lähetetään Raspberrylle suoritettavaksi. Arvauksien lähetystä jatketaan niin kauan, kunnes ohjelma keskeytetään tai suljetaan. (38; 39)

8.3 Auton käyttö

Auton ollessa yhdistettynä langattomaan verkkoon Rasberryllä käynnistetään Host_drive.py-sovellus. Raspberry odottaa prediction_client.py-sovelluksen yhdistämistä. Isäntäohjelman ja asiakasohjelman yhdistettyä, asiakas alkaa luoda arvauksia, jotka isäntä ottaa vastaan ja suorittaa ne. Auton toiminnan lopetus tapahtuu painamalla asiakassovelluksesta ”q”-näppäintä näppäimistössä tai tappamalla toisen ohjelmista.

8.4 Ongelmia

Projektin auton suurimmaksi ongelmaksi muodostuu auton koko suhteessa kameran näkökenttään. Kamera on sijoitettu auton etuakselin kohdalle parhaan näkyvyyden takaamiseksi, mutta johtuen suhteessa pienestä näkökentästä auto havaitsee esteet esimerkiksi sohvan tai seinä vasta, kun on liian myöhäistä tehdä kunnollista väistöliikettä ja auto jää jumiin. Siirtämällä kameraa taaemmaksi auto jäi harvemmin jumiin ja kykeni suorittamaan pidempiä matkoja itsenäisesti.

Autossa on myös erittäin tehokas sähkömoottori, ja ulkokäyttöön tehdyissä renkaissa on suuret kuviot, jotka yhdessä johtavat kameran heilumiseen ja tärinään varsinkin liikkeellelähdössä ja jonkin verran ajon aikana. Tämä muodostaa epätarkan kuvan autolle, mikä vaikeuttaa auton päätöksentekoa.

Auton käyttö langattomassa lähiverkossa 2,4 GHz:n taajuudella on kerrostaloissa ongelma ruuhkautuneen kaistan takia. Auto välillä pysähtyy lähiverkon lähetysongelmien johdosta. Muita ongelmia on auton liian tehokas moottori sisätiloihin käytettäväksi ja koosta johtuen suurehko kääntösäde.

9 Yhteenveto

Opinnäytetyön tavoitteena oli tutkia, mitä vaatimuksia ja haasteita autonomisen auton luominen tuottaa. Idea tuli minulta itseltäni johtuen mielenkiinnosta autojen mahdolliseen tulevaisuuden näkymään. Opinnäytetyö vaati huomattavaa määrää ohjelmointia, josta minulla ei ole paljoa kokemusta, joten olen erittäin tyytyväinen saadessani kokonaisuuden toimimaan. Opinnäytetyössä sivuttiin aiheita, jotka voisivat olla itsenäisesti kokonaisia aiheita opinnäytetöiksi.

Auto luotiin pienessä skaalassa ajamaan sisätiloissa lyhyttä rataa itsenäisesti. Tässä tavoitteessa onnistuttiin, mutta autonomisen auton tuominen julkiseen liikenteeseen vaatii huomattavasti enemmän työtä, dataa, sensoreita ja laskentatehoa, koska oikeassa lii-

kenteessä on huomattavasti enemmän muuttujia pieneen malliin verrattuna. Autoihin ollaan tuomassa lähitulevaisuudessa 5G-verkkotekniikkaa, joka osaltaan tulee varmasti parantamaan osittaista autonomista toimintaa. Jos haluamme autoista kuitenkin täysin itsestään ajavia, pitää laskenta mielestäni suorittaa lokaalisti autossa, koska jatkuvaa yhteyttä pilvipalveluihin joka tilanteessa on hankala saavuttaa. Auton pitää hankalissa tilanteissa sekä erilaisissa paikoissa pystyä tekemään päätökset täysin itsenäisesti tai emme voi puhua täysin autonomisesta autosta.

Opinnäytetyön autoa voisi jatkokehittää huomattavasti nykyistä paremmaksi. Valitsemalla kameran suuremmalla näkökentällä tai vaihtoehtoisesti asentamalla usean kameran voisi tietä ja esteitä nähdä entistä paremmin. Auton valitseminen käyttökohde huomioon ottaen olisi helpottanut myös auton itsenäistä ajamista. Paremmalla tietokoneella olisi voinut pystyä luomaan suuremman ja paremman neuroverkon. Opinnäytetyön auton moottorilla on kaksi eri asentoa, on ja off, ja ohjauksella kolme. Autoa pystysi jatkokehittämään niin, että moottorin nopeutta voisi kasvattaa ja vähentää tilanteesta riippuen ja auton ohjausta kääntää mutkasta riippuen asteittain ääriasentojen sijasta.

Opinnäytetyössä käytettiin hyväksi konenäköä ja neuroverkkoja, mutta melkein tärkein osa-alue työssä oli datan keräys ja sen prosessointi. Tämä jäi työn alussa huomioimatta keskittymisen kohdistuessa konenäköön, verkkotekniikkaan ja neuroverkkoihin. Tästä osa-alueesta iso kiitos kuuluu Ryan Zottille ja hänen luomalleen projektille, josta hyödynnettiin hänen tekemää datan keräystä ja prosessointia. Datan syöttöä neuroverkon käsiteltäväksi voisi kehittää eteenpäin. Tämänhetkinen tapa ladata kaikki ram-muistiin toimii pienellä datasetillä, mutta isojen datamäärien kanssa tulee nopeasti ongelmia muistimäärän kanssa mukaan lukien näytönohjaimen muistin kanssa. Erityismaininnan ansaitsee myös Harrison Kinsley, jonka opetusmateriaaleja käytin Python-ohjelmoinnin opetteluun.

Opinnäytetyössä käytiin lopulta erittäin paljon minulle täysin uusia asioita, mutta tuttuna puolena oli radio-ohjattava auto, jossa pääsin käyttämään autosähkötekniikan osaamistani auton muokkauksessa itsestään ajavaan muotoon. Vaikka parannettavaa löytyy melkein jokaiselta osa-alueelta, oli auton autonomiseksi tekeminen onnistunut operaatio.

Kuvassa 19 on opinnäytetyön valmis auto *”Raidillon”.*

Kuva 19. Raidillon.

Lähteet

1. Fridman, Lex. 2018. MIT 6.S094: Deep Learning for Self-Driving Cars. Lecture 2 Self-Driving Cars. Verkkoaineisto. MIT. https://selfdrivingcars.mit.edu/. Luettu 3.9.2018.

2. Automated Vehicles for Safety. Verkkoaineisto. NHTSA. https://www.nhtsa.gov/technology-innovation/automated-vehicles-safety. Luettu 11.11.2018.

3. 3.0 History of Neural Networks. Verkkoainesto. DOD DACS. http://www.psych.utoronto.ca/users/reingold/courses/ai/cache/neural4.html. Luettu 10.10.2018.

4. Parloff, Roger. 2016. Why Deep Learning Is Suddenly Changing Your Life.. Verkkoainesto. Fortune. http://fortune.com/ai-artificial-intelligence-deep-machine-learning/. Luettu 5.10.2018.

5. Arnoud, Sacha. 2018. MIT 6.S094: Deep Learning for Self-Driving Cars. Guest Talk Sacha Arnoud, Director of Engineering. Waymo. Verkkoaineisto. MIT. https://selfdrivingcars.mit.edu/. 7.2.2018. Luettu 8.9.2018.

6. Hua, Quan. Ul Azeem. Sham & Ahmed Saif. 2017. Machine Learning with TensorFlow 1.x. Birmingham UK: Packt Publishing.

7. Sau, Tapan. 2015. Artificial Neuron Model. Verkkoaineisto. https://www.researchgate.net/figure/Artificial-Neuron-model_fig4_277774116. Luettu 6.1.2019

8. Valkov, Venelin, 2017. Creating a Neural Network from Scratch –TensorFlow for Hackers (Part IV). Verkkoaineisto. https://medium.com/@curiousily/tensorflow-for-hackers-part-iv-neural-network-from-scratch-1a4f504dfa8. Luettu 11.11.2018.

9. Van Boxel, Dan. 2016. Deep Learning With Tensorflow. 2.1 Basic Neural Nets. Birmingham UK: Packt Publishing.

10. Fridman, Lex. 2018. MIT 6.S094: Deep Learning for Self-Driving Cars. Lecture 1 Deep Learning. Verkkokaineisto. MIT. https://selfdrivingcars.mit.edu/. Luettu 3.9.2018.

11. Maverick Strada MT Instruction Manual. Verkkoaineisto. Manualslib. https://www.manualslib.com/manual/1046172/Maverick-Strada-Mt.html?page=4#manual. Luettu 12.2.2018.

12. Raspberry Pi Model 3 B. Verkkoaineisto. Stontronics. https://www.alliedelec.com/m/d/4252b1ecd92888dbb9d8a39b536e7bf2.pdf. Luettu 6.4.2018.

13. Rasbian. Verkkoaineisto. Raspberry Pi Foundation. https://www.raspberrypi.org/downloads/raspbian/. Luettu 3.3.2018.

14. Capacity (SD/SDHC/SDXC/SDUC). Verkkoainesto. SD Association. ttps://www.sdcard.org/developers/overview/capacity/index.html. Luettu 13.11.2018.

15. NOOBS. Verkkoainesto. Raspberry Pi Foundation. https://www.raspberrypi.org/documentation/installation/noobs.md. Luettu 3.3.2018.

16. Matt. 2012. Simple Guide to the Raspberry Pi GPIO Header and Pins. Verkkoainesto. Raspberry Pi Spy. https://www.raspberrypi-spy.co.uk/2012/06/simple-guide-to-the-rpi-gpio-header-and-pins/#prettyPhoto. Luettu 15.3.2018.

17. GPIO. Verkkoainesto. Raspberry Pi Foundation. https://www.raspberrypi.org/documentation/usage/gpio/README.md. Luettu 21.3.2018

18. Cook, Mike. Understanding Outputs. Verkkoaineisto. http://www.thebox.myzen.co.uk/Raspberry/Understanding_Outputs.html. Luettu 12.4.2018.

19. Lazorenko, Andriy. 2017. TensorFlow performance test: CPU VS GPU. Verkkoaineisto. https://medium.com/@andriylazorenko/tensorflow-performance-test-cpu-vs-gpu-79fcd39170c. Luettu 6.12.2018.

20. NumPy. Verkkoaineisto. Python Software Foundation. https://pypi.org/project/numpy/. Luettu 13.4.2018.

21. Mordvintsev, Alexander & Abid, K. 2013. OpenCV-Python Tutorials. Verkkoaineisto. https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_tutorials.html. Luettu 18.4.2018.

22. Tornado Web Server. Verkkoaineisto. The Tornado Authors. https://www.tornadoweb.org/en/stable/guide.html. Luettu 7.3.2018.

23. Socket — Low-level networking interface. Verkkoaineisto. Python Software Foundation. https://docs.python.org/3.6/library/socket.html. Luettu 18.10.2018.

24. RPi-Cam-Web-Interface. 2018. Verkkoaineisto. Embedded Linux Wiki. https://elinux.org/RPi-Cam-Web-Interface. Luettu 20.4.2018.

25. GPU support. 2018. Verkkoaineisto. Tensorflow. https://www.tensorflow.org/install/gpu. Luettu 3.5.2018.

26. Installation. Verkkoaineisto. TFLearn. http://tflearn.org/getting_started/. Luettu 15.10.2018.

27. Zotti, Ryan. RyanZotti/Self-Driving-Car. 2018. Verkkoaineisto. https://github.com/RyanZotti/Self-Driving-Car. Luettu 5.4.2018.

28. Mordvintsev, Alexander & Abid, K. 2013. Getting Started with Videos. Verkkoaineisto. Open-CV.

https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_gui/py_video_display/py_video_display.html#display-video. Luettu 19.4.2018.

29. Van Boxel, Dan. 2016. Deep Learning With Tensorflow. 3.1 Convolutional Layer Motivation. Birmingham UK: Packt Publishing.

30. Fridman, Lex . 2018. MIT 6.S094: Deep Learning for Self-Driving Cars. Lecture 4 Computer Vision. Verkkoaineisto. MIT. https://selfdrivingcars.mit.edu/. Luettu 7.9.2018.

31. Van Boxel, Dan. 2016. Deep Learning With Tensorflow. 3.3 Pooling Layer Motivation. Birmingham UK: Packt Publishing.

32. Van Boxel, Dan. 2016. Deep Learning With Tensorflow. 3.4 Pooling Layer Application. Birmingham UK: Packt Publishing.

33. ReLU. 2017. Verkkoaineisto. Stack Overflow. https://stackoverflow.com/questions/43504248/what-does-relu-stand-for-in-tf-nn-relu. Luettu 11.11.2018.

34. Yang, Ji. 2017. ReLU and Softmax Activation Functions. Verkkoaineisto. https://github.com/Kulbear/deep-learning-nano-foundation/wiki/ReLU-and-Softmax-Activation-Functions. Luettu 2.11.2018

35. Kinsley, Harrison. 2016. Convolutional Neural Network CNN with TensorFlow tutorial. Verkkoaineisto. https://pythonprogramming.net/cnn-tensorflow-convolutional-nerual-network-machine-learning-tutorial/?completed=/convolutional-neural-network-cnn-machine-learning-tutorial/. Luettu 12.9.2018.

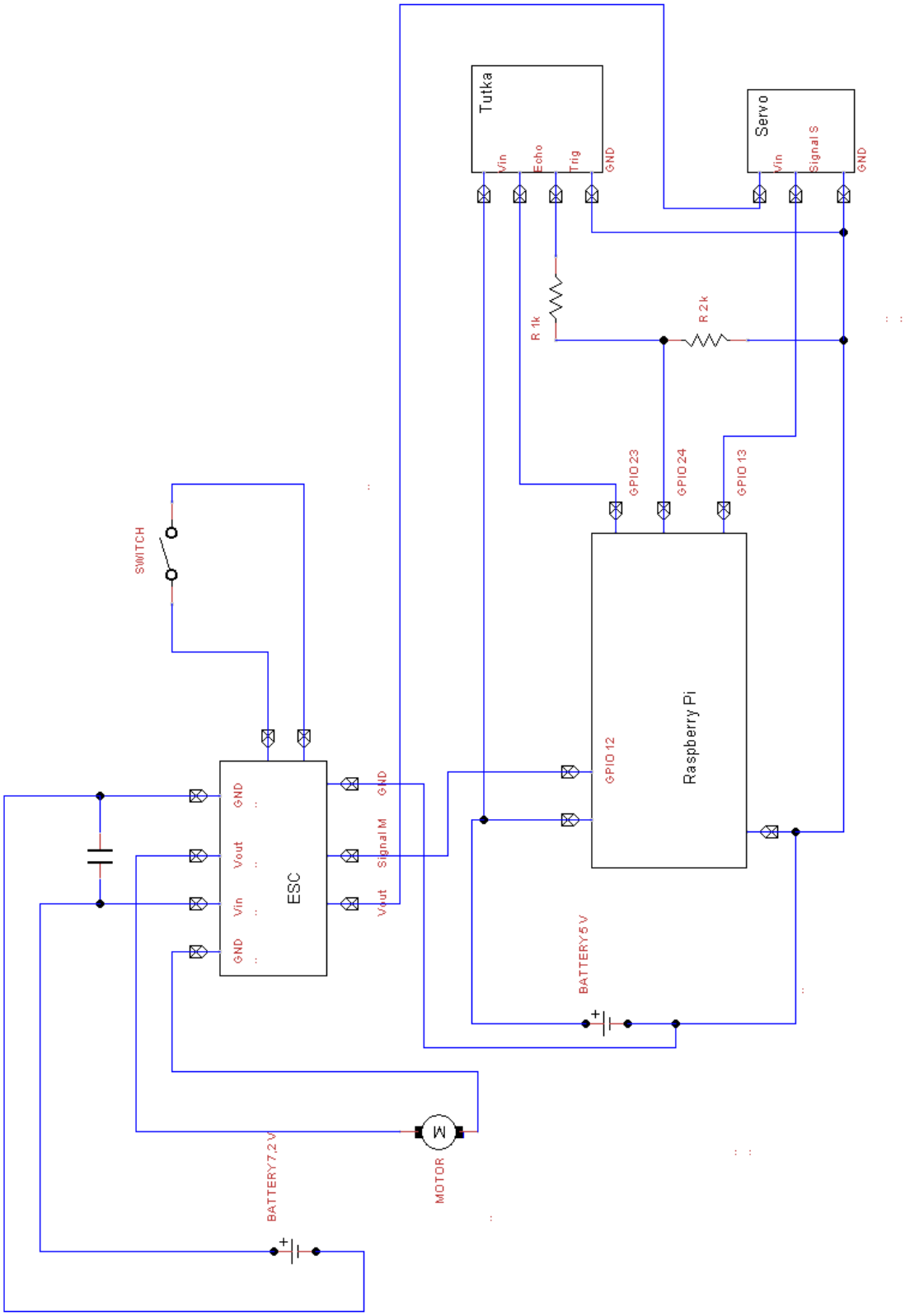
36. Kinsley, Harrison. 2016. TFLearn - High Level Abstraction Layer for TensorFlow Tutorial. Verkkoaineisto. https://pythonprogramming.net/tflearn-machine-learning-tutorial/?completed=/cnn-tensorflow-convolutional-nerual-network-machine-learning-tutorial/. Luettu 12.9.2018.

37. Kinsley, Harrison. 2017. Testing self-driving car neural network- Python Plays GTA V. Verkkoaineisto. https://pythonprogramming.net/testing-self-driving-car-neural-network-python-plays-gta-v/. Luettu 22.4.2018.

38. Kinsley, Harrison 2014. Binding and Listening with Sockets. Verkkoaineisto. https://pythonprogramming.net/python-binding-listening-sockets/?completed=/python-threaded-port-scanner/. Luettu 20.10.2018.

39. Kinsley, Harrison 2014. Client Server System with Sockets. Verkkoaineisto. https://pythonprogramming.net/client-server-python-sockets/?completed=/python-binding-listening-sockets/. Luettu 20.10.2018.

Piirikaavio



Manuaaliajo.py

```
import argparse
import tornado.ioloop
import tornado.web
from datetime import datetime
import os
from operator import itemgetter
import RPi.GPIO as GPIO
import requests
from time import sleep

class PostHandler(tornado.web.RequestHandler):

    @property
    def settings(self):
        return self._settings

    @settings.setter
    def settings(self,settings):
        self._settings = settings

    def initialize(self, settings):
        self.settings = settings

    def post(self):
        timestamp = datetime.now()
        data_json = tornado.escape.json_decode(self.request.body)
        allowed_commands = set(['37','38','39','40'])
        command = data_json['command']
        command = list(command.keys())
        command = set(command)
        command = allowed_commands & command
        file_path = str(os.path.dirname(os.path.realpath(__file__)))+"/session.txt"
        log_entry = str(command)+" "+str(timestamp)
```

```
        log_entries.append((command,timestamp))
        with open(file_path,"a") as writer:
            writer.write(log_entry+"\n")
        print(log_entry)
        speed = self.settings['speed']
        if '37' in command:
            move.forward_left()
        elif '38' in command:
            move.forward()
        elif '39' in command:
            move.forward_right()
        elif '40' in command:
            move.backward()
        else:
            move.stop()

class StoreLogEntriesHandler(tornado.web.RequestHandler):
    def get(self):
        file_path = str(os.path.dirname(os.path.realpath(__file__)))+"/clean_session.txt"
        sorted_log_entries = sorted(log_entries,key=itemgetter(1))
        prev_command = set()
        allowed_commands = set(['38','37','39','40'])
        for log_entry in sorted_log_entries:
            command = log_entry[0]
            timestamp = log_entry[1]
            if len(command ^ prev_command) > 0:
                prev_command = command
                with open(file_path,"a") as writer:
                    readable_command = []
                    for element in list(command):
                        if element == '37':
                            readable_command.append("left")
                        if element == '38':
                            readable_command.append("up")
                        if element == '39':
```

```
                    readable_command.append("right")
                if element == '40':
                    readable_command.append("down")
            log_entry = str(list(readable_command))+" "+str(timestamp)
            writer.write(log_entry+"\n")
        print(log_entry)
    self.write("Finished")

class MultipleKeysHandler(tornado.web.RequestHandler):

  def get(self):
    print("HelloWorld")
    self.write('''
        <!DOCTYPE html>
        <html>
            <head>
                <script                                                src="https://ajax.goog-
leapis.com/ajax/libs/jquery/1.12.2/jquery.min.js"></script>
                <script>
                    var keys = {};

                    $(document).keydown(function (e) {
                        keys[e.which] = true;

                        var json_upload = JSON.stringify({command:keys});
                        var xmlhttp = new XMLHttpRequest();
                        xmlhttp.open("POST", "/post");
                        xmlhttp.setRequestHeader("Content-Type",      "application/x-www-
form-urlencoded");
                        xmlhttp.send(json_upload);

                        printKeys();
                    });

                    $(document).keyup(function (e) {
```

```
                    delete keys[e.which];

                    var json_upload = JSON.stringify({command:keys});
                    var xmlhttp = new XMLHttpRequest();
                    xmlhttp.open("POST", "/post");
                    xmlhttp.setRequestHeader("Content-Type",      "application/x-www-
form-urlencoded");
                    xmlhttp.send(json_upload);

                    printKeys();
                });

                function printKeys() {
                    var html = '';
                    for (var i in keys) {
                        if (!keys.hasOwnProperty(i)) continue;
                        html += '<p>' + i + '</p>';
                    }
                    $('#out').html(html);
                }

            </script>
        </head>
        <body>
            Click in this frame, then try holding down some keys
            <div id="out"></div>
        </body>
    </html>
"''')

class Move:

    def __init__(self, pinServo, pinMotor):

        self.pinServo = pinServo
```

```
        self.pinMotor = pinMotor

        GPIO.setup(self.pinMotor, GPIO.OUT)
        GPIO.setup(self.pinServo, GPIO.OUT)

        self.pwm_motor = GPIO.PWM(self.pinMotor, 50)
        self.pwm_servo = GPIO.PWM(self.pinServo, 50)

        self.motor_idle = 0
        self.motor_forward = 8.2
        self.motor_back = 5.7

        self.servo_idle = 7.5
        self.servo_right = 6.0
        self.servo_left = 8.8

        self.pwm_motor.start(6.6)
        self.pwm_servo.start(7.5)

    def forward(self):

        self.pwm_motor.ChangeDutyCycle(self.motor_forward)
        self.pwm_servo.ChangeDutyCycle(self.servo_idle)

    def forward_left(self):

        self.pwm_motor.ChangeDutyCycle(self.motor_forward)
        self.pwm_servo.ChangeDutyCycle(self.servo_left)

    def forward_right(self):

        self.pwm_motor.ChangeDutyCycle(self.motor_forward)
        self.pwm_servo.ChangeDutyCycle(self.servo_right)

    def backward(self):
```

```
        self.pwm_motor.ChangeDutyCycle(self.motor_back)

    def left(self):

        self.pwm_servo.ChangeDutyCycle(self.servo_left)

    def right(self):

        self.pwm_servo.ChangeDutyCycle(self.servo_right)

    def stop(self):

        self.pwm_motor.ChangeDutyCycle(0)
        self.pwm_servo.ChangeDutyCycle(self.servo_idle)

def make_app(settings):
    return tornado.web.Application([
        (r"/drive",MultipleKeysHandler),(r"/post", PostHandler, {'settings':settings}),
        (r"/StoreLogEntries",StoreLogEntriesHandler)
    ])

if __name__ == "__main__":


    ap = argparse.ArgumentParser()
    ap.add_argument("-s", "--speed_percent", required=False, help="Between 0 and
100")

    args = vars(ap.parse_args())
    GPIO.setmode(GPIO.BCM)
    move = Move(12, 13)
    log_entries = []
    settings = {'speed':float(args['speed_percent'])}
    app = make_app(settings)
```

```
app.listen(5555)
tornado.ioloop.IOLoop.current().start()
```

tallenna_striimi.py

```
import cv2
import numpy as np
from datetime import datetime
import os

ip = "http://192.168.1.105/html/cam_pic.php?"
file_path = str(os.path.dirname(os.path.realpath(__file__)))+"/video_timestamps.txt"
fourcc = cv2.VideoWriter_fourcc(*'jpeg')
out = cv2.VideoWriter('output.mov',fourcc, 20.0, (512,288))
## Väärä resoluutio aiheutti korruptoimisen

while True:
    cap = cv2.VideoCapture(ip)
    ret, frame = cap.read()
    nyt = datetime.now()
    print(nyt)
    if ret==True:

        out.write(frame)
        timestamp = datetime.now()
        with open(file_path,"a") as writer:
            writer.write(str(timestamp)+"\n")

        cv2.imshow('frame',frame)
        if cv2.waitKey(1) & 0xFF == ord('q'):
            break
    else:
        break

cap.release()
out.release()
cv2.destroyAllWindows()
```

save_all_runs_as_numpy_files.py

```
from dataprep import video_to_rgb_npz, make_gamma_tables, process_session
import numpy as np
import os

def write_metadata(input_dir_path):
    input_file_path = input_dir_path + '/predictors_and_targets.npz'
    npzfile = np.load(input_file_path)
    image_count = len(npzfile['predictors'])
    metadata_file_path = input_dir_path + '/metadata.txt'
    with open(metadata_file_path, 'w') as writer:
        writer.write('image_count:' + str(image_count) + '\n')


data_path = 'C:/Users/Markus/Documents/Insinoorityo/data'
data_folders = os.listdir(data_path)
gamma_map = make_gamma_tables([1])
rgb = True

for folder in data_folders:
    cmd = 'dir '+data_path + '/' + folder
    dir_contents = 'C:/Users/Markus/Documents/Insinoorityo/data/1'
    print("Started work on "+str(folder))
    print(dir_contents)
    input_dir_path = data_path + '/' + folder
    if 'predictors_and_targets.npz' not in dir_contents:
        predictors, targets = process_session(data_path + '/' + folder, gamma_map, rgb)
        video_to_rgb_npz(input_dir_path,predictors,targets)
        print("Completed work on: "+str(folder)+". Created new npz and metadata files.")
        write_metadata(input_dir_path)
    elif 'metadata.csv' not in dir_contents:
        write_metadata(input_dir_path)
        print("Added only metadata file for: " + str(folder) + ".")
```

```
    else:
        print("Completed work on "+str(dir)+". File already exists. No processing necessary.")
print("Finished.")

    cv2.destroyAllWindows()
    return predictors, targets
def video_to_rgb_npz(session_path,predictors,targets):
    np.savez(session_path + '/predictors_and_targets', predictors=predictors,targets=targets)
```

tf_CNN.py

```
import tflearn
from tflearn.layers.conv import conv_2d, max_pool_2d
from tflearn.layers.core import input_data, dropout, fully_connected
from tflearn.layers.estimator import regression
import numpy as np

def shuffle_dataset(predictors, targets):
    record_count = predictors.shape[0]
    shuffle_index = np.arange(record_count)
    np.random.shuffle(shuffle_index)
    predictors = predictors[shuffle_index]
    targets = targets[shuffle_index]
    return predictors, targets

def DATA(data_path):
    #datan nouto ja jaottelu

    images = []
    labels = []
    npzfile = np.load(data_path)
    images.extend(npzfile["predictors"])
    labels.extend(npzfile["targets"])
    images = np.array(images)
    labels = np.array(labels)

    images,labels = shuffle_dataset(images,labels)
    train_x, test_x = images[:-2600], images[-2600:]
    train_y, test_y = labels[:-2600], labels[-2600:]
    return train_x, train_y, test_x, test_y


data = "C:/Users/Markus/Documents/Insinoorityo/data/1/predictors_and_targets.npz"
X, Y, test_x, test_y = DATA(data)
```

```
X = X.reshape([-1, 288, 512, 3])
test_x = test_x.reshape([-1, 288, 512, 3]) # -1 pitää muodon oikeana, että tensorflow on tyytyväinen / loput reso + värikanavat

convnet = input_data(shape=[None, 288, 512, 3], name='input')
convnet = conv_2d(convnet, 32, 2, activation='relu')
convnet = max_pool_2d(convnet, 2)

convnet = conv_2d(convnet, 64, 2, activation='relu')
convnet = max_pool_2d(convnet, 2)

convnet = fully_connected(convnet, 256, activation='relu')
convnet = dropout(convnet, 0.8) # pienennetään ylisovitteisuuden mahdollisuutta

convnet = fully_connected(convnet, 3, activation='softmax') # Kokeile muuttaa numeroa.
convnet = regression(convnet, optimizer='adam', learning_rate=0.000001, loss='categorical_crossentropy', name='targets')


model = tflearn.DNN(convnet)

model.fit({'input':X},{'targets':Y}, n_epoch=50, validation_set=({'input':test_x},{'targets':test_y}), batch_size=22, validation_batch_size=22,
        snapshot_step=500, show_metric=True, run_id='Raidillon')
model.save("tflearnCNN.model")
```

Host_drive.py

```
import socket
import sys
from _thread import *
import RPi.GPIO as GPIO
from manuaali_ajo import Move
import time

GPIO.setmode(GPIO.BCM)
move = Move(12, 13)
move.stop()

host = "192.168.1.105"
port = 5050
server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)



try:
    server.bind((host, port))
except socket.error as e:
    print(e)

server.listen(5)
print("Odottaa yhteyttä")
def threaded_client(conn):
    while True:
        data = conn.recv(4096)
        data = data.decode("utf-8")

        reply = "Serveri vastaus: " + data
        if not data:
            break
        elif data == "right":
```

```
            move.forward_right()
            move.forward_right()
##              print(data)

        elif data == "forward":
##              print(data)
            move.forward()
            move.forward()

        elif data == "left":
            move.forward_left()
            move.forward_left()

##              print(data)

        time.sleep(0.11)

    conn.close()
    GPIO.cleanup()
while True:
    conn, addr = server.accept()
    print("Yhteys luotu osoitteeseen: "+addr[0]+":"+str(addr[1]))
    start_new_thread(threaded_client,(conn,))
```

prediction_client.py

```
import socket
import sys
import time
import cv2
import tflearn
from tflearn.layers.conv import conv_2d, max_pool_2d
from tflearn.layers.core import input_data, dropout, fully_connected
from tflearn.layers.estimator import regression
import time
import numpy as np

ip = "http://192.168.1.105/html/cam_pic.php?"
# mallin määritys
convnet = input_data(shape=[None, 288, 512, 3], name='input')
convnet = conv_2d(convnet, 32, 2, activation='relu')
convnet = max_pool_2d(convnet, 2)
convnet = conv_2d(convnet, 64, 2, activation='relu')
convnet = max_pool_2d(convnet, 2)
convnet = fully_connected(convnet, 256, activation='relu')
convnet = dropout(convnet, 0.8)
convnet = fully_connected(convnet, 3, activation='softmax')
convnet = regression(convnet, optimizer='adam', learning_rate=0.000001, loss='cate-
gorical_crossentropy', name='targets')
model = tflearn.DNN(convnet)
##malli = model.load("tflearnCNN.model")
#ladataan koulutetut painot
model.load("tflearnCNN.model")

#yhteyden luonti raspiin
server = "192.168.1.105" ## MUUTA OIKEA IP
client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
port = 5050
```

```
client.connect((server, port))
print("Yhteys luotu / Connected")

while True:
    # haetaan raspberry kameran kuva
    cap = cv2.VideoCapture(ip)
    ret, frame = cap.read()
    cv2.imshow("striimi", frame)
    # kuva muotoillaan CNN:lle sopivaksi
    frame = frame.reshape([-1, 288, 512, 3])
    # tehdään arvaus
    predict = model.predict(frame)
    # muotoillaan arvaus listaksi
    move = list(np.around(predict)[0])

    # openCV vaatima rimpsu, että video toimii oikein
    if cv2.waitKey(1) & 0xFF == ord("q"):
        break

    if move == [1,0,0]:
        print("vasemmalle\n\n")
        order = "left"
    elif move == [0,1,0]:
        print("suoraan\n\n")
        order = "forward"
    elif move == [0,0,1]:
        print("oikealle\n\n")
        order = "right"
    else:
        continue
    client.send(order.encode())
    time.sleep(0.05)

client.close()
```

dataprep.py

```
import numpy as np
import cv2
import re
import os
from datetime import datetime
import collections

def make_gamma_tables(gammas):
    gamma_map = collections.OrderedDict()
    for gamma in gammas:

        invGamma = 1.0 / gamma
        table = np.array([((i / 255.0) ** invGamma) * 255
                    for i in np.arange(0, 256)]).astype("uint8")
        gamma_map[gamma] = table
    return gamma_map

def adjust_gamma(image, table):
                # apply gamma correction using the lookup table
                return cv2.LUT(image, table)

def process_session(session_path,gamma_map,rgb=True):



    up_arrow, left_arrow, right_arrow = read_arrow_key_images()

    cap = cv2.VideoCapture(session_path + "/output.mov")
    video_timestamps = []
    with open(session_path + '/video_timestamps.txt') as video_timestamps_reader:
        for line in video_timestamps_reader:
            line = line.replace("\n", "")
            ts = datetime.strptime(line, '%Y-%m-%d %H:%M:%S.%f')
            video_timestamps.append(ts)
```

```
commands = []
with open(session_path + '/clean_session.txt') as clean_session_reader:
    for line in clean_session_reader:
        line = line.replace("\n", "")

        # Ignore / skip all invalid commands
        if "down" in line:
            continue
        if "left" in line and "right" in line:
            continue
        if "left" in line and "up" in line:
            continue
        if "right" in line and "up" in line:
            continue
        if "left" in line and "up" in line and "right" in line:
            continue

        match = re.match(r"^.*\['(.*)'\].*$", line)
        if match is not None:
            command = match.group(1)
        else:
            command = 'no command'
        raw_ts = line[line.index(" ") + 1:]
        ts = datetime.strptime(raw_ts, '%Y-%m-%d %H:%M:%S.%f')
        commands.append([command, ts])

# time after which no other data is relevant because driving session has ended
end_time = commands[len(commands) - 1][1]

# cleanup to track only command transitions
compact_commands = []
prev_command = None
for item in commands:
    command, ts = item[0], item[1]
```

```
        if command != prev_command and command != 'no command' and command !=
'down':
            compact_commands.append([command, ts])
            prev_command = command
    commands = compact_commands

    # time before which no other data is relevant because driving session just started
    start_time = commands[0][1]

    current_command = commands[0][0]
    command_counter = 1

    # Fixes bug that arises when only one command type is used the whole session
    future_command = None
    future_command_ts = None
    if command_counter < len(commands):
        future_command = commands[command_counter][0]
        future_command_ts = commands[command_counter][1]
    else:
        future_command = "END"
        future_command_ts = end_time

    predictors = []
    targets = []

    frame_counter = -1
    while (cap.isOpened()):
        frame_counter = frame_counter + 1
        ret, frame = cap.read()
        if cv2.waitKey(1) & 0xFF == ord('q'):
            break
        if frame_counter == len(video_timestamps):
            break  # Fixes bug where video timestamps not as long as video frames
        video_timestamp = video_timestamps[frame_counter]
        if video_timestamp > start_time:
```

```
if video_timestamp < end_time:
    if video_timestamp > future_command_ts:
        current_command = future_command
        command_counter = command_counter + 1
        if command_counter < len(commands):
            future_command = commands[command_counter][0]
            future_command_ts = commands[command_counter][1]
        else:
            future_command = "END"
            future_command_ts = end_time

    target = [0, 0, 0]  # in order: left, up, right
    key_image = get_key_image(current_command)
    if current_command == 'left':
        target[0] = 1
    elif current_command == 'up':
        target[1] = 1
    elif current_command == 'right':
        target[2] = 1

    if rgb == True:
        for gamma, gamma_table in gamma_map.items():
            gamma_image = adjust_gamma(frame, gamma_table)
            targets.append(target)
            predictors.append(gamma_image)
    else:
        for gamma, gamma_table in gamma_map.items():
            bw_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
            #half_image = bw_frame[120:, :]
            #cv2.imshow(str(gamma), half_image)
            #print(half_image.shape)
            #bw_frame = half_image
            gamma_image = adjust_gamma(bw_frame, gamma_table)
            #gamma_image = np.reshape(gamma_image, [120, 320, 1])
            gamma_image = np.reshape(gamma_image, [288, 512, 1])
```

```
                    targets.append(target)
                    predictors.append(gamma_image)
                    #cv2.imshow(str(gamma), gamma_image)



            show_image_with_command(frame, key_image)

    cap.release()
    cv2.destroyAllWindows()
    return predictors, targets


def data_prep(data_path,rgb=True):

    #gamma_map = make_gamma_tables(np.arange(1.0,3.0,0.5))
    gamma_map = make_gamma_tables([1])

    data_folders = os.listdir(data_path)
    data_folders = sanitize_data_folders(data_folders)
    shuffle(data_folders)
    #data_folders = data_folders[:10]
    train_folder_size = int(len(data_folders) * 0.8)

    train_predictors = []
    train_targets = []
    for folder in data_folders[:train_folder_size]:
        print("Started session: " + str(folder))
        predictors, targets = process_session(data_path+'/'+folder,gamma_map,rgb)
        train_predictors.extend(predictors)
        train_targets.extend(targets)
        print("Completed session: "+str(folder))
    train_predictors_np = np.array(train_predictors)
    train_targets_np = np.array(train_targets)
```

```
    validation_predictors = []
    validation_targets = []
    for folder in data_folders[train_folder_size:]:
        print("Started session: " + str(folder))
        predictors, targets = process_session(data_path + '/' + folder,gamma_map,rgb)
        validation_predictors.extend(predictors)
        validation_targets.extend(targets)
        print("Completed session: " + str(folder))
    validation_predictors_np = np.array(validation_predictors)
    validation_targets_np = np.array(validation_targets)

    max_folder = max([int(folder) for folder in data_folders])
    new_file_name = None
    if rgb:
        new_file_name = '/data_rgb_'+str(max_folder)
    else:
        new_file_name = '/data_bw_'+str(max_folder)
    np.savez(data_path+new_file_name, train_predictors=train_predictors_np,
            train_targets=train_targets_np,validation_predictors = validation_predictors_np,
            validation_targets = validation_targets_np)


# Built so that a 3D convolution doesn't span multiple sessions (since there is no continuity between sessions)
def video_to_rgb_npz(session_path,predictors,targets):
    np.savez(session_path + '/predictors_and_targets', predictors=predictors,targets=targets)


def show_image_with_command(frame, key_image):

    arrow_key_scale = 0.125
    resized_image = cv2.resize(key_image, None, fx=arrow_key_scale, fy=arrow_key_scale, interpolation=cv2.INTER_CUBIC)
```

```
    # Thresholding requires grayscale only, so that threshold only needs to happen in one
dimension
    img2gray = cv2.cvtColor(resized_image, cv2.COLOR_BGR2GRAY)


    ret, mask = cv2.threshold(img2gray, 240, 255, cv2.THRESH_BINARY)


    mask_inv = cv2.bitwise_not(mask)  # invert the mask
    rows, cols, channels = resized_image.shape  # get size of image
    region_of_interest = frame[0:rows, 0:cols]
    img1_bg = cv2.bitwise_and(region_of_interest, region_of_interest, mask=mask)
    img2_fg = cv2.bitwise_and(resized_image, resized_image, mask=mask_inv)
    dst = cv2.add(img1_bg, img2_fg)  # ???
    frame[0:rows, 0:cols] = dst

    # Finally, show image with the an overlay of identified target key image
    cv2.imshow('frame', frame)


def read_arrow_key_images():
    image_path  =  str(os.path.dirname(os.path.realpath(__file__)))  +  "/arrow_key_im-
ages"
    up_arrow = cv2.imread(image_path + '/UpArrow.tif')
    left_arrow = cv2.imread(image_path + '/LeftArrow.tif')
    right_arrow = cv2.imread(image_path + '/Right Arrow.tif')
    return up_arrow, left_arrow, right_arrow


def get_key_image(label):
    up_arrow, left_arrow, right_arrow = read_arrow_key_images()
    key_image = None
    if label == 'left':
        key_image = left_arrow
    elif label == 'up':
```

```
        key_image = up_arrow
    elif label == 'right':
        key_image = right_arrow
    return key_image


def get_key_image_from_array(label):
    up_arrow, left_arrow, right_arrow = read_arrow_key_images()
    key_image = None
    if label[0][0] == 1:
        key_image = left_arrow
    elif label[0][1] == 1:
        key_image = up_arrow
    elif label[0][2] == 1:
        key_image = right_arrow
    return key_image
```